Chapitre 4

Langages réguliers et hors-contexte

Alphabets et mots

Définition 4.1

Un alphabet est un ensemble fini et non vide de symboles.

Exemple 4.2

$$
\begin{aligned}
\Sigma_1 &= \{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\} \\
\Sigma_2 &= \{\mathtt{a}, \mathtt{b}\} \\
\Sigma_3 &= \{\mathtt{0}, \mathtt{1}\} \\
\Sigma_4 &= \{\mathtt{0}, \mathtt{1}, \ldots, \mathtt{9}\} \\
\Sigma_5 &= \{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\} \\
\Sigma_6 &= \{\mathtt{0}, \mathtt{1}, \ldots, 9, +, -, *, /, (,)\} \\
\Sigma_7 &= \text{ASCII} \\
\Sigma_8 &= \text{ADN} = \{\langle \textsc{Adénine} \rangle, \langle \textsc{Cytosine} \rangle, \langle \textsc{Guanine} \rangle, \langle \textsc{Thymine} \rangle\}
\end{aligned}
$$

Définition 4.3

Un mot est une suite finie, possiblement vide, de symboles appartenant à un alphabet.

Exemple 4.4

abba *est un mot sur l'alphabet* $\{\mathtt{a}, \mathtt{b}\}$.

Définition 4.5

La lettre grecque epsilon minuscule, ε, dénote le mot vide, quel que soit l'alphabet en usage.

Définition 4.6

Σ^* est l'ensemble de tous les mots pouvant être formés à partir de l'alphabet Σ.

Remarque 4.7

On a toujours $\varepsilon \in \Sigma^*$, quel que soit Σ.

Exemple 4.8

$$
\begin{aligned}
\Sigma_2 &= \{\mathtt{a}, \mathtt{b}\} \\
\Sigma_2^* &= \{\varepsilon, \mathtt{a}, \mathtt{b}, \mathtt{aa}, \mathtt{ab}, \mathtt{ba}, \mathtt{bb}, \mathtt{aaa}, \mathtt{aab}, ...\}
\end{aligned}
$$

$$
\begin{aligned}
\Sigma_3 &= \{0, 1\} \\
\Sigma_3^* &= \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, ...\}
\end{aligned}
$$

$$
\begin{aligned}
\Sigma_6 &= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, (,)\} \\
\Sigma_6^* &\ni (25 + 50) * 56 \\
\Sigma_6^* &\ni (()25 + + + 6
\end{aligned}
$$

Définition 4.9

Soit $w = w_1 w_2 \ldots w_k \in \Sigma^*$, alors

$$|w| = k.$$

L'entier k est la longueur du mot w, c'est-à-dire le nombre de symboles dans w.

Exemple 4.10

$$|\texttt{allo}| = 4.$$

Définition 4.11

Soient $w = w_1 w_2 \dots w_k \in \Sigma^*$ et $\mathsf{a} \in \Sigma$, alors

$$|w|_{\mathsf{a}} = \#\{w_i \mid w_i = \mathsf{a}\},$$

c'est-à-dire le nombre d'occurrences du symbole a dans le mot w.

Exemple 4.12

$$|\mathtt{yabadabadoo}|_{\mathsf{a}} = 4 \qquad \textit{et} \qquad |\mathtt{yabadabadoo}|_{\mathsf{b}} = 2.$$

Définition 4.13

Soient

$$x = x_1 x_2 \dots x_k \in \Sigma^* \quad \text{et} \quad y = y_1 y_2 \dots y_l \in \Sigma^*,$$

alors

$$x \cdot y = x_1 x_2 \dots x_k y_1 y_2 \dots y_l$$

est la concaténation de x et y.

Remarque 4.14

$w \cdot \varepsilon = w$, quel que soit le mot w.

Exemple 4.15

$$\texttt{bon} \cdot \texttt{jour} = \texttt{bonjour}$$
$$\texttt{bon} \cdot \varepsilon = \texttt{bon}$$

Définition 4.16

Soient un mot $w \in \Sigma^*$ et un entier positif i, alors

$$w^i = \underbrace{w \cdot w \cdots w}_{i \text{ fois}},$$

et

$$w^0 = \varepsilon.$$

De plus, on omet souvent l'opérateur de concaténation :

$$xy = x \cdot y.$$

Définition 4.17

Soit un mot $w \in \Sigma^*$, alors w^{R} est le renversé de w, c'est-à-dire le mot formé avec les symboles de w pris de droite à gauche.

Définition 4.18

Soit $<_\Sigma$ une relation d'ordre sur l'alphabet Σ. On appelle ordre lexicographique la relation d'ordre suivante sur Σ^* :

$$|w| < |w'| \quad \Rightarrow \quad w <_{\Sigma^*} w'$$

$$\left.\begin{array}{l} |w| = |w'| \\ w = x \cdot \mathtt{a} \cdot y \\ w' = x \cdot \mathtt{b} \cdot z \\ \mathtt{a} <_\Sigma \mathtt{b} \end{array}\right\} \Rightarrow \quad w <_{\Sigma^*} w'$$

Exemple 4.19

$$\begin{aligned} \Sigma &= \{0, 1\} \\ \Sigma^* &= \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, ...\} \end{aligned}$$

Définition 4.20

Un langage sur l'alphabet Σ est un sous-ensemble de Σ^*.

Définition 4.21

On note $\{\}$ ou $\emptyset$ le langage vide, c'est-à-dire le langage qui ne contient aucun mot.

Remarque 4.22

$\varepsilon \notin \emptyset$.

Exemple 4.23

Divers langages sur l'alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ *:*

$$
\begin{aligned}
L_1 &= \{\mathtt{a}, \mathtt{b}, \mathtt{abba}, \mathtt{abbb}\} \\
\\
L_2 &= \{w \mid \ |w| = 2k, k \in \mathbb{N}\} \\
&= \{\varepsilon, \mathtt{aa}, \mathtt{ab}, \mathtt{ba}, \mathtt{bb}, \mathtt{aaaa}, \mathtt{aaab}, \mathtt{aaba}, \mathtt{aabb}, \ldots\} \\
&= \textit{les mots qui contiennent un nombre pair de symboles} \\
\\
L_3 &= \{w \mid w = aw',\ w' \in \Sigma^*\} \\
&= \{\mathtt{a}, \mathtt{aa}, \mathtt{ab}, \mathtt{aaa}, \mathtt{aab}, \mathtt{aba}, \mathtt{abb}, \mathtt{aaaa}, \ldots\} \\
&= \textit{les mots qui commencent par } \mathtt{a}
\end{aligned}
$$

$$
\begin{aligned}
L_4 &= \{w \mid \ |w|_{\mathtt{a}} = |w|_{\mathtt{b}}\} \\
&= \{\varepsilon, \mathtt{ab}, \mathtt{ba}, \mathtt{aabb}, \mathtt{abab}, \mathtt{abba}, \mathtt{baab}, \mathtt{baba}, \ldots\} \\
&= \text{les mots qui contiennent autant de a que de b}
\end{aligned}
$$

$$
\begin{aligned}
L_5 &= \{w \mid \ |w|_{\mathtt{a}} - |w|_{\mathtt{b}} \text{ est un nombre premier}\} \\
&= \{\mathtt{aaab}, \ldots, \mathtt{aabbabaaaaa}, \ldots\}
\end{aligned}
$$

Exemple 4.24

Divers langages sur l'alphabet $\Sigma = \text{ASCII}$ *:*

$$
\begin{aligned}
L_6 &= \{w \mid w = (x, y, z) \text{ avec } x, y, z \in \{0, 1, \ldots, 9\}^* \text{ et } xy = z\} \\
L_7 &= \{w \mid w \text{ est une fonction syntaxiquement correcte} \\
&\quad \text{dans le langage de programmation Java} \\
&\quad \text{et cette fonction retourne toujours } \langle \textsc{True} \rangle\}
\end{aligned}
$$

$$
L_8 = \{w \mid w \text{ compile sans erreur en C++}\}
$$

$$
L_9 = \{w \mid w \text{ est un programme C++ qui écrit “Hello world !”}\}
$$

Exemple 4.25

Divers langages sur l'alphabet $\Sigma =$ ADN *:*

$$L_9 = \{w \mid w \textit{ est la séquence d'un gène bactérien}\}$$

$$L_{10} = \{w \mid w \textit{ est le génome d'un individu}\}$$

Opérations sur les langages

- l'union
- l'intersection
- le complément
- le produit cartésien
- l'ensemble des parties

Définition 4.26 (Union de deux langages)

L'union de deux langages L_1 et L_2 :

$$L_1 \cup L_2 = \{w \mid w \in L_1 \text{ ou } w \in L_2\}.$$

Exemple 4.27

Soient

$$\Sigma = \{\mathtt{a}, \mathtt{b}\},\ L_1 = \{\varepsilon, \mathtt{a}, \mathtt{aa}\} \textit{ et } L_2 = \{\mathtt{a}, \mathtt{bb}\},$$

alors

$$L_1 \cup L_2 = \{\varepsilon, \mathtt{a}, \mathtt{aa}, \mathtt{bb}\}.$$

Définition 4.28 (Intersection de deux langages)

L'intersection de deux langages L_1 et L_2 :

$$L_1 \cap L_2 = \{w \mid w \in L_1 \text{ et } w \in L_2\}.$$

Exemple 4.29

Soient

$$\Sigma = \{\mathtt{a}, \mathtt{b}\} \textit{ et } L_1 = \{\varepsilon, \mathtt{a}, \mathtt{aa}\} \textit{ et } L_2 = \{\mathtt{a}, \mathtt{bb}\},$$

alors

$$\begin{aligned} L_1 \cap L_2 &= \{\mathtt{a}\}, \\ L_1 \cap \emptyset &= \emptyset. \end{aligned}$$

Définition 4.30 (Complément d'un langage)

Le complément d'un langage L sur l'alphabet Σ :

$$\overline{L} = \{w \in \Sigma^* \mid w \notin L\}.$$

Exemple 4.31

Soient

$$\Sigma = \{\mathtt{a}, \mathtt{b}\} \text{ et } L = \{\varepsilon, \mathtt{a}, \mathtt{aa}\},$$

alors

$$\overline{L} = \{\mathtt{b}, \mathtt{ab}, \mathtt{ba}, \mathtt{bb}, \mathtt{aaa}, \mathtt{aab}, \mathtt{aba}, \mathtt{baa}, \mathtt{bba}, \ldots\}.$$

Définition 4.32 (Produit cartésien de deux langages)

Le *produit cartésien* de deux ensembles $\mathcal{A}$ et $\mathcal{B}$:

$$\mathcal{A} \times \mathcal{B} = \{(x, y) : x \in \mathcal{A} \text{ et } y \in \mathcal{B}\}.$$

Remarque 4.33

On a que $\#(\mathcal{A} \times \mathcal{B}) = (\#\mathcal{A})(\#\mathcal{B})$ *et que* $\mathcal{A} \times \emptyset = \emptyset$.

Exemple 4.34

Soient

$$\Sigma = \{\mathtt{a}, \mathtt{b}\}, L_1 = \{\varepsilon, \mathtt{a}, \mathtt{aa}\}, \text{ et } L_2 = \{\mathtt{a}, \mathtt{bb}\},$$

alors

$$L_1 \times L_2 = \{(\varepsilon, \mathtt{a}), (\varepsilon, \mathtt{bb}), (\mathtt{a}, \mathtt{a}), (\mathtt{a}, \mathtt{bb}), (\mathtt{aa}, \mathtt{a}), (\mathtt{aa}, \mathtt{bb})\}.$$

Notation 4.35

Pour un ensemble $\mathcal{A}$ et un entier $k \geqslant 2$:

$$\mathcal{A}^k = \underbrace{\mathcal{A} \times \mathcal{A} \times \cdots \times \mathcal{A}}_{k \text{ fois}}$$

$$= \text{l'ensemble des } k\text{-tuplets d'éléments de } \mathcal{A}.$$

Remarque 4.36

$$\#(\mathcal{A}^k) = (\#\mathcal{A})^k.$$

Définition 4.37 (Ensemble des parties)

L'ensemble des parties d'un ensemble $\mathcal{A}$ est défini comme :

$$\mathcal{P}(\mathcal{A}) = \{\mathcal{B} : \mathcal{B} \subseteq \mathcal{A}\}.$$

Remarque 4.38

$$\#\mathcal{P}(\mathcal{A}) = 2^{\#\mathcal{A}}.$$

Exemple 4.39

Soient

$$\Sigma = \{\mathtt{a}, \mathtt{b}\} \text{ et } L = \{\varepsilon, \mathtt{a}, \mathtt{aa}\},$$

alors

$$\mathcal{P}(L) = \{\emptyset, \{\varepsilon\}, \{\mathtt{a}\}, \{\mathtt{aa}\}, \{\varepsilon, \mathtt{a}\}, \{\varepsilon, \mathtt{aa}\}, \{\mathtt{a}, \mathtt{aa}\}, \{\varepsilon, \mathtt{a}, \mathtt{aa}\}\}.$$

Classes de langages

Définition 4.40

Une classe de langages est un ensemble de langages.

La plupart du temps, on n'étudie que les classes associées à certains modèles calculatoires.

Par exemple, la classe des langages qui sont reconnus par un type de machine ayant telles ressources et telles limitations...

Le terme classe de complexité est souvent utilisé dans la littérature.

Définition 4.41

FINI désigne la classe de tous les langages finis, c'est-à-dire l'ensemble des langages qui contiennent un nombre fini de mots.

Exemple 4.42

Si

$$\begin{aligned} \Sigma &= \{\mathtt{a}, \mathtt{b}\} \\ L &= \{\mathtt{abba}, \mathtt{baba}, \varepsilon\}, \end{aligned}$$

alors

$$\begin{aligned} L &\in \mathsf{FINI} \\ \emptyset &\in \mathsf{FINI} \\ \Sigma^* &\notin \mathsf{FINI} \end{aligned}$$

Définition 4.43

On désigne la classe de tous les langages par UNIVERS.

Fermeture d'une classe de langages par rapport à une opération

Définition 4.44

On dit qu'une classe est fermée par rapport à une opération, ou fermée pour une opération, si le résultat de l'opération est toujours un élément de la classe lorsque les opérandes sont pris dans la classe.

Théorème 4.45

La classe FINI *est fermée pour l'union.*

Démonstration :

Il suffit de voir que l'union de deux langages finis est un langage fini.

Théorème 4.46

La classe FINI *est fermée pour l'intersection.*

Démonstration :

L'intersection de deux langages finis donne toujours un langage fini.

Théorème 4.47

La classe FINI *n'est pas fermée pour l'opération complément.*

Démonstration :

On a : $\emptyset \in$ FINI, *mais* $\overline{\emptyset} = \Sigma^* \notin$ FINI.

Remarque 4.48

Le complément d'un langage fini, quel qu'il soit, ne donne jamais un langage fini.

Résumé

Concept	**Définition**	**Peut-être vide ?**	**Peut-être infini ?**	**Exemples**
symbole	—	—	—	a
alphabet	ensemble de symboles	non	non	{a, b, c}
mot	suite de symboles	oui (ε)	non	abaac
langage	ensemble de mots	oui ($\emptyset$)	oui	{ε, ba, bba, bbba, ...}
classe	ensemble de langages	oui (inintéressant)	oui (toutes les classes intéressantes sont infinies)	FINI REG HC P

Définition 4.49 (Automate fini déterministe)

Un automate fini est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où :

- Q est un ensemble fini d'états ;
- Σ est un alphabet ;
- $\delta : Q \times \Sigma \to Q$ est la fonction de transition ;
- $q_0 \in Q$ est l'état initial ;
- $F \subseteq Q$ est l'ensemble états acceptants.

Exemple 4.50

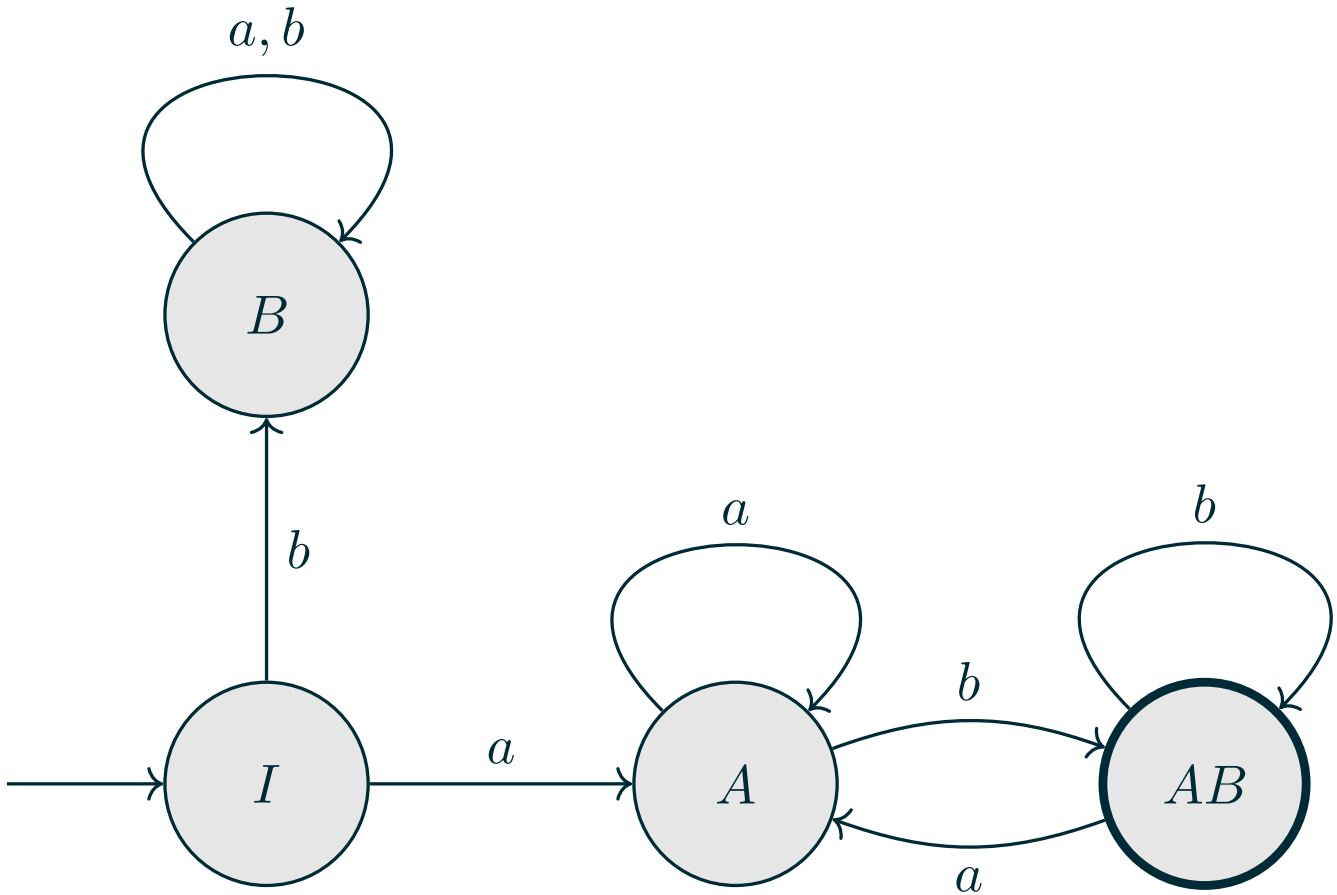
Soit

$$M = (Q, \Sigma, \delta, q_0, F),$$

où

$$\begin{aligned} Q &= \{I, A, B, AB\}, \\ \Sigma &= \{a, b\}, \\ q_0 &= I, \\ F &= \{AB\}, \end{aligned}$$

et où la fonction δ de transition est donnée par le diagramme de la Figure **??**.


a, b
B
b
a
b
b
I
a
A
AB
a

Le langage

$$\{w \mid w = \mathtt{a}x\mathtt{b} \text{ avec } x \in \{\mathtt{a}, \mathtt{b}\}^*\}.$$

est l'ensemble des mots qui amènent l'automate M de son état initial à l'état AB.

Définition 4.51

Soit $M = (Q, \Sigma, \delta, q_0, F)$ un automate.

Une configuration de l'automate M est une paire $[w, q]$ où $w \in \Sigma^*$ et $q \in Q$.

Définition 4.52

Soit $M = (Q, \Sigma, \delta, q_0, F)$ un automate.

Si $\mathsf{a} \in \Sigma$ et $\delta(q, \mathsf{a}) = q'$, alors

$$[\mathsf{a} \cdot w, q] \vdash [w, q']$$

dénote une transition de l'automate M de la configuration $[\mathsf{a} \cdot w, q]$ à la configuration $[w, q']$.

Définition 4.53

Soit $M = (Q, \Sigma, \delta, q_0, F)$ un automate.

Pour $x \in \Sigma^*$ et $w \in \Sigma^*$,

$$[x \cdot w, q] \overset{n}{\vdash} [w, q']$$

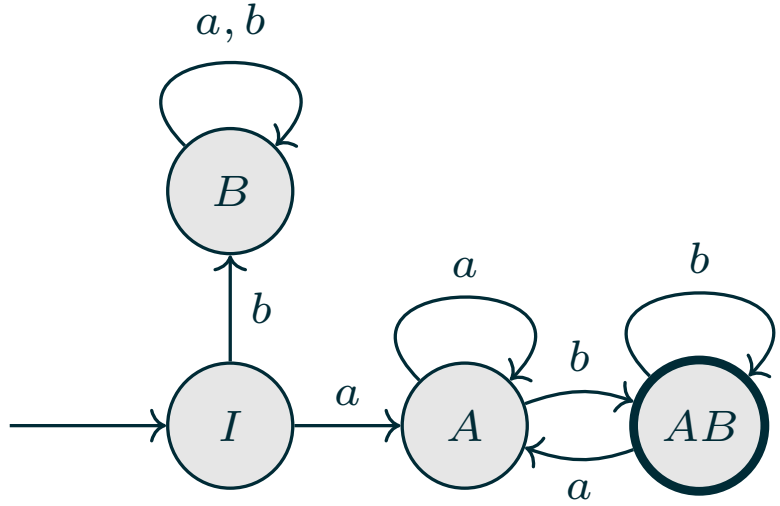
dénote n transitions successives amenant l'automate M de la configuration $[x \cdot w, q]$ à la configuration $[w, q']$, et

$$[x \cdot w, q] \overset{*}{\vdash} [w, q']$$

indique que

$$[x \cdot w, q] \overset{n}{\vdash} [w, q']$$

pour $n = |x|$.



$$
\begin{aligned}
[abab, I] &\vdash [bab, A] \\
&\vdash [ab, AB] \\
&\vdash [b, A] \\
&\vdash [\varepsilon, AB]
\end{aligned}
\qquad
\begin{aligned}
[aab, I] &\vdash [ab, A] \\
&\vdash [b, A] \\
&\vdash [\varepsilon, AB]
\end{aligned}
$$

$$
\begin{aligned}
[ba, I] &\vdash [a, B] \\
&\vdash [\varepsilon, B]
\end{aligned}
$$

Définition 4.54

Soit $M = (Q, \Sigma, \delta, q_0, F)$ un automate.

On dira que l'automate M accepte le mot $w \in \Sigma^*$ s'il $\exists q \in F$ tel que

$$[w, q_0] \overset{*}{\vdash} [\varepsilon, q].$$

Définition 4.55

Soit $M = (Q, \Sigma, \delta, q_0, F)$ un automate.

L'ensemble

$$L(M) = \{w \in \Sigma^* \mid M \text{ accepte } w\}$$

est le langage reconnu par l'automate M.

Définition 4.56

Un langage K est régulier s'il existe un automate M tel que $K = L(M)$.

Définition 4.57

REG désigne la classe des langages réguliers

Exemple de langage régulier

Soit

$$M = (Q, \Sigma, \delta, q_0, F)$$

où

$$\begin{aligned} Q &= \{\langle 0\rangle, \langle 1\rangle, \langle 2\rangle\} \\ \Sigma &= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\ q_0 &= \langle 0\rangle \\ F &= \{\langle 0\rangle\}, \end{aligned}$$

et où la fonction de transition δ est donnée par le diagramme de la Figure 4.1

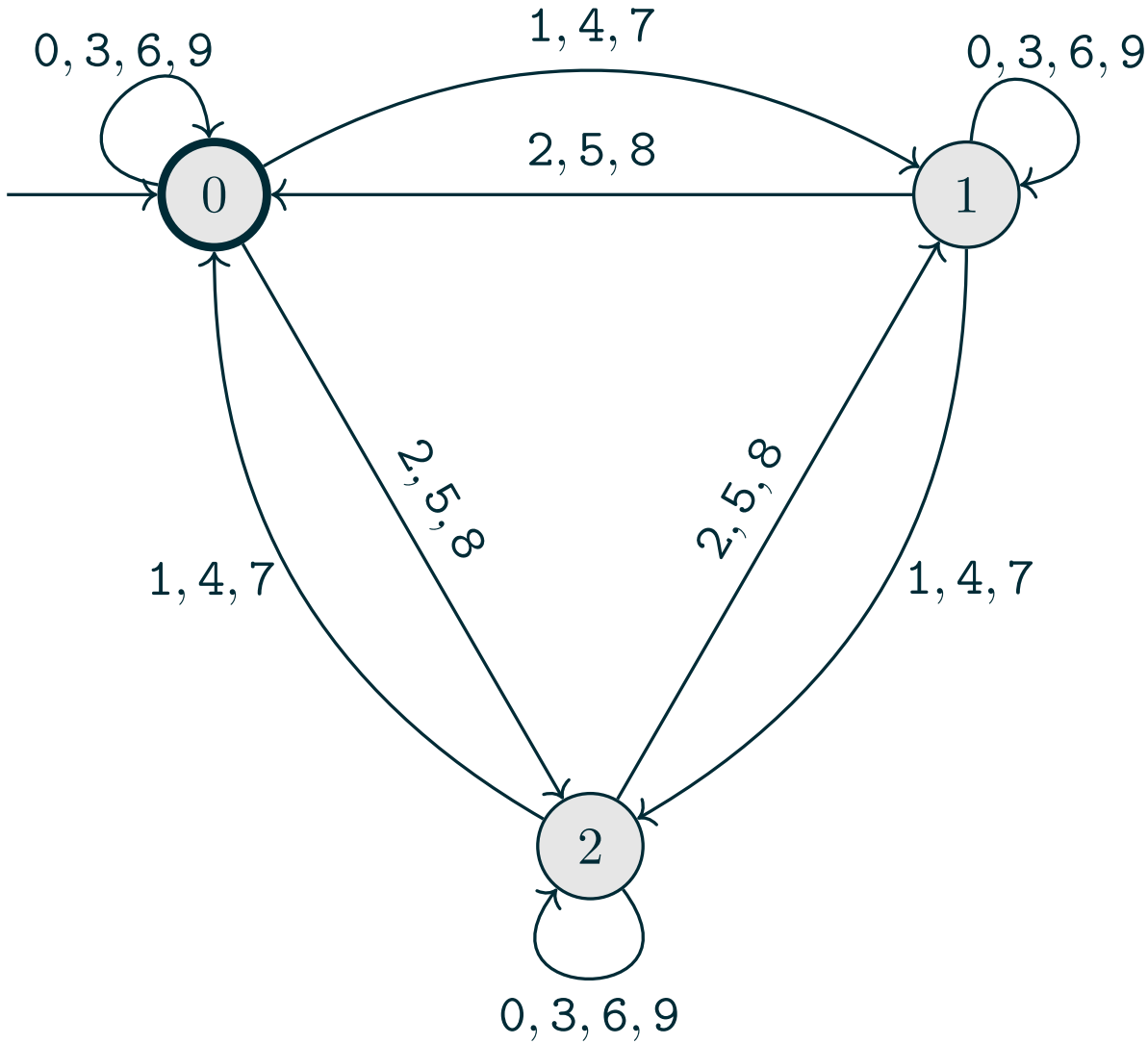
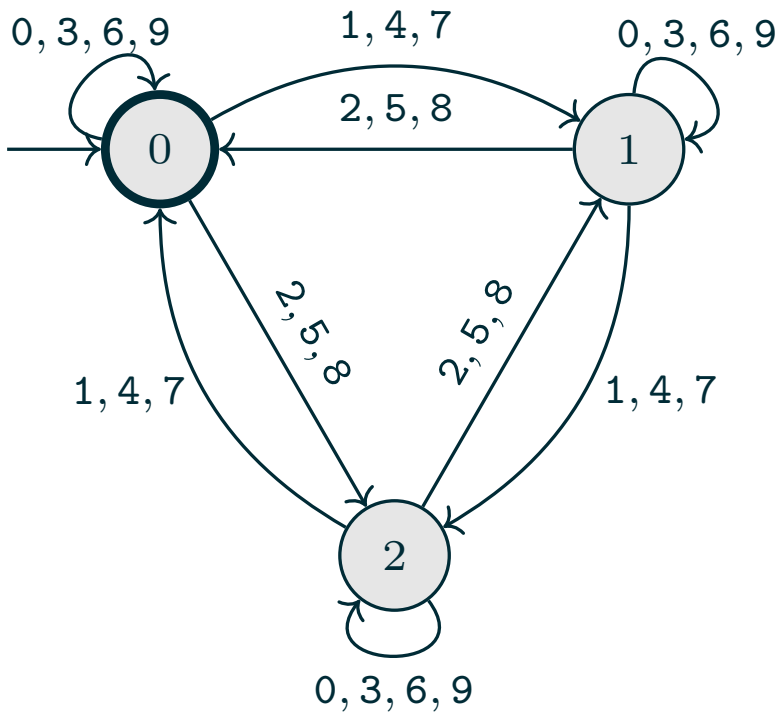


Fig. 4.1 : Un exemple d'automate fini.



$$
\begin{aligned}
[68725, \langle 0 \rangle] &\vdash [8725, \langle 0 \rangle] \\
&\vdash [725, \langle 2 \rangle] \\
&\vdash [25, \langle 0 \rangle] \\
&\vdash [5, \langle 2 \rangle] \\
&\vdash [\varepsilon, \langle 1 \rangle].
\end{aligned}
$$

Donc 68725 $\notin L(M)$.

Proposition 4.58

L'automate de la Figure 4.1 reconnaît le langage contenant le mot vide et les chaînes de chiffres décimaux qui forment un nombre divisible par 3.

Démonstration :

L'inspection du diagramme de la fonction de transition δ révèle que M accepte ε de même que les nombres dont la somme des chiffres est congrue à 0 modulo 3.

Or, si le développement décimal de l'entier n est $d_i d_{i-1} \cdots d_0$, on a :

$$
\begin{aligned}
& n \equiv 0 \mod 3 \\
\Leftrightarrow \quad & \sum_{j=0}^{i} d_j 10^j \equiv 0 \mod 3 \\
\Leftrightarrow \quad & \sum_{j=0}^{i} d_j \equiv 0 \mod 3 \qquad (\text{car } 10^j \cong 1)
\end{aligned}
$$

□

Théorème 4.59

La classe REG *est fermée pour l'opération complément.*

Démonstration :

Si $K \in$ REG, alors il est reconnu par un automate M.

Soit M' l'automate construit comme M, mais où les états acceptants et non-acceptants ont été interchangés.

Alors M' reconnaît $\overline{K}$. Donc $\overline{K} \in$ REG et la classe est fermée sur cette opération. □

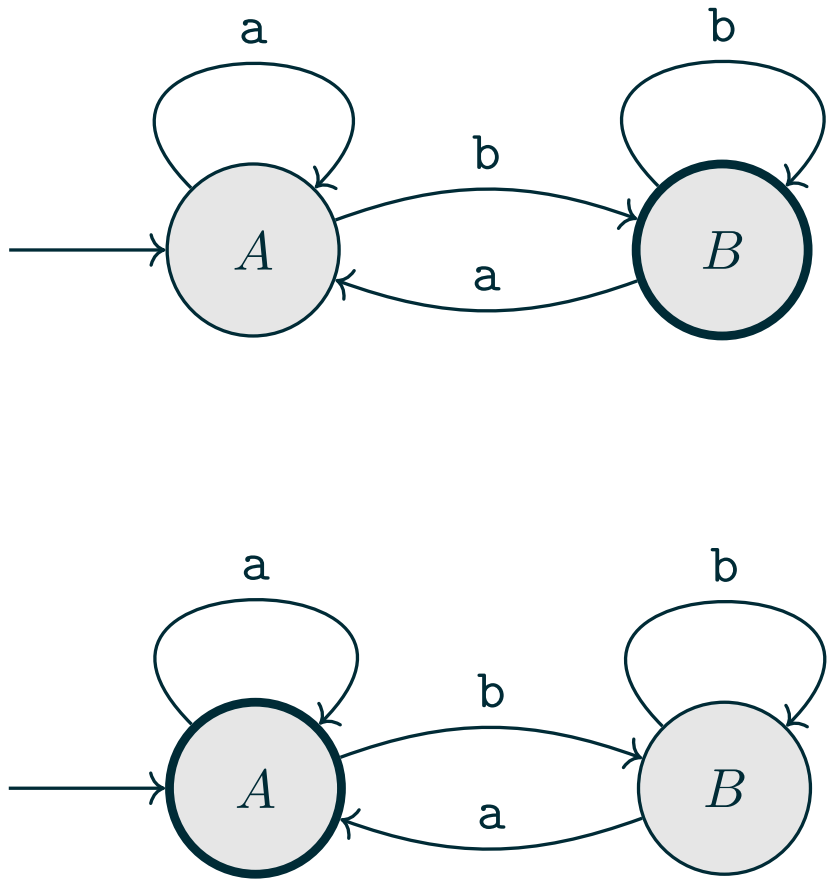


Fig. 4.2 : Exemple d'automate pour l'opération complément.

Théorème 4.60

La classe REG *est fermée pour l'intersection.*

Démonstration :

Soient $K_1 \in$ REG et $K_2 \in$ REG deux langages réguliers reconnus respectivement par les automates M_1 et M_2 :

$$\begin{aligned} M_1 &= (Q_1,\ \Sigma,\ \delta_1,\ q_{0,1},\ F_1) \\ M_2 &= (Q_2,\ \Sigma,\ \delta_2,\ q_{0,2},\ F_2). \end{aligned}$$

Considérons l'automate M suivant :

$$M = (Q_1 \times Q_2,\ \Sigma,\ \delta,\ (q_{0,1}, q_{0,2}),\ F_1 \times F_2),$$

où

$$\delta((q_1, q_2), x) = (q_1', q_2')$$

si et seulement si

$$\delta_1(q_1, x) = q_1' \qquad \text{et} \qquad \delta_2(q_2, x) = q_2',$$

comme dans l'exemple de la Figure 4.3.

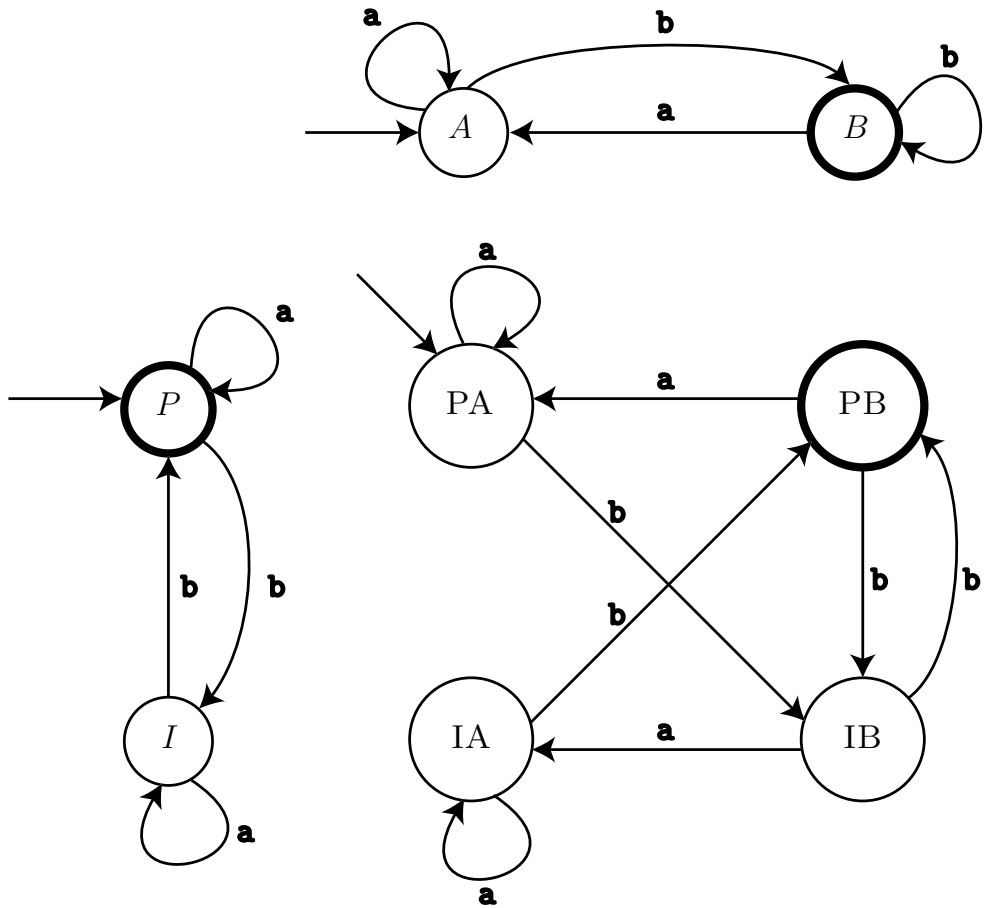


Fig. 4.3 : Exemple d'automate pour l'intersection.

Soit $w \in K_1 \cap K_2$.

Alors w amène M_1 de $q_{0,1}$ à un état final $f_1 \in F_1$ et amène également M_2 de $q_{0,2}$ à $f_2 \in F_2$.

Par construction, le mot w amène l'automate M de $(q_{0,1}, q_{0,2})$ à $(f_1, f_2) \in F_1 \times F_2$.

Par conséquent, M accepte w et $w \in L(M)$.

Inversement, si $f_1 \notin F_1$ ou $f_2 \notin F_2$, alors $(f_1, f_2) \notin F_1 \times F_2$ et M n'accepte pas w.

On a donc $L(M) = K_1 \cap K_2$ et $K_1 \cap K_2 \in$ REG. $\square$

Théorème 4.61

La classe REG *est fermée sous l'union.*

Démonstration :

Soient $K_1 \in$ REG et $K_2 \in$ REG, deux langages réguliers.

On a :

$$K_1 \cup K_2 = \overline{\overline{K_1} \cap \overline{K_2}}.$$

Comme la classe REG est fermée pour le complément et l'intersection, (théorèmes 4.59 et 4.60) alors les langages $\overline{K_1}$, $\overline{K_2}$, $\overline{K_1} \cap \overline{K_2}$, et enfin $\overline{\overline{K_1} \cap \overline{K_2}}$ sont réguliers.

On a donc $K_1 \cup K_2 \in$ REG. □

Théorème 4.62

$$\text{FINI} \subset \text{REG}$$

Démonstration :
Tout d'abord, on remarque que l'inclusion doit être stricte, étant donné que le langage $L = \Sigma^*$, qui contient une infinité de mots, est régulier.

Ensuite, il est facile de montrer que le langage vide est régulier.

Notons en plus qu'un automate accepte le mot vide si et seulement si son état initial est acceptant.

Nous allons maintenant considérer les langages finis, non vides, qui ne contiennent pas le mot vide.

Soient $L = \{w_1, \ldots, w_k\}$ un langage fini de k mots, $k \geq 1$ et $\varepsilon \notin L$, avec

$$\begin{aligned} w_1 &= x_{1,1}\, x_{1,2}\, \ldots\, x_{1,l_1} \\ w_2 &= x_{2,1}\, x_{2,2}\, \ldots\, x_{2,l_2} \\ &\vdots \\ w_k &= x_{k,1}\, x_{k,2}\, \ldots\, x_{k,l_k} \end{aligned}$$

où $l_i = |w_i|$ et $x_{i,j} \in \Sigma$.

Soit

$$l = \max_{1 \leq i \leq k} l_i$$

la longueur maximale des mots de L.

Considérons l'automate suivant :

$$M = (Q, \Sigma, \delta, q_0, F)$$

où

$$\begin{aligned} Q &= \{\langle \textsc{Init} \rangle\} \cup \Sigma \cup (\Sigma \times \Sigma) \cup \cdots \cup \Sigma^l \cup \{\langle \textsc{Puits} \rangle\} \\ q_0 &= \langle \textsc{Init} \rangle \\ F &= \{\langle x_{1,1}\, x_{1,2}\, \ldots\, x_{1,l_1} \rangle,\ \langle x_{2,1}\, x_{2,2}\, \ldots\, x_{2,l_2} \rangle,\ \ldots,\ \langle x_{k,1}\, x_{k,2}\, \ldots\, x_{k,l_k} \rangle\} \end{aligned}$$

et où la fonction δ de transition est donnée par :

$\forall z,\ z_1,\ z_2,\ \ldots,\ z_l \in \Sigma\ :$

$$
\begin{aligned}
\delta(\langle \textsc{Init} \rangle,\ z) &= \langle z \rangle \\
\delta(\langle z_1\, z_2\, \ldots\, z_h \rangle,\ z) &= \langle z_1\, z_2\, \ldots\, z_h\, z \rangle, \quad \text{si } h < l \\
\delta(\langle z_1\, z_2\, \ldots\, z_l \rangle,\ z) &= \langle \textsc{Puits} \rangle \\
\delta(\langle \textsc{Puits} \rangle,\ z) &= \langle \textsc{Puits} \rangle
\end{aligned}
$$

comme dans l'exemple de la Figure 4.4.

$$\Sigma = \{\mathtt{a},\ \mathtt{b}\} \quad L = \{\mathtt{b},\ \mathtt{ab},\ \mathtt{aaa},\ \mathtt{bba}\} \quad k = 4 \quad l = 3$$

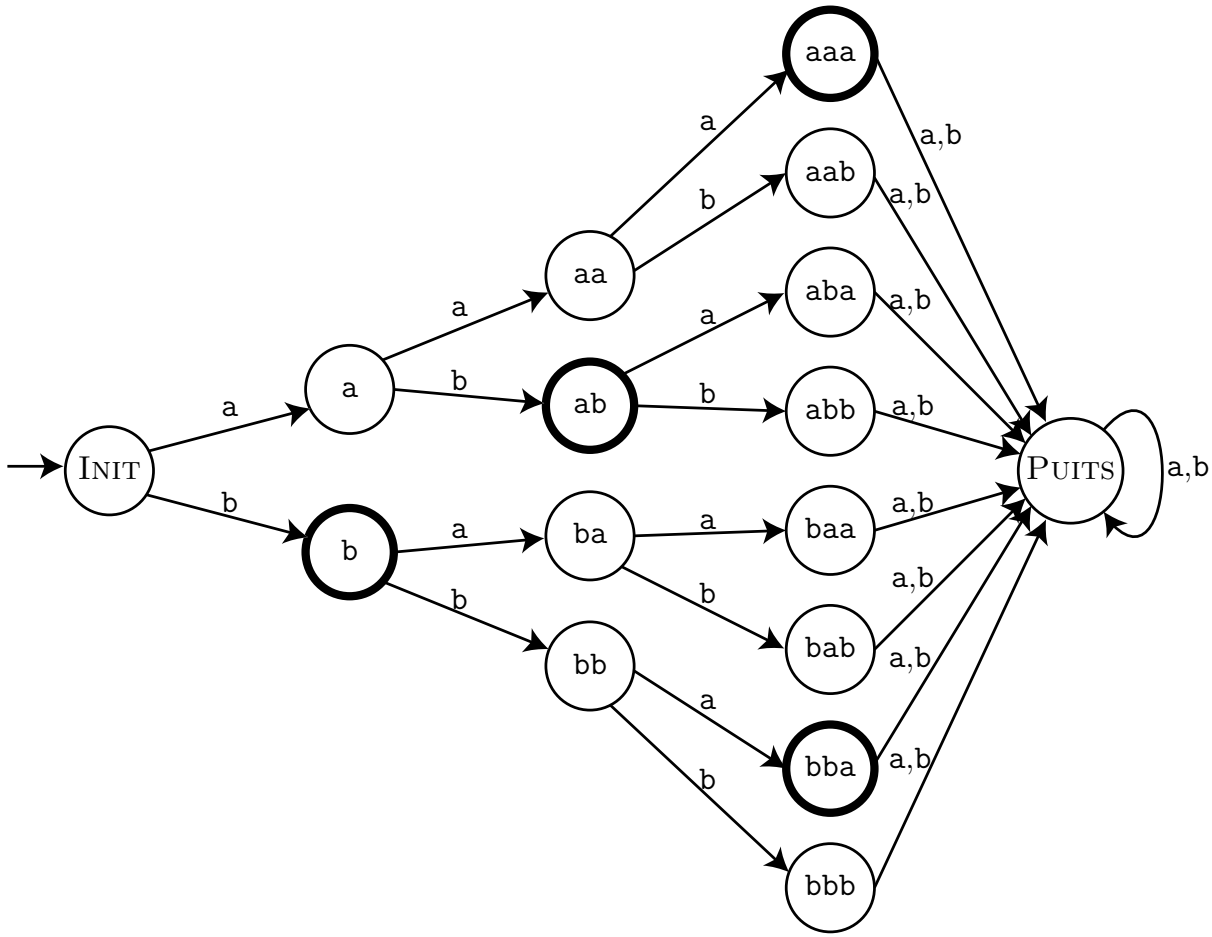


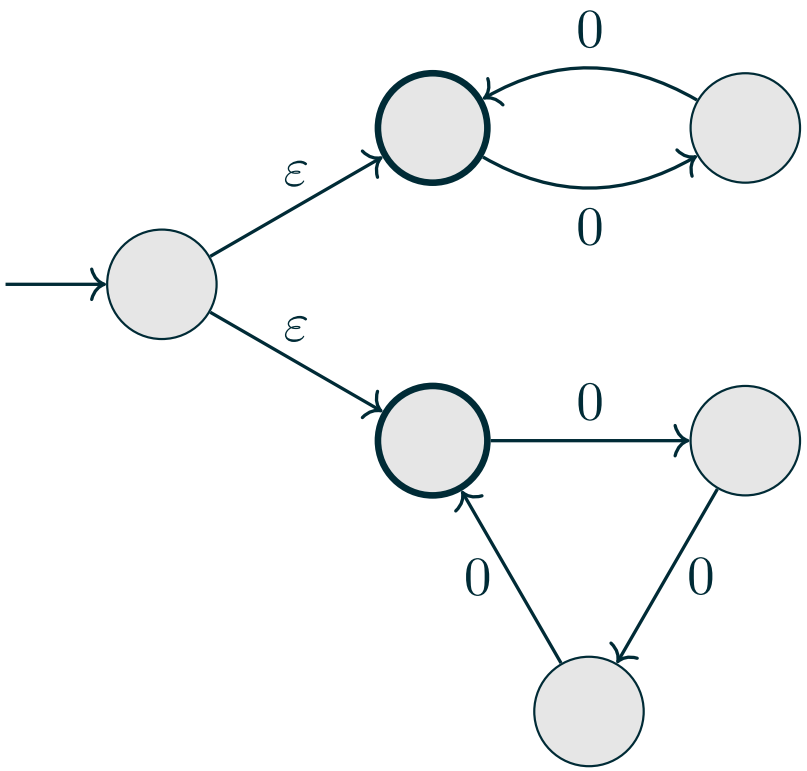
Fig. 4.4 : Exemple d'automate acceptant un langage fini.

L'automate ainsi construit acceptera les mots du langage L et uniquement ces mots-là.

Donc $L \in$ REG et FINI $\subseteq$ REG. $\square$

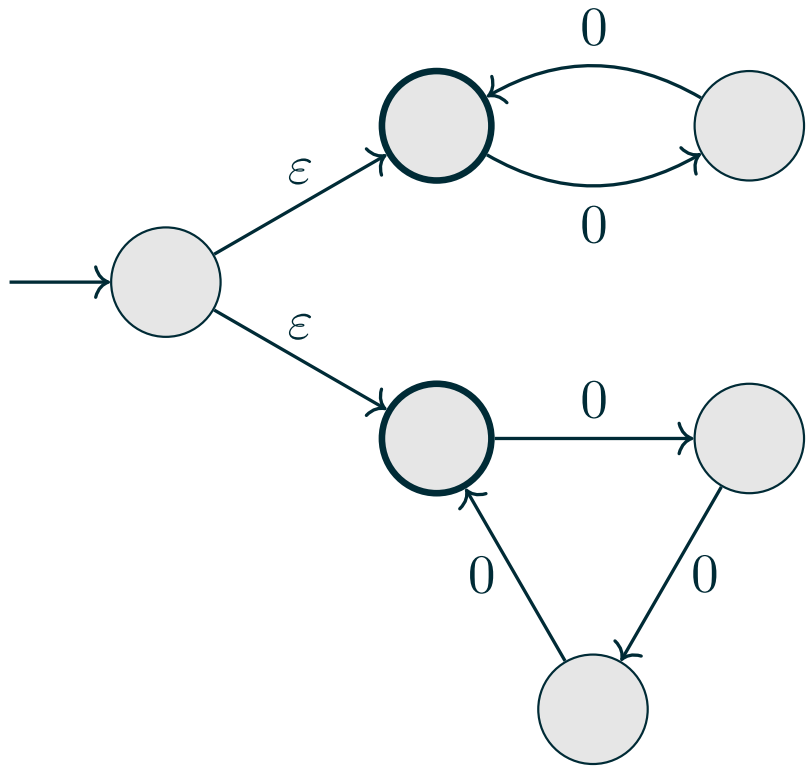
Automates finis non-déterministes

Un automate fini *non-déterministe* (AFN) est un automate fini où, au lieu de spécifier la transition qui se produit à chaque étape, on spécifie les transitions *possibles*.



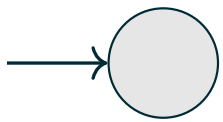
Ici, $\Sigma = \{0\}$, ε signifie une transition « gratuite » (qui peut s'effectuer sans lire de caractère).

Lors de l'exécution, l'automate « essaye » tous les chemins possibles, et si au moins *un* chemin possible mène à l'acceptation du mot, l'automate accepte.

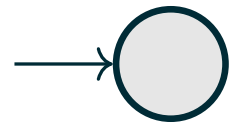


Cet automate accepte les chaînes de zéros de longueur divisible par 2 ou par 3.

Autres exemples :



$L(M) = \emptyset$



$L(M) = \{\varepsilon\}$

$L(M) = \{a\}$

Définition 4.63 (Automate fini non-déterministe)

Un automate fini non-déterministe (AFN) est un 5-tuple $(Q, \Sigma, \delta, q_0, F)$, où

- Q est un ensemble d'états
- Σ est un alphabet
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ est la fonction de transition
- $q_0 \in Q$ est l'état de départ
- $F \subseteq Q$ est l'ensemble d'états acceptants.

$\Rightarrow$ même définition que les automates déterministes, mais δ prend en entrée l'état et le prochain symbole **ou la chaîne vide** et produit **l'ensemble des états possibles** à la prochaine étape.

Équivalence entre AFN et AFD

Théorème 4.64

La classe des langages reconnus par un AFN est REG

Les AFN ne sont donc pas plus puissants que les AFD.

Démonstration :

Clairement, les AFN reconnaissent tous les langages dans REG, car un AFD est aussi un AFN. On doit donc prouver que pour tout AFN, il existe un AFD qui accepte le même langage.

Soit $M = (Q, \Sigma, \delta, q_0, F)$ un AFN quelconque. On construit un AFD $M' = (Q', \Sigma, \delta', q_0', F')$ avec $Q' = \mathcal{P}(Q)$ avec la propriété qu'à chaque étape du calcul, l'état $q \in \mathcal{P}(Q)$ de l'AFD contient l'ensemble des états dans lequel l'AFN pourrait se trouver à cette étape.

Comment construire l'automate formellement ?

Supposons d'abord qu'il n'y a pas de transition gratuites (donc pas de flèches ε). Alors, pour tout état de l'AFD $S \subseteq Q$ et tout symbole $a \in \Sigma$,

$$\begin{aligned}\delta'(S, a) &= \{q \mid q \text{ est atteignable en une étape depuis } S\} \\ &= \bigcup_{q \in S} \delta(q, a).\end{aligned}$$

L'état de départ est alors :

$$q'_0 = \{q_0\},$$

et les états acceptants sont tous les états contenant un état acceptant :

$$F' = \{S \subseteq Q \mid S \text{ contient un état } q \in F\}.$$

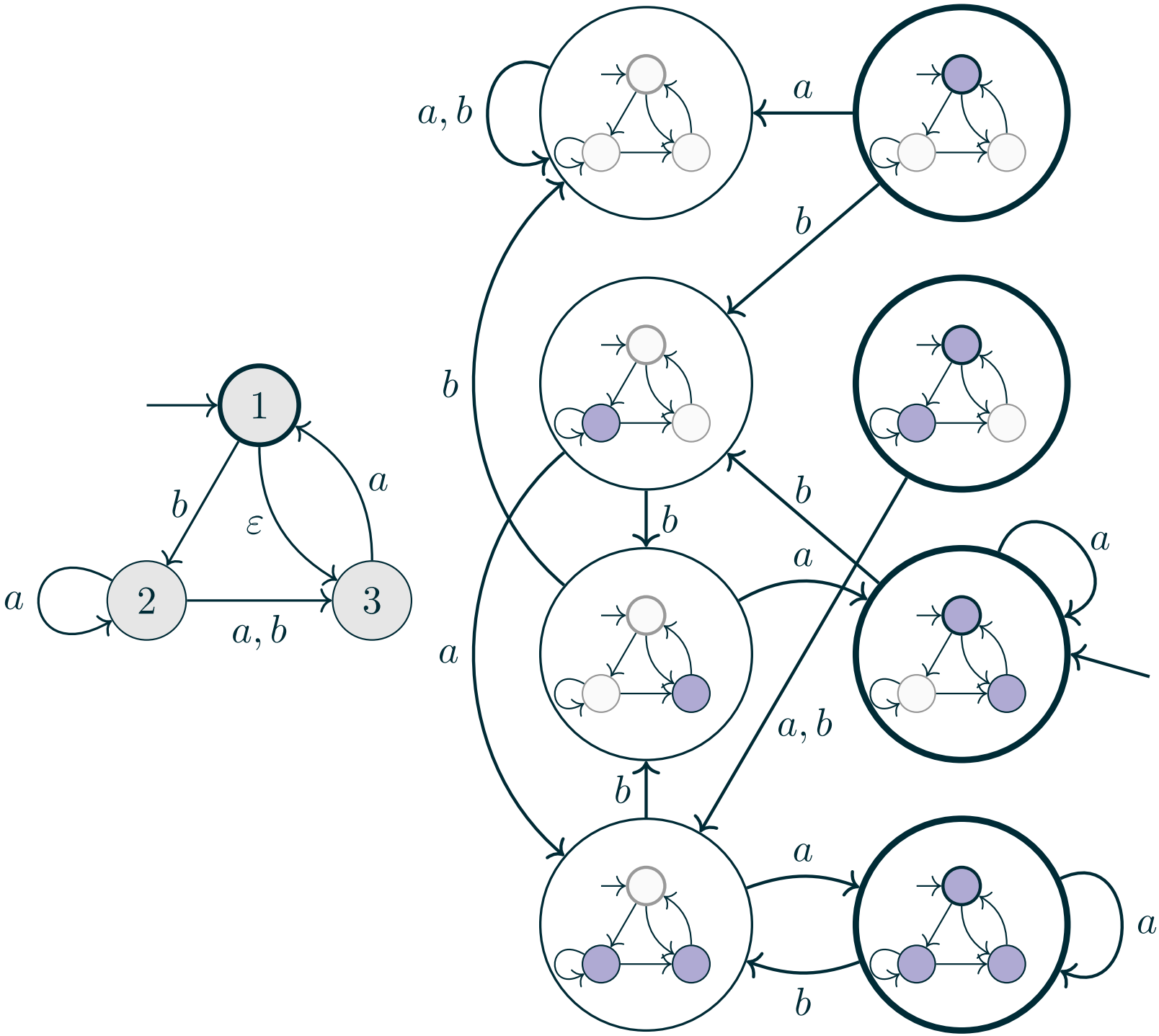
Si on a des flèches ε, alors définissons pour tout ensemble d'états $S \subseteq Q$, l'ensemble des états atteignables par des flèches ε :

$$E(S) = \{q \mid q \text{ peut être atteint avec 0 ou plus flèches } \varepsilon \text{ depuis } S\}.$$

On peut donc ajuster la définition précédente :

$$\begin{aligned} \delta'(S, a) &= \{q \mid q \text{ est atteignable en une étape depuis } S\} \\ &= \bigcup_{q \in S} E(\delta(q, a)) \\ q'_0 &= E\left(\{q_0\}\right). \end{aligned}$$

□


1
2
3
a
b
ε
a
a
a, b
a, b
a
b
b
b
b
a
a
a, b
b
a
a
b

Théorème 4.65

La classe REG *est fermée sous la concaténation*

Démonstration :

Soient L_1 et L_2 deux langages réguliers sur le même alphabet. On doit démontrer que $L_1 \circ L_2$ est également régulier.

Soient $M_1 = (Q_1, \Sigma, \delta_1, q_{0,1}, F_1)$ et $M_2 = (Q_2, \Sigma, \delta_2, q_{0,2}, F_1)$ deux AFN acceptant L_1 et L_2 respectivement. Nous allons construire un AFN $M = (Q, \Sigma, \delta, q_0, F)$ qui accepte $L_1 \circ L_2$ en « concaténant » M_1 et M_2 :

$$Q = Q_1 \cup Q_2$$

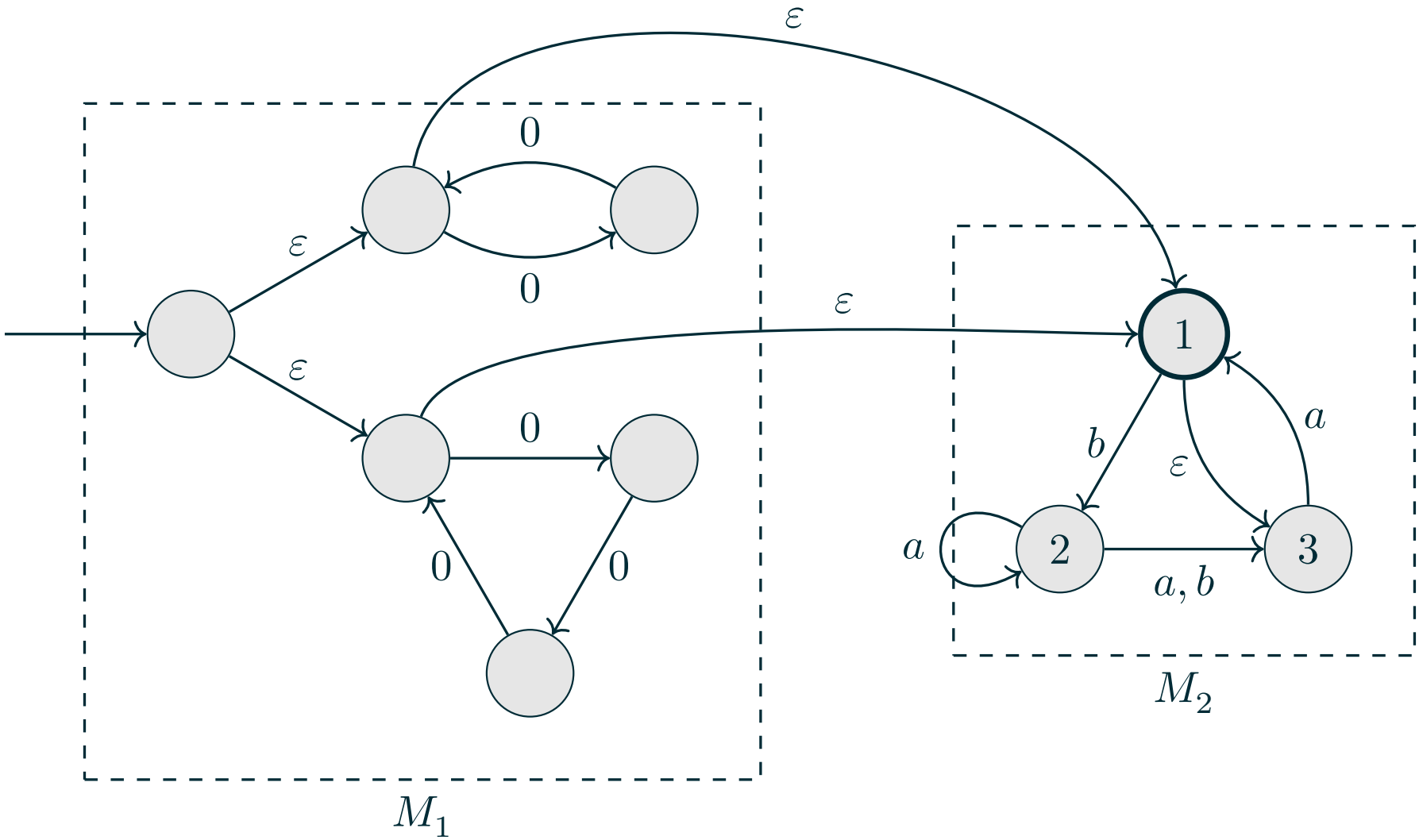
$$q_0 = q_{0,1}$$

$$F = F_2$$

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{si } q \in Q_1 \text{ et } q \notin F_1 \\ \delta_1(q,a) & \text{si } q \in F_1 \text{ et } a \neq \varepsilon \\ \delta_2(q,a) & \text{si } q \in Q_2 \\ \delta_1(q,a) \cup \{q_{0,2}\} & \text{si } q \in F_1 \text{ et } a = \varepsilon. \end{cases}$$

Pour passer de M_1 à M_2, il faut avoir lu un mot dans L_1, et ensuite pour accepter, il faut avoir lu un mot dans L_2. □

Exemple :



Théorème 4.66

La classe REG *est fermée sous l'étoile*

Ici, pour tout langage L,

$$L^i = \{w_1 w_2 \dots w_i \mid w_j \in L \ \ \forall i \geqslant 0\}$$

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

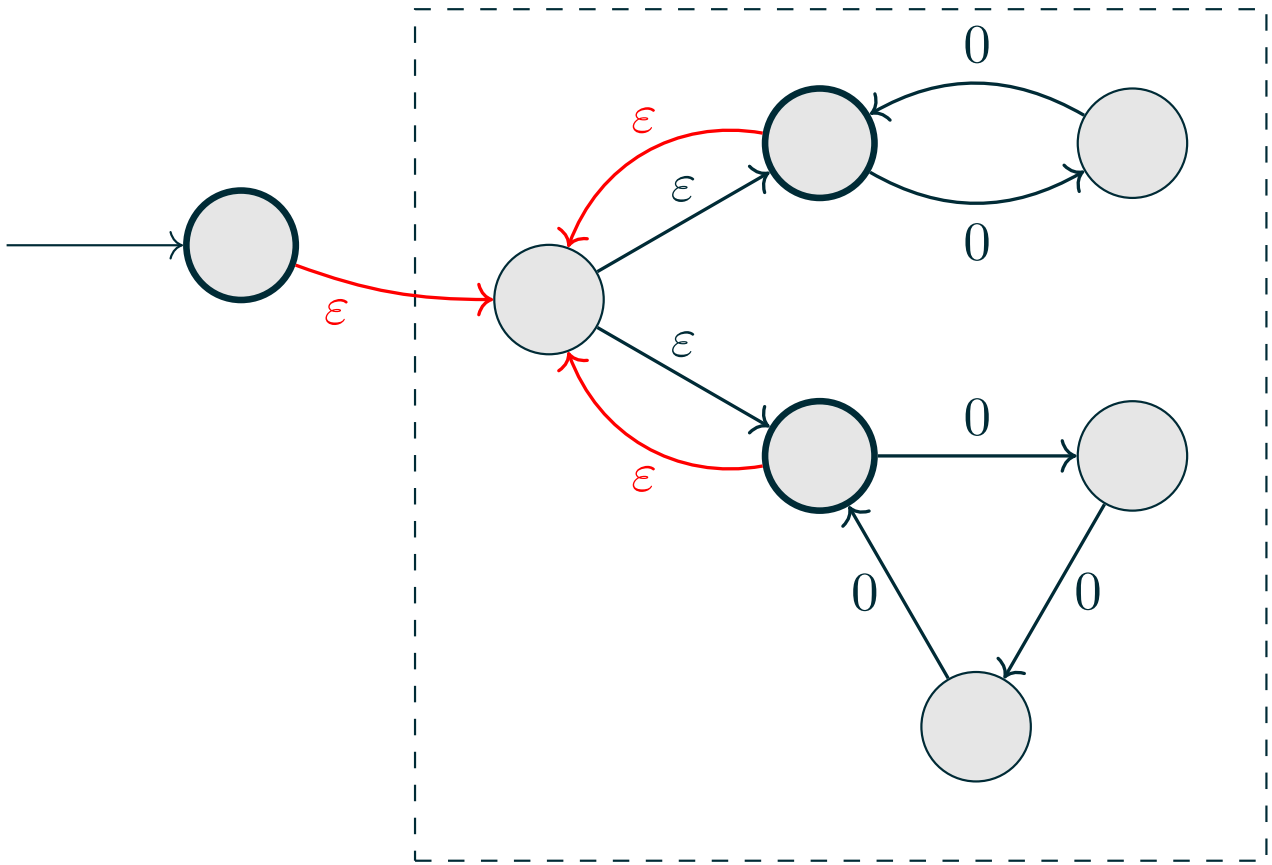
Démonstration :

Soit $M = (Q, \Sigma, \delta, q_0, F)$ un AFN reconnaissant L. Alors, on construit un AFN $M' = (Q', \Sigma, \delta', q_0', F')$ reconnaissant L^* :

$$Q' = Q \cup \{\langle init \rangle\}$$
$$q_0' = \langle init \rangle$$
$$F' = F \cup \{\langle init \rangle\}$$
$$\delta'(q,a) = \begin{cases} q_0 & \text{si } q = \langle init \rangle \text{ et } a = \varepsilon \\ \delta(q,a) \cup \{q_0\} & \text{si } q \in F \text{ et } a = \varepsilon \\ \delta(q,a) & \text{sinon.} \end{cases}$$

Donc, on a le même automate, mais on ajoute des flèches ε de tous les états acceptants à l'état de départ, et on ajoute un nouvel état de départ acceptant.

Exemple :



Expressions régulières

Définition 4.67 (Expression régulière)

$R \subseteq (\Sigma \cup \{\varepsilon, \emptyset, \cup, \circ, *\})^*$ est une expression régulière si R est

1. a où $a \in \Sigma$,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, où R_1 et R_1 sont des expressions régulières,
5. $(R_1 \circ R_2)$, où R_1 et R_2 sont des expressions régulières,
6. $(R_1)^*$, où R_1 est une expression régulière.

On écrira également $R_1 \circ R_2$ comme $R_1 R_2$.

Ces expressions représentent :

1. $R = a$: le langage $\{a\}$
2. $R = \varepsilon$: le langage $\{\varepsilon\}$
3. $R = \emptyset$: le langage vide
4. $(R_1 \cup R_2)$: l'union des langages représentés par R_1 et R_2
5. $(R_1 \circ R_2)$: la concaténation des langages représentés par R_1 et R_2
6. $(R_1)^*$: l'étoile du langage représenté par R_1.

Ici l'étoile est défini comme :

$$R^* := \{w_1 w_2 w_3 \cdots w_n \mid n \geqslant 0, \forall i, w_i \in R\}.$$

Voici quelques exemples d'expressions régulières :

1. $\mathtt{a} \cup \mathtt{b}$: le language fini $\{a, b\}$
2. 0^*10^* : le langage qui ne contient qu'un seul 1
3. $(00)^* \cup (000)^*$: le langage qui contient un nombre de zéro divisible par 2 ou par 3
4. $((0 \cup 1)(0 \cup 1))^*$: le langage qui contient un nombre pair de bits
5. $0^*\emptyset$: le langage vide

Équivalence avec les automates finis

Quelle est la classe des langages reconnus pouvant être exprimés par une expression régulière ?

Théorème 4.68

$L \in$ REG si et seulement si L peut être exprimé par une expression régulière.

On prouvera les deux directions de ce théorème via deux lemmes séparés.

Lemme 4.69

Si L peut être exprimé par une expression régulière, alors $L \in$ REG.

Démonstration :
Par induction structurelle sur l'expression régulière. On peut voir l'expression régulière comme une structure définie récursivement avec

$$\mathcal{S}_0 = \{\emptyset, \varepsilon\} \cup \Sigma,$$

et avec trois règles de construction pour l'union, la concaténation, et l'étoile.

Base de l'induction :

- $R = \emptyset$: le langage vide est clairement régulier.
- $R = \varepsilon$: le mot vide est clairement régulier.
- $R = \{a\}$ où $a \in \Sigma$: aussi clairement régulier.

Pas d'induction : Supposons que R_1 et R_2 sont réguliers. Alors :

- $R_1 \cup R_2$ est régulier, par la fermeture sous l'union.
- $R_1 \circ R_2$ est régulier, par la fermeture sous la concaténation.
- R_1^* est régulier, par la fermeture sous l'étoile.

□

Lemme 4.70

Si $L \in$ REG, alors il existe une expression régulière R qui exprime L.

Démonstration :

L'idée consiste à prendre un AFN reconnaissant L, et de le transformer étape par étape en une expression régulière. On va donc généraliser les AFN pour permettre les expressions régulières sur les flèches. Ensuite, on enlèvera les états un par un dans le but d'arriver à un automate ayant seulement deux états et l'expression régulière qu'on veut sur la seule flèche restante.

Définition 4.71 (AFNER)

Un automate fini non-déterministe à expression régulière (AFNER) est un tuple $(Q, \Sigma, \delta, q_0, q_F)$ où

- Q est l'ensemble des états
- Σ est l'alphabet
- $\delta : Q \times Q \to \mathcal{R}(\Sigma)$ est la fonction de transition, où $\mathcal{R}(\Sigma)$ représente l'ensemble des expressions régulières sur Σ.
- q_0 est l'état de départ
- q_F est l'état final ($q_0 \neq q_F$)

De plus nous requerrons que q_0 n'ait aucune flèche entrante ($\delta(q, q_0) = \emptyset$ pour tout q), et que q_F n'ait aucune flèche sortante ($\delta(q_F, q) = \emptyset$ pour tout q).

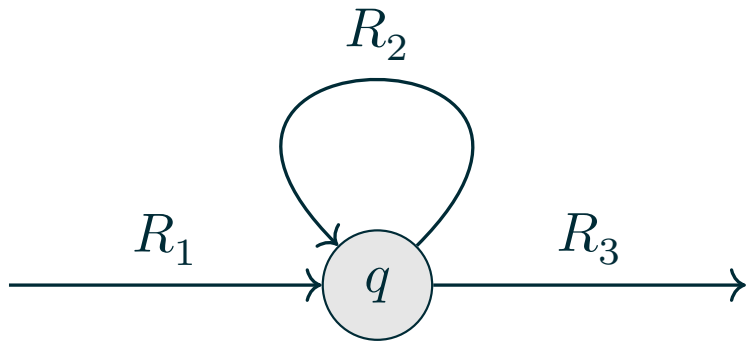
Par convention, une flèche étiquetée $\emptyset$ sera représentée par une absence de flèche.

Pour transformer un AFN en AFNER, on doit

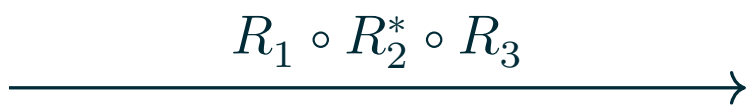
1. Créer un nouvel état de départ avec une flèche ε vers l'ancien état de départ
2. Créer un nouvel état acceptant unique avec des flèches ε depuis les anciens états acceptants
3. Convertir les flèches de type $\mathtt{a}, \mathtt{b}$ en $\mathtt{a} \cup \mathtt{b}$

L'algorithme d'élimination :

1. On transforme l'AFN en AFNER.
2. On choisit un état q à éliminer (sauf q_0 et q_F), et pour toute paire (flèche entrante, flèche sortante), on remplace :



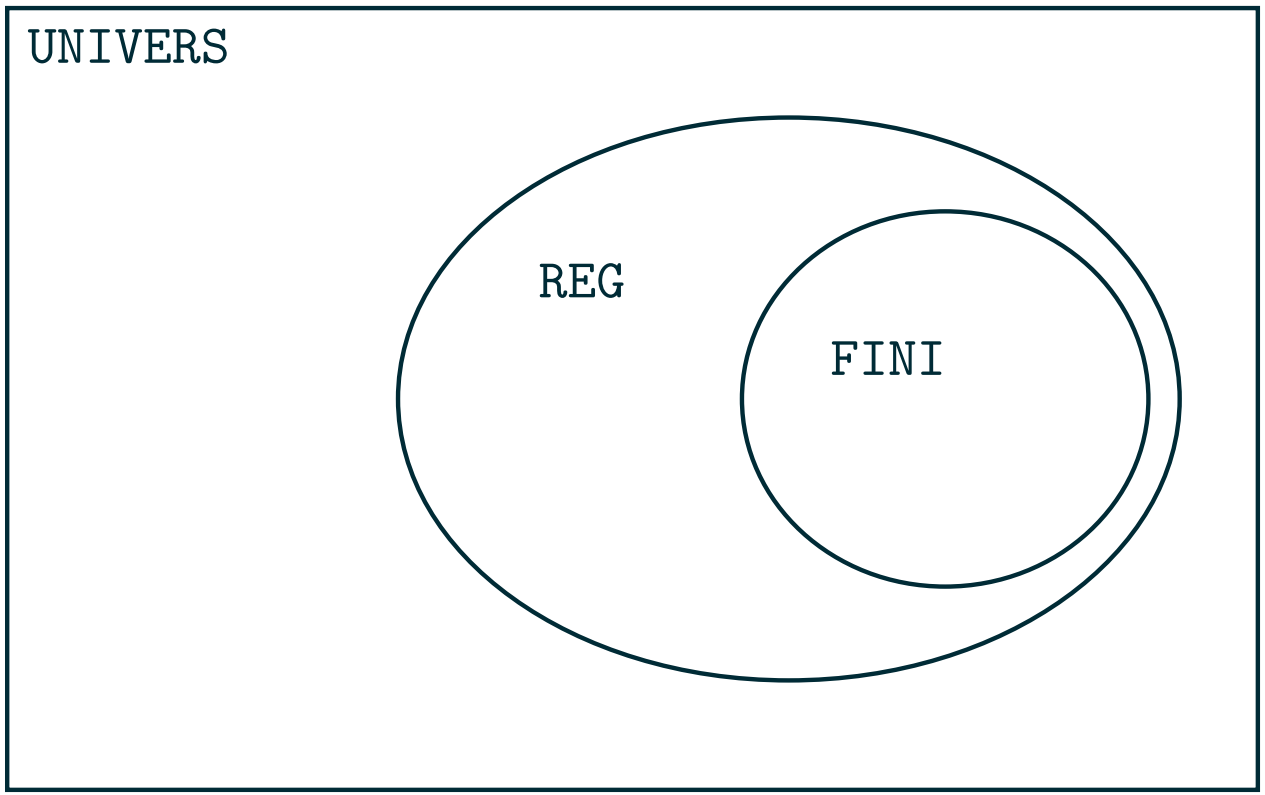
par :



3. Si des flèches parallèles sont créées, on les remplace par une seule flèche avec l'union des flèches parallèles.
4. On répète jusqu'à ce qu'il ne reste que q_0 et q_F
5. Output : l'expression régulière sur la flèche de q_0 à q_F.

□

Hiérarchie



Pour prouver qu'un langage L est régulier

On peut...

- construire un automate qui reconnaît L,
- utiliser les propriétés de fermeture de la classe REG,
- montrer que L est fini.

Théorème 4.72 (Lemme du pompiste)

Soit $L \in$ REG. Il existe un entier $p \geq 1$ tel que pour tout $w \in L$ avec $|w| \geq p$, on peut écrire $w = x \cdot y \cdot z$ où

- $|y| > 0$,
- $|x \cdot y| \leq p$,
- $\forall i \geq 0 \ : \ x \cdot y^i \cdot z \in L$.

Démonstration :

Soit $M = (Q, \Sigma, \delta, q_0, F)$ un AFD qui reconnaît le langage L.

Choisissons $p = \#Q$.

Soit $w \in L$ tel que $|w| \geq p$.

L'automate M va passer par au moins $p + 1$ états avant d'accepter w.

Par le principe du pigeonnier, il existe un état par lequel M va passer au moins deux fois, parmi les $p + 1$ premiers états visités.

Soit $r \in Q$ le premier état par lequel M passe deux fois.

Soit x la partie de w qui est lue entre l'état initial q_0 et la première visite de r.

Soit y la partie de w lue entre les deux premières visites de l'état r.

Soit z la partie restante de w après $x \cdot y$. On a donc :

$$\begin{aligned} w &= x \cdot y \cdot z, \\ |y| &> 0, \\ |x \cdot y| &\leq p, \end{aligned}$$

comme illustré dans la Figure 4.5.

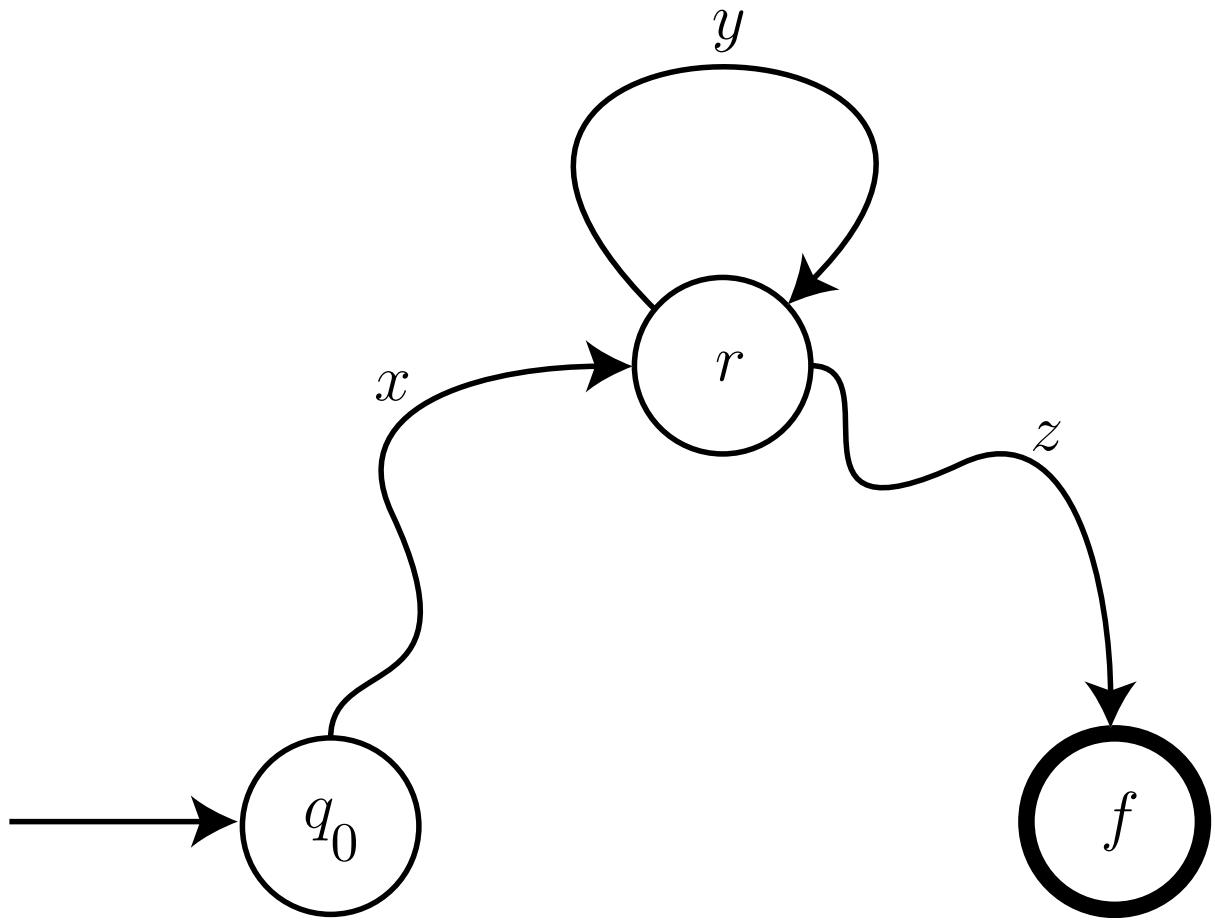


Fig. 4.5 : Illustration du lemme du pompiste.

Il est maintenant clair que les mots suivants seront acceptés par M :

$$x \cdot z$$

$$x \cdot y \cdot z$$

$$x \cdot y \cdot y \cdot z$$

$$x \cdot y \cdot y \cdot y \cdot z$$

$$\vdots$$

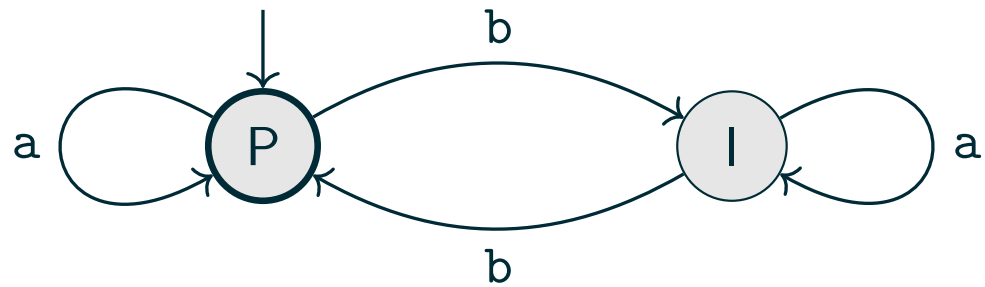
Ces mots appartiennent donc tous à L. □

Exemple 4.73

On sait que le langage

$$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid \ |w|_{\mathtt{b}} \text{ est pair}\}$$

est régulier parce qu'il existe un automate qui le reconnaît :



Cet automate a 2 états.

Par la preuve du lemme du pompiste, on peut prendre $p = 2$.

Considérons le mot $w = \mathtt{abb}$.

On a bien $w \in L$ et $|w| \geq p$.

Choisissons la décomposition $w = x \cdot y \cdot z$ où

$$\begin{aligned} x &= \varepsilon, \\ y &= \mathtt{a}, \\ z &= \mathtt{bb}. \end{aligned}$$

On a bien :

- $|y| > 0$,
- $|x \cdot y| \leq p$,
- $\mathtt{bb},\ \mathtt{abb},\ \mathtt{aabb},\ \mathtt{aaabb},\ \mathtt{aaaabb},\ \ldots \in L$.

□

Pour prouver qu'un langage L n'est pas régulier

On peut...

- utiliser les propriétés de fermeture de la classe REG,
- appliquer le lemme du pompiste.

Théorème 4.74

Le langage

$$L = \{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$$

n'est pas régulier.

Démonstration :

Supposons au contraire que $L \in$ REG, afin d'arriver à une contradiction.

Soit $p \geq 1$ tel que donné par le lemme du pompiste.

Considérons le mot $w = \mathsf{a}^p\mathsf{b}^p$.

On a bien $w \in L$ et $|w| = 2p \geq p$.

Soient $w = x \cdot y \cdot z$, $|y| > 0$, $|x \cdot y| \leq p$, tels que donnés par le lemme du pompiste.

Il est clair que y ne contient que des a.

Considérons $i = 2$ dans l'expression $x \cdot y^i \cdot z$.

Ce mot contient plus de a que de b, et donc $x \cdot y^2 \cdot z \notin L$, en contradiction avec l'énoncé du lemme.

Donc $L \notin$ REG. $\square$

Théorème 4.75

Le langage

$$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid \ |w|_{\mathtt{a}} = |w|_{\mathtt{b}}\}$$

n'est pas régulier.

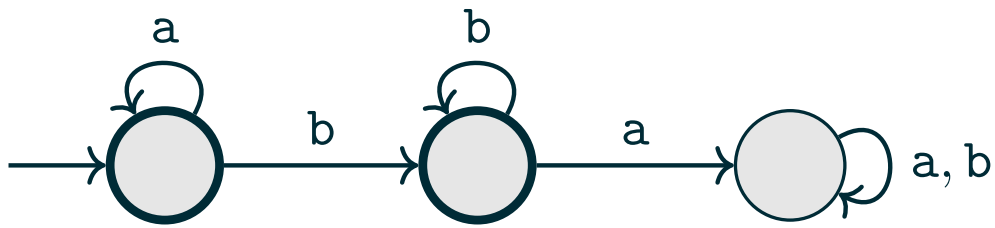
Démonstration :

Supposons au contraire que $L \in$ REG, afin d'arriver à une contradiction.

Considérons le langage

$$R = \{\mathtt{a}^k\mathtt{b}^l \mid k \geq 0, l \geq 0\}.$$

Il est facile de voir que $R \in$ REG, car il est reconnu par l'automate suivant :

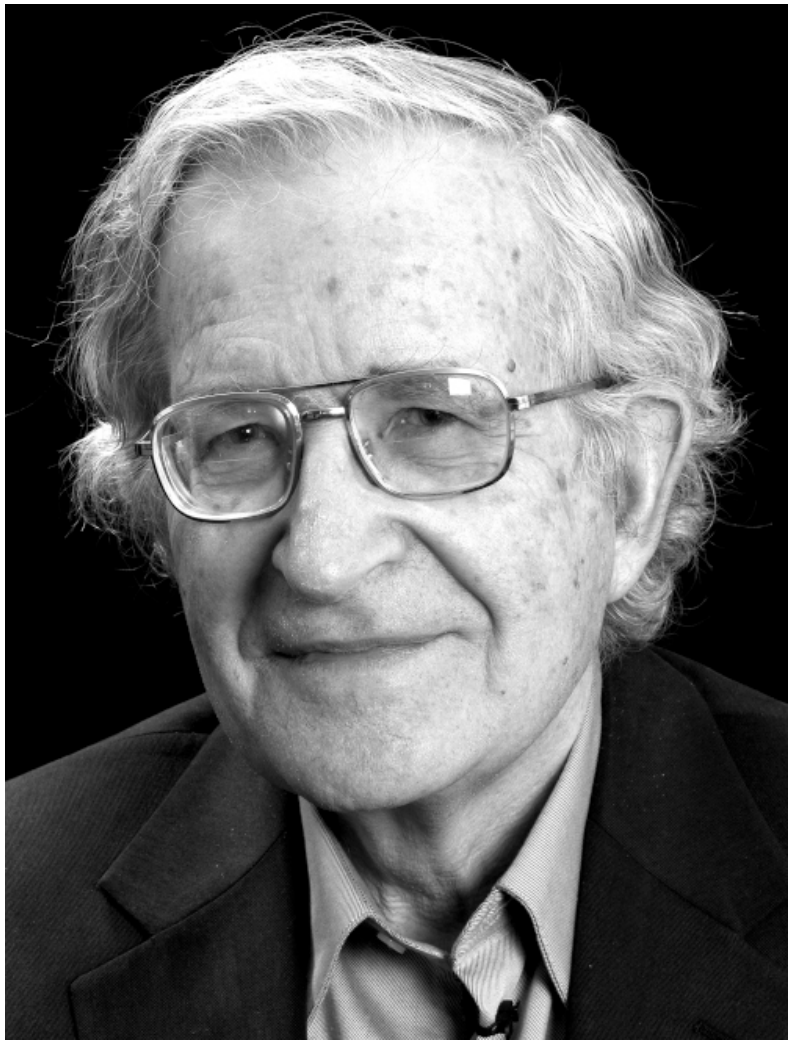


On a donc $L \cap R \in$ REG, par la fermeture de la classe REG par rapport à l'opération d'intersection (théorème 4.60).

Mais $L \cap R = \{\mathsf{a}^n\mathsf{b}^n \mid n \geq 0\}$ n'est pas régulier par le théorème 4.74.

On a une contradiction et le théorème est démontré. □

Grammaires hors-contexte



Exemple de grammaire hors-contexte :

$$A \to \mathtt{0}A\mathtt{1}$$
$$A \to B$$
$$B \to \#$$

Les grammaires hors-contexte ont été définies par Noam Chomsky dans les années 50 :

Définition 4.76 (Grammaire hors-contexte)

Une grammaire hors contexte (GHC) est un quadruplet (V, Σ, R, S) où :

- V est un ensemble fini de variables ;
- Σ est un alphabet, c'est-à-dire un ensemble non vide et fini de symboles appelés terminaux, $V \cap \Sigma = \emptyset$;
- R est un ensemble fini de règles de la forme $v \to z$ où $v \in V$ et $z \in (V \cup \Sigma)^*$;
- $S \in V$ est la variable initiale.

Exemple 4.77

$$G = (V, \Sigma, R, S)$$

où :

$$\begin{aligned} V &= \{S, X\} \\ \Sigma &= \{\mathtt{a}, \mathtt{b}, \mathtt{c}\} \\ R &= \{S \to SS \mid X \mid \mathtt{c}, \quad X \to \mathtt{a}X\mathtt{b} \mid \mathtt{c}\} \end{aligned}$$

$$S \to SS \mid X \mid \mathtt{c}$$
$$X \to \mathtt{a}X\mathtt{b} \mid \mathtt{c}$$

Dérivation du mot cacb :

$$\begin{aligned} S &\Rightarrow SS \\ &\Rightarrow \mathtt{c}S \\ &\Rightarrow \mathtt{c}X \\ &\Rightarrow \mathtt{ca}X\mathtt{b} \\ &\Rightarrow \mathtt{cacb} \end{aligned}$$

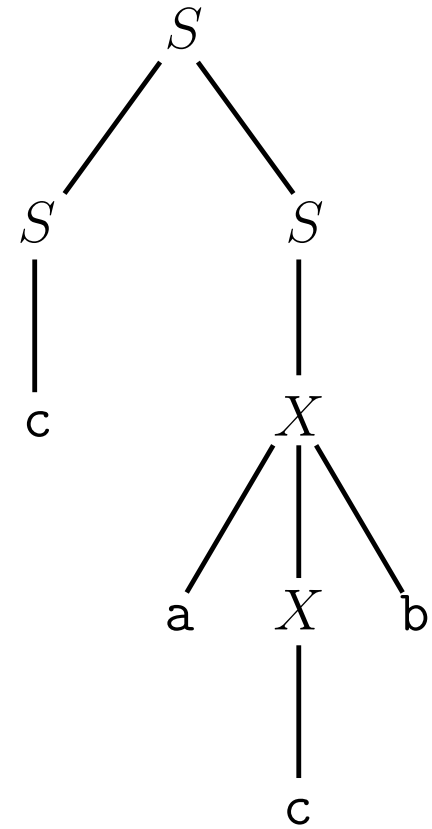


Fig. 4.6 : Arbre de dérivation

Exemple linguistique :

$$\begin{aligned}
\langle \textsc{Phrase} \rangle &\to \langle \textsc{Groupe nominal} \rangle \langle \textsc{Groupe verbal} \rangle \\
\langle \textsc{Groupe nominal} \rangle &\to \langle \textsc{Nom} \rangle \\
\langle \textsc{Groupe verbal} \rangle &\to \langle \textsc{Verbe} \rangle \mid \langle \textsc{Verbe} \rangle \langle \textsc{Complement} \rangle \\
\langle \textsc{Complement} \rangle &\to \langle \textsc{Nom} \rangle \mid \texttt{que}\ \langle \textsc{Phrase} \rangle \\
\langle \textsc{Nom} \rangle &\to \texttt{la table} \mid \texttt{Paulette} \mid \texttt{Jean} - \texttt{Pierre} \\
\langle \textsc{Verbe} \rangle &\to \texttt{voit} \mid \texttt{dit}
\end{aligned}$$

On peut dériver des phrases :

```
la table voit que Paulette dit Jean-Pierre
```

Grammaire HC pour les programmes RÉPÉTER :

$$
\begin{aligned}
S &\to \varepsilon \\
&\mid \langle \textsc{Incrémentation} \rangle S \\
&\mid \langle \textsc{Affectation} \rangle S \\
&\mid \langle \textsc{Répéter} \rangle S \\
\langle \textsc{Incrémentation} \rangle &\to \texttt{inc}(V) \\
\langle \textsc{Affectation} \rangle &\to V \leftarrow V \\
\langle \textsc{Répéter} \rangle &\to \texttt{répéter } V \texttt{ fois } [S] \\
V &\to r_N \\
N &\to C | CN \\
C &\to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}
$$

Définition 4.78

Soit $G = (V, \Sigma, R, S)$ une GHC et soient $u, v, w \in (V \cup \Sigma)^*$.

On dit que uAv donne ou produit ou engendre uwv, noté

$$uAv \Rightarrow uwv,$$

si $A \to w \in R$.

On note $u \overset{n}{\Rightarrow} v$ si u donne v en n étapes :

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_n = v.$$

Aussi, $u \overset{*}{\Rightarrow} v$ s'il existe un $n \geq 0$ tel que $u \overset{n}{\Rightarrow} v$.

Définition 4.79

Soit $G = (V, \Sigma, R, S)$ une GHC.

$$L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$$

est le langage engendré par G.

Définition 4.80

Un langage L est hors contexte s'il existe une GHC G telle que $L = L(G)$.

Définition 4.81

La classe des langages hors contexte est notée HC.

Exemple 4.82

Soit $G = (\{S\}, \{\mathtt{a}, \mathtt{b}\}, R, S)$ *une GHC où*

$$R = \{S \to \mathtt{a}S\mathtt{b} \mid \varepsilon\}.$$

Alors

$$\begin{aligned} L(G) &= \{\varepsilon, \mathtt{ab}, \mathtt{aabb}, \mathtt{aaabbb}, \ldots\} \\ &= \{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}. \end{aligned}$$

Dérivation du mot aaabbb *:*

$$\begin{aligned} S &\Rightarrow \mathtt{a}S\mathtt{b} \\ &\Rightarrow \mathtt{aa}S\mathtt{bb} \\ &\Rightarrow \mathtt{aaa}S\mathtt{bbb} \\ &\Rightarrow \mathtt{aaabbb} \end{aligned}$$

Exemple 4.83

Soit $G = (V, \Sigma, R, S)$ une GHC où

$$\begin{aligned} V &= \{S, M, N\} \\ \Sigma &= \{0, 1, \ldots, 9, +, *, (,)\} \end{aligned}$$

$$R : \begin{cases} S & \to & N \mid (S{+}S) \mid (S{*}S) \\ N & \to & 0 \mid 1M \mid 2M \mid \ \ldots \ \mid 9M \\ M & \to & \varepsilon \mid 0M \mid 1M \mid \ \ldots \ \mid 9M \end{cases}$$

La variable N donne les nombres :

$$N \ \Rightarrow \ 2M \ \Rightarrow \ 23M \ \Rightarrow \ 23.$$

La variable S donne les expressions arithmétiques :

$$S \ \Rightarrow \ (S{+}S) \ \Rightarrow \ (S{+}(S{*}S)) \ \overset{*}{\Rightarrow} \ (N{+}(N{*}N)) \ \overset{*}{\Rightarrow} \ (123 + (245 * 19))$$

Proposition 4.84

Soit $G = (\{S\}, \{\mathtt{a}, \mathtt{b}\}, R, S)$ *une GHC où*

$$R \ : \ S \to \mathtt{a}S\mathtt{b} \mid \mathtt{b}S\mathtt{a} \mid SS \mid \varepsilon,$$

et soit

$$L = \{w \mid \ |w|_{\mathtt{a}} = |w|_{\mathtt{b}}\},$$

alors

$$L(G) = L.$$

Démonstration :

Il est facile de voir que $L(G) \subseteq L$ car toutes les règles de G produisent un nombre égal de a et de b.

Il reste à montrer que $L \subseteq L(G)$.

Soit $w \in L$.

La preuve fonctionne par induction sur la longueur $n = |w|$.

Base de l'induction. Soit $n = 0$. Alors :

$$\begin{aligned} S &\Rightarrow \varepsilon \\ &= w, \end{aligned}$$

donc $w \in L(G)$.

Hypothèse d'induction. Pour $n > 0$, supposons que

$$\forall z \in L, |z| < n \ : \ S \overset{*}{\Rightarrow} z.$$

Pas d'induction. Remarquons que tous les mots de L ont un nombre pair de symboles.

Considérons 4 cas pour w :

1. $w = \mathtt{a}z\mathtt{b}$. Dans ce cas, on a la dérivation :

$$\begin{array}{lcll} S & \Rightarrow & \mathtt{a}S\mathtt{b} & \\ & \overset{*}{\Rightarrow} & \mathtt{a}z\mathtt{b} & (\text{par hypothèse d'induction}) \\ & = & w. & \end{array}$$

2. $w = \mathtt{b}z\mathtt{a}$:

$$\begin{array}{rcll} S & \Rightarrow & \mathtt{b}S\mathtt{a} & \\ & \overset{*}{\Rightarrow} & \mathtt{b}z\mathtt{a} & (\text{par hypothèse d'induction}) \\ & = & w. & \end{array}$$

3. $w = \mathtt{a}z\mathtt{a}$. Dans ce cas, z contient deux $\mathtt{b}$ de plus que de $\mathtt{a}$, et on peut donc le décomposer en $z = z_1 \cdot z_2$, où z_1 et z_2 contiennent chacun un $\mathtt{b}$ de plus que de $\mathtt{a}$. D'où :

$$
\begin{aligned}
S &\Rightarrow SS \\
&\overset{*}{\Rightarrow} \mathtt{a}z_1 S \quad (\text{car } \mathtt{a}z_1 \in L \text{ et par hypothèse d'induction}) \\
&\overset{*}{\Rightarrow} \mathtt{a}z_1 z_2 \mathtt{a} \quad (\text{car } z_2\mathtt{a} \in L \text{ et par hypothèse d'induction}) \\
&= \mathtt{a}z\mathtt{a} \\
&= w.
\end{aligned}
$$

4. $w = \mathtt{b}z\mathtt{b}$. Ce cas se traite comme le cas 3, *mutatis mutandis*.

Dans tous les cas, on a $w \in L(G)$. □

Construction de grammaires

Construire une GHC pour un langage donné demande souvent de la créativité.

Cependant, il y a un certain nombre de techniques qui aident beaucoup. Entre autres :

- décomposer le langage en l'union de deux langages plus simples ;
- utiliser des règles qui produisent des listes ;

comme dans l'exemple qui suit.

Exemple 4.85

Soit

$$L = \{\mathsf{a}^i\mathsf{b}^j\mathsf{a}^k\mathsf{b}^l \mid (i = k \text{ ou } i = l),\ j \geq 0,\ k \geq 1,\ l \geq 1\},$$

c'est-à-dire :

$$\begin{aligned} L &= \{\mathsf{a}^i\mathsf{b}^j\mathsf{a}^i\mathsf{b}^l \mid i \geq 1,\ j \geq 0,\ l \geq 1\} \\ &\cup \{\mathsf{a}^i\mathsf{b}^j\mathsf{a}^k\mathsf{b}^i \mid i \geq 1,\ j \geq 0,\ k \geq 1\}. \end{aligned}$$

Une GHC pour L est donnée par :

$$G = (\{S, S_1, S_1', S_2, S_2', A, B\},\ \{\mathtt{a}, \mathtt{b}\},\ R,\ S)$$

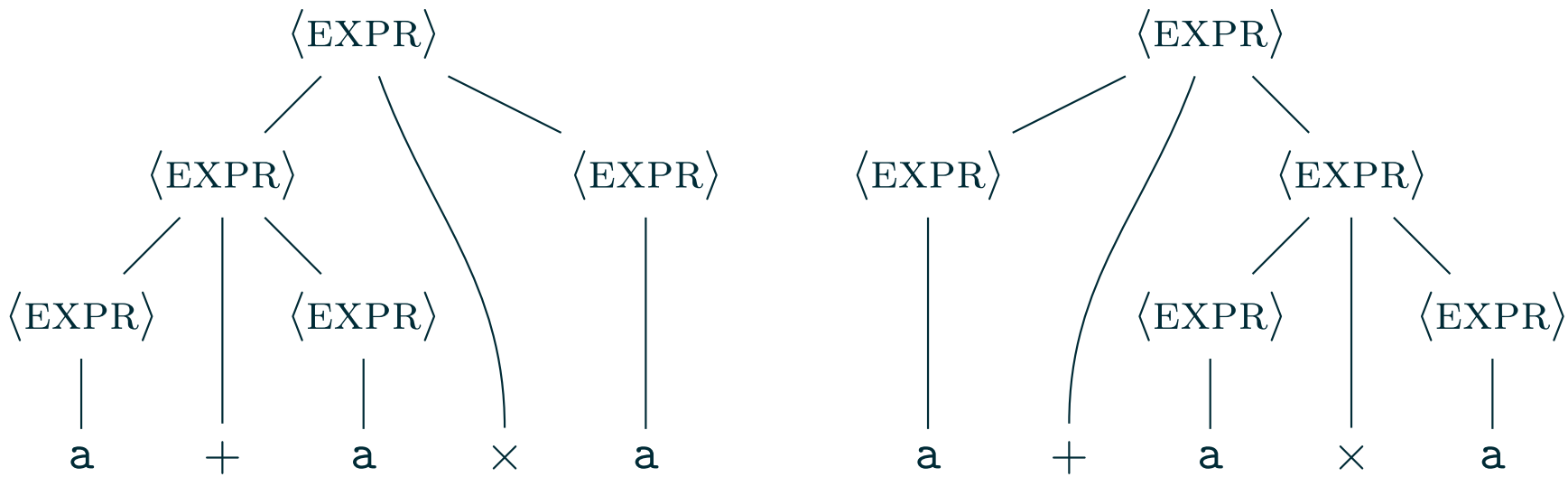
avec

$$R\ :\ \begin{cases} S & \to & S_1 \mid S_2 \\ S_1 & \to & S_1'\mathtt{b}B \\ S_1' & \to & \mathtt{a}S_1'\mathtt{a} \mid \mathtt{a}B\mathtt{a} \\ B & \to & \mathtt{b}B \mid \varepsilon \\ S_2 & \to & \mathtt{a}S_2'\mathtt{b} \\ S_2' & \to & \mathtt{a}S_2'\mathtt{b} \mid B\mathtt{a}A \\ A & \to & \mathtt{a}A \mid \varepsilon \end{cases}$$

Ambiguïté dans les GHC

Il est possible d'avoir plusieurs dérivations d'une même mot :

$$\begin{aligned}\langle \text{EXPR} \rangle \rightarrow\ & \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle \\ \mid\ & \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle \\ \mid\ & (\langle \text{EXPR} \rangle) \\ \mid\ & \texttt{a}\end{aligned}$$



Forme normale de Chomsky

Définition 4.86

La GHC $G = (V, \Sigma, R, S)$ est sous forme normale de Chomsky (FNC) si toutes ses règles sont de la forme

$$\begin{aligned} A &\to BC \text{ ou} \\ A &\to x, \end{aligned}$$

avec

$$\begin{aligned} A, B, C &\in V, \\ B &\neq S, \\ C &\neq S, \\ x &\in \Sigma, \end{aligned}$$

en permettant aussi

$$S \to \varepsilon.$$

Théorème 4.87

Il existe un algorithme qui prend en entrée une GHC G et qui retourne une grammaire en FNC G' telle que $L(G) = L(G')$.

Algorithme 4.88

Prendre en entrée : $G = (V, \Sigma, R, S)$, une GHC.

1. *(INIT) Ajouter S_0 à V, la nouvelle variable initiale, et ajouter la règle $S_0 \to S$ à R.*
2. *(TERM) Éliminer les règles avec des terminaux non-isolés :*
   - 2.1 *Pour tout terminal a apparaissant à droite d'une règle avec d'autres symboles, on ajoute une variable N_a et la règle $N_a \to a$.*
   - 2.2 *On remplace toute règle de forme $A \to X_1 \cdots a \cdots X_n$ par $A \to X_1 \cdots N_a \cdots X_n$.*
3. *(BIN) Éliminer de R toutes les règles de la forme $A \to u_1 u_2 \ldots u_k$ où $k \geq 3$ et où $u_1, u_2, \ldots, u_k \in V \cup \Sigma$:*
   - 3.1 *Enlever $A \to u_1 u_2 \ldots u_k$.*
   - 3.2 *Ajouter $A_1,\ A_2,\ \ldots,\ A_{k-2}$ à V.*
   - 3.3 *Ajouter les règles $A \to u_1 A_1,\ A_1 \to u_2 A_2,\ \ldots,\ A_{k-2} \to u_{k-1} u_k$.*

4. (VIDE) Éliminer de R toutes les règles de la forme $A \to \varepsilon$, sauf possiblement la règle $S_0 \to \varepsilon$:
   - 4.1 Déterminer toutes les variables « annulables » $A \in V$ telles que $A \overset{*}{\Rightarrow} \varepsilon$.
   - 4.2 Pour toute règle de type $A \to BC$, ajouter $A \to C$ si B est annulable et $A \to B$ si C est annulable.
   - 4.3 Si S_0 est annulable, ajouter $S_0 \to \varepsilon$; éliminer toutes les autres règles de type $A \to \varepsilon$.
5. (UNAIRE) Éliminer les règles de type $A \to B$:
   - 5.1 On enlève $A \to B$
   - 5.2 Pour chaque règle de type $B \to w$ (où $w \in (\Sigma \cup V)^*$), on ajoute $A \to w$, sauf si cette a déjà été enlevée, ou est égale à $A \to A$.

Produire en sortie la grammaire ainsi modifiée.

□

Exemple 4.89

$$G = (\{S, A, B\}, \{\mathtt{a}, \mathtt{b}\}, R, S)$$

où

$$R : \begin{cases} S & \to & ASA|\mathtt{a}B \\ A & \to & B|S \\ B & \to & \mathtt{b}|\varepsilon. \end{cases}$$

1. (INIT) Ajouter une nouvelle variable initiale

Avant :

$$S \to ASA \mid \mathtt{a}B$$
$$A \to B \mid S$$
$$B \to \mathtt{b} \mid \varepsilon$$

Après :

$$S_0 \to S$$
$$S \to ASA \mid \mathtt{a}B$$
$$A \to B \mid S$$
$$B \to \mathtt{b} \mid \varepsilon$$

2. (TERM) Éliminer les règles avec des non-terminaux non-isolés :

Avant :

$$
\begin{aligned}
S_0 &\to S \\
S &\to ASA \mid \mathtt{a}B \\
A &\to B \mid S \\
B &\to \mathtt{b} \mid \varepsilon
\end{aligned}
$$

Après :

$$
\begin{aligned}
S_0 &\to S \\
S &\to ASA \mid N_aB \\
A &\to B \mid S \\
B &\to \mathtt{b} \mid \varepsilon \\
N_a &\to \mathtt{a}
\end{aligned}
$$

3. (BIN) Éliminer les règles à plus de deux symboles à droite :

Avant :

$$
\begin{aligned}
S_0 &\to S \\
S &\to ASA \mid N_a B \\
A &\to B \mid S \\
B &\to \mathtt{b} \mid \varepsilon \\
N_a &\to \mathtt{a}
\end{aligned}
$$

Après :

$$
\begin{aligned}
S_0 &\to S \\
S &\to AM \mid N_a B \\
M &\to SA \\
A &\to B \mid S \\
B &\to \mathtt{b} \mid \varepsilon \\
N_a &\to \mathtt{a}
\end{aligned}
$$

4. (VIDE) Éliminer les règles $A \to \varepsilon$:

Avant :

$$
\begin{aligned}
S_0 &\to S \\
S &\to AM \mid N_a B \\
M &\to SA \\
A &\to B \mid S \\
B &\to \mathtt{b} \mid \varepsilon \\
N_a &\to \mathtt{a}
\end{aligned}
$$

Variables annulables : A, B

Après :

$$
\begin{aligned}
S_0 &\to S \\
S &\to AM \mid M \mid N_a B \mid N_a \\
M &\to SA \mid S \\
A &\to B \mid S \\
B &\to \mathtt{b} \\
N_a &\to \mathtt{a}
\end{aligned}
$$

5. (UNAIRE) Éliminer les règles $A \to B$:

Avant :

$$
\begin{aligned}
S_0 &\to S \\
S &\to AM \mid M \mid N_a B \mid N_a \\
M &\to SA \mid S \\
A &\to B \mid S \\
B &\to \texttt{b} \\
N_a &\to \texttt{a}
\end{aligned}
$$

Après :

$$
\begin{aligned}
S_0 &\to AM \mid SA \mid N_a B \mid \texttt{a} \\
M &\to SA \mid AM \mid N_a B \mid \texttt{a} \\
B &\to \texttt{b}
\end{aligned}
\qquad
\begin{aligned}
S &\to AM \mid SA \mid N_a B \mid \texttt{a} \\
A &\to \texttt{b} \mid AM \mid SA \mid N_a B \mid \texttt{a} \\
N_a &\to \texttt{a}
\end{aligned}
$$

Théorème 4.90

L'algorithme 4.88 produit une grammaire de taille $O(n^2)$, où n est la taille de la grammaire initiale (somme du nombre de symboles à droite des règles).

Démonstration :

1. Ajout de S_0 : on ajoute une seule règle.
2. Élimination des règles avec non-terminaux isolés : dans le pire cas, on ajoute $|\Sigma|$ règles, ce qui est plus petit que n si tous les éléments de Σ sont utilisés $\Rightarrow O(n)$ jusqu'à maintenant.
3. Élimination des règles à plus de deux symboles : dans le pire cas, on double n, donc toujours linéaire jusqu'ici.
4. Élimination des ε : dans le pire cas, toutes les règles de forme $A \to BC$ deviennent $A \to BC \mid B \mid C$, donc encore là on double n au pire.

4. Élimination des règles $A \to B$: au final, tout ce qui est à droite d'une règle doit déjà avoir été présent avant, possiblement avec une autre variable à gauche. Il y a au plus n variables, et les règles ayant une même variable A à gauche ont au total au plus n variables à droite. La grammaire finale est donc de taille au plus $O(n^2)$.

Les remarques 4.91 et 4.92 sont cruciales pour la compréhension de l'algorithme 4.94.

Remarque 4.91

Si $G = (V, \Sigma, R, S)$ est sous FNC, et si $\varepsilon \in L(G)$, alors la seule façon de produire ε est

$$S \Rightarrow \varepsilon.$$

Remarque 4.92

Si $G = (V, \Sigma, R, S)$ est sous FNC, si $x \in \Sigma$ et $x \in L(G)$, alors la seule façon de produire x est

$$A \Rightarrow x$$

où $A \in V$.

Appartenance d'un mot à un langage hors contexte

Théorème 4.93

Il existe un algorithme qui prend en entrée un mot w et une GHC G en FNC et qui détermine si $w \in L(G)$. Pour une grammaire G fixée, l'algorithme fonctionne en temps $O(n^3)$, où n est la taille du mot donné.

Aperçu de la démonstration :

Nous donnons l'algorithme et argumentons pour son temps d'exécution.

Algorithme 4.94 (Cocke, Younger, Kasami (CYK))

Prendre en entrée : $G = (V, \Sigma, R, S)$, *une GHC, et* $w \in \Sigma^*$. *Soit* $w = w_1 w_2 \ldots w_n$, *avec* $w_1, \ldots, w_n \in \Sigma$.

1. *Si* $w = \varepsilon$, *alors :*
   si $(S_0 \to \varepsilon) \in R'$, *alors retourner* $\langle \textsc{Vrai} \rangle$,
   sinon retourner $\langle \textsc{Faux} \rangle$.
2. *Si* $w \neq \varepsilon$:

   Pour $1 \leq i \leq j \leq n$, *soit* $f(i, j)$ *l'ensemble des variables dans* V' *qui engendrent le sous-mot* $w_i \ldots w_j$.

   Calculer $f(1, n)$ *en appelant la sous-routine décrite à l'étape 3.*

   Si $S_0 \in f(1, n)$, *alors retourner* $\langle \textsc{Vrai} \rangle$, *sinon retourner* $\langle \textsc{Faux} \rangle$.

3. Calcul de $f(i,j)$, pour $1 \leq i \leq j \leq n$ donnés.

   Conserver les résultats obtenus au fur et à mesure. Si $f(i,j)$ a déjà été calculé, alors retourner le résultat tout de suite, sans faire de calculs.

   Si $i = j$, alors $f(i,i) \leftarrow \{A \in V' \mid (A \to w_i) \in R'\}$.

   Si $i < j$, alors :

   $f(i,j) \leftarrow \emptyset$,
   pour $k \leftarrow i$ à $j-1$ faire :
   pour toute règle $A \to BC$ faire :
   si $B \in f(i,k)$ et $C \in f(k+1,j)$, alors :
   $F \leftarrow F \cup \{A\}$

Pour une grammaire G fixée, et si la grammaire G' est calculée une fois pour toutes, le temps d'exécution de l'algorithme 4.94 sur un mot de longueur n est dominé par le nombre d'appels à la fonction f de l'étape 3.

La fonction récursive f est équivalente à un algorithme de programmation dynamique qui consiste à remplir une table de taille $O(n^2)$ où chaque élément prend un temps $O(n)$ à calculer. Le temps d'exécution total est dans $O(n^3)$. □

Théorème 4.95

$$\text{REG} \subseteq \text{HC}$$

Démonstration :

Soit le langage $L \in \text{REG}$ et soit $M = (Q, \Sigma, \delta, q_0, F)$ un automate qui le reconnaît.

Alors on peut voir que la GHC $G = (V, \Sigma, R, S)$ où

$$\begin{aligned} V &= Q, \\ R &= \{(A \to xB) \mid A \in Q,\ x \in \Sigma,\ B \in Q,\ \delta(A, x) = B\} \\ &\quad \cup \{(A \to \varepsilon) \mid A \in F\}, \\ S &= q_0, \end{aligned}$$

engendre L. □

Corollaire 4.96

La classe REG *est strictement incluse dans la classe* HC*.*

Démonstration :

Soit $L = \{\mathsf{a}^n\mathsf{b}^n \mid n \geq 0\}$.

Nous avons déjà vu que $L \in$ HC et $L \notin$ REG. □

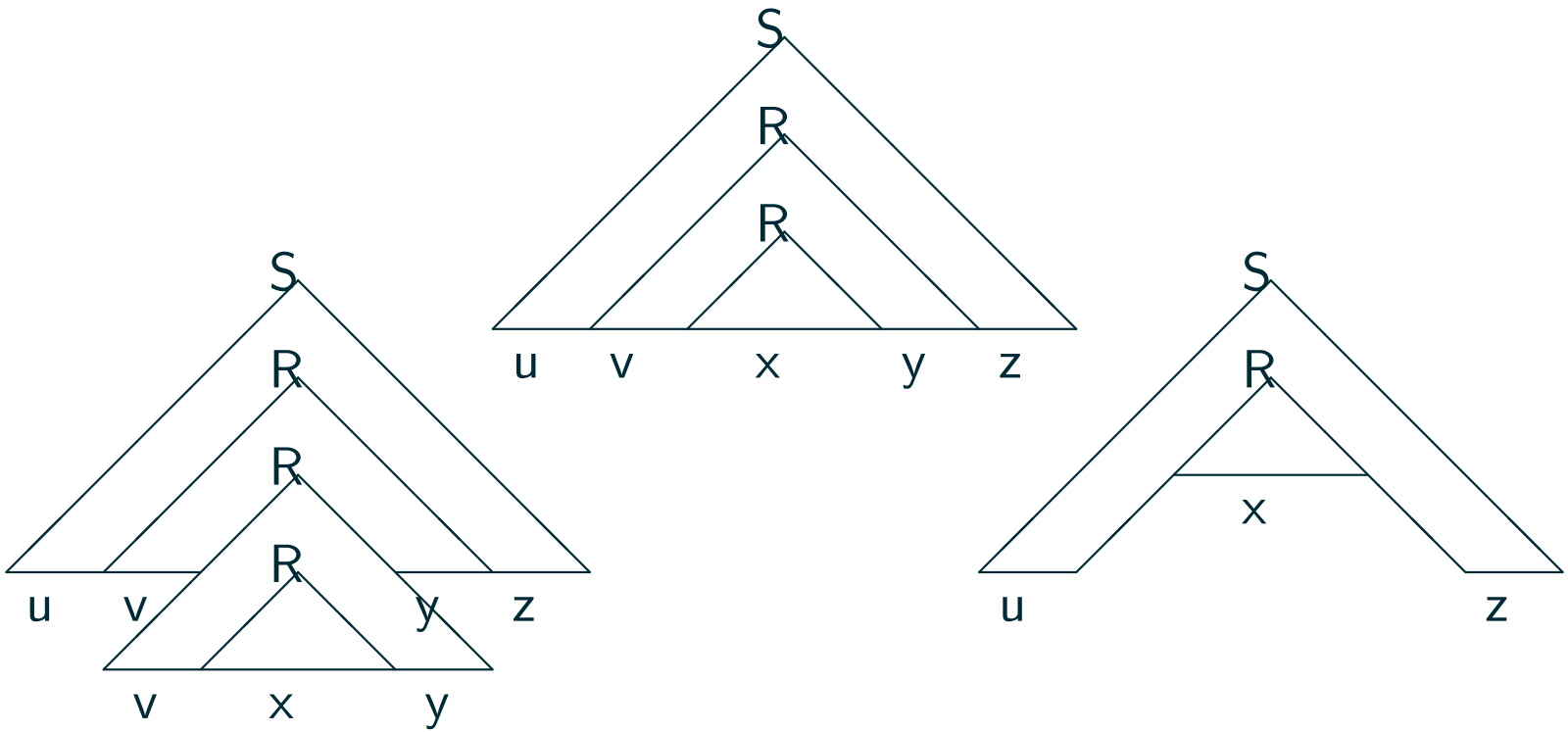
Théorème 4.97 (Lemme du pompiste pour les langages HC)

Soit $L \in$ HC. Il existe un entier $p \geq 1$ tel que pour tout $w \in L$ avec $|w| \geq p$, on peut écrire $w = uvxyz$, où

- $|vy| > 0$,
- $|vxy| \leq p$,
- $\forall i \geq 0 \ : \ uv^i xy^i z \in L$.

Aperçu de la démonstration :

Si un mot $w \in L$ est suffisamment long, alors il aura un arbre de dérivation très haut. Dans cet arbre, il y aura un chemin, du sommet jusqu'à la base, avec une répétition de variable :



Démonstration du Théorème 4.97 :

Soit G une GHC en FNC telle que $L(G) = L$, et choisissons $p = 2^{|V|}$.

Considérons un arbre de dérivation pour w, et soit $t_0, \cdots, t_h$ le chemin le plus long de la racine à une feuille dans cet arbre. Pour tout sommet t, dénotons par $s(t)$ la variable associée et par $w(t)$ le mot engendré par le sous-arbre avec t comme racine.

Supposons d'abord que $h \geqslant |V| + 1$, et regardons les $|V| + 2$ derniers sommets. Il y a au plus $|V| + 1$ variables parmi eux et donc il y aura un i et un j tels que $i < j$ et $s(t_i) = s(t_j) = R \in V$ par le principe du pigeonnier.

Alors, définissons

$$x := w(t_j)$$
$$vxy := w(t_i)$$
$$uvxyz := w(t_0)$$

On a donc que

$$R \overset{*}{\Rightarrow} x$$
$$R \overset{*}{\Rightarrow} vRy$$
$$S \overset{*}{\Rightarrow} uRz$$

On peut maintenant prouver les trois propriétés promises par le lemme :

1. Clairement $|vy| > 0$, car G est en FNC et qu'il est impossible de dériver $R \overset{*}{\Rightarrow} R$ sauf trivialement en 0 étapes.
2. $|vxy| \leqslant p$, car on a choisi t_i et t_j parmi les $|V| + 2$ derniers sommets. Le sous-arbre engendré par t_i a donc une hauteur d'au plus $|V| + 1$ et donc $|w(t_i)| = |vxy| \leqslant 2^{|V|} = p$.
3. On peut dériver

   $$S \overset{*}{\Rightarrow} uRz \overset{*}{\Rightarrow} uxz$$

   ainsi que

   $$S \overset{*}{\Rightarrow} uRz \overset{*}{\Rightarrow} uvRyz \overset{*}{\Rightarrow} uv^2Ry^2z \overset{*}{\Rightarrow} \cdots \overset{*}{\Rightarrow} uv^iRy^iz \overset{*}{\Rightarrow} uv^ixy^iz$$

   donc $uv^ixy^iz \in L$ pour tout $i \geqslant 0$.

Comment s'assurer que $h \geqslant |V| + 1$?

À chaque niveau, on double, sauf au dernier niveau où on a des transitions de type $A \Rightarrow a$. Donc, $|w| \leqslant 2^{h-1}$.

Donc, puisqu'on a choisi $p = 2^{|V|}$, on a que

$$p = 2^{|V|} \leqslant |w| \leqslant 2^{h-1},$$

et donc

$$h \geqslant |V| + 1.$$

Ceci conclut donc la démonstration. □

Exemple 4.98

Soit $L = \{\mathtt{a}^n\mathtt{b}^n \mid n \geqslant 0\}$. *Nous avons vu que ce langage est HC, et une grammaire en FNC à 5 variables permet de l'engendrer.*

Donc, le lemme du pompiste HC devrait fonctionner pour $p = 2^5 = 32$. *Et effectivement, pour tout* $w = \mathtt{a}^k\mathtt{b}^k$ *pour* $k \geqslant 16$, *on peut écrire*

$$w = uvxyz$$
$$x = \mathtt{ab}$$
$$v = \mathtt{a}$$
$$y = \mathtt{b}$$
$$u = \mathtt{a}^{k-2}$$
$$z = \mathtt{b}^{k-2}$$

et toutes les propriétés du lemme sont satisfaites.

Théorème 4.99

Le langage

$$L = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 0\}$$

n'est pas hors contexte.

Démonstration :

Supposons au contraire que $L \in$ HC, afin d'arriver à une contradiction.

Soit $p \geq 1$ tel que donné par le lemme du pompiste pour les langages hors contexte.

Considérons le mot $w = \mathtt{a}^p\mathtt{b}^p\mathtt{c}^p$.

On a bien $w \in L$ et $|w| = 3p \geq p$.

Soient $w = uvxyz$, $|vy| > 0$ et $|vxy| \leq p$, tels que donnés par le lemme.

Considérons les deux cas suivants :

1. La partie v, ou y, contient plus d'une sorte de symboles parmi a, b et c.

   Choisissons $i = 2$ dans l'expression $uv^i xy^i z$.

   Ce mot n'est pas de la forme $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n$, et donc $uv^i xy^i z \notin L$, en contradiction avec l'énoncé du lemme.

2. La partie v ne contient qu'une seule sorte de symbole, de même que y.

   Choisissons $i = 0$ dans l'expression uv^ixy^iz, c'est-à-dire uxz.

   Ce mot ne contient pas un nombre égal des trois symboles et n'est pas de la forme $\mathsf{a}^n\mathsf{b}^n\mathsf{c}^n$, et donc $uv^ixy^iz \notin L$, de nouveau en contradiction avec l'énoncé.

Donc $L \notin$ HC. □

Théorème 4.100

La classe HC *est fermée pour l'union.*

Démonstration :

Soient

$$L_1 = L(G_1) \text{ où } G_1 = (V_1, \Sigma, R_1, S_1)$$

et

$$L_2 = L(G_2) \text{ où } G_2 = (V_2, \Sigma, R_2, S_2),$$

en supposant que V_1 et V_2 soient disjoints, tout comme R_1 et R_2.

Alors la GHC suivante :

$$G = (V_1 \cup V_2 \cup \{S\},\ \Sigma,\ R_1 \cup R_2 \cup \{S \to S_1 \mid S_2\},\ S)$$

engendre le langage $L_1 \cup L_2$. □

Théorème 4.101

La classe HC *n'est pas fermée pour l'intersection.*

Démonstration :

Soient

$$\begin{aligned} L_1 &= \{\mathtt{a}^m\mathtt{b}^n\mathtt{c}^n \mid m \geq 0, n \geq 0\}, \\ L_2 &= \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^m \mid m \geq 0, n \geq 0\}. \end{aligned}$$

Alors

$$L_1 \cap L_2 = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 0\}.$$

Or, on a vu au théorème 4.99 que ce langage n'est pas hors contexte. □

Théorème 4.102

La classe des langages hors contexte n'est pas fermée pour la complémentation.

Démonstration :

Supposons au contraire que la classe est fermée afin d'arriver à une contradiction.

Soient $L_1 \in$ HC et $L_2 \in$ HC.

En considérant

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

et que la classe HC est fermée pour l'union (théorème 4.100), on obtient une contradiction avec le fait que HC n'est pas fermée pour l'intersection (théorème 4.101). □

Théorème 4.103

La classe HC *est fermée sous la concaténation*

Démonstration :

Soient L_1 et L_2 deux langages hors-contexte ; nous devons prouver que $L_1 \circ L_2 \in$ HC. Soient $G_1 = (V_1, \Sigma, R_1, S_1)$ et $G_2 = (V_2, \Sigma, R_2, S_2)$ des grammaires engendrant L_1 et L_2 respectivement. Définissons $G = (V, \Sigma, R, S)$ pour $L_1 \circ L_2$ comme suit :

$$V = V_1 \cup V_2 \cup \{S\}$$
$$R = R_1 \cup R_2 \cup \{S \to S_1 S_2\}.$$

□

Théorème 4.104

La classe HC *est fermée sous l'étoile.*

Démonstration :
Soit $L \in$ HC, et soit $G = (V, \Sigma, R, S)$ une grammaire en FNC engendrant L. Alors, définissons la grammaire $G' = (V', \Sigma, R', S')$ engendrant L^* comme suit :

$$V' = V$$
$$S' = S$$
$$R' = R \cup \{S \to \varepsilon, S \to SS\}$$

Notons que vu que G est en FNC, S ne peut être utilisé qu'au début de la dérivation, et les deux nouvelles règles ajoutées ne peuvent pas être utilisées « à l'intérieur » d'une dérivation dans G. □

Théorème 4.105

Si $L_1 \in$ HC *et* $L_2 \in$ REG, *alors* $L_1 \cap L_2 \in$ HC.

Démonstration :

Soit $G = (V, \Sigma, R, S)$ une grammaire hors-contexte en FNC pour L_1 et $M = (Q, \Sigma, \delta, q_0, F)$ un AFD reconnaissant L_2.

Nous allons construire une grammaire $G' = (V', \Sigma, R', S')$ engendrant $L_1 \cap L_2$.

Soient :

$$V' = (V \times Q \times Q) \cup \{\langle \textsc{init} \rangle\}$$
$$S' = \langle \textsc{init} \rangle$$

Le but sera de créer R' de façon à ce que la variable (A, q_i, q_j) puisse engendrer les mots pouvant être engendrés par G à partir de A, et reconnus par M en passant de l'état q_i à q_j.

Construisons R' de la façon suivante :

- Pour tout $q \in F$: on ajoute la règle

$$\langle \text{INIT} \rangle \to (S, q_0, q)$$

- Si la règle $S \to \varepsilon$ existe dans G et que $q_0 \in F$, alors on ajoute la règle $\langle \text{INIT} \rangle \to \varepsilon$.
- Pour toute règle de forme $A \to a$ et tout $q \in Q$, on ajoute la règle

$$(A, q, \delta(q, a)) \to a$$

- Pour toute règle de forme $A \to BC$ et pour tout $q_1, q_2, q_3 \in Q$, on ajoute :

$$(A, q_1, q_3) \to (B, q_1, q_2)(C, q_2, q_3)$$

□

Les langues naturelles sont-elles hors-contexte ?

L'idée originale de Chomsky était d'utiliser les grammaires hors-contexte pour décrire les langues naturelles.

Dans plusieurs langues, il existe des phrases sur lesquelles on peut utiliser le lemme du pompiste pour infirmer cette théorie.

Considérons la phrase :

Pierre, Jean, Pierrette, Jeannette, [...] sont respectivement chauve, roux, myope, presbyte, [...]

On retrouve la structure :

$\langle \mathrm{NM} \rangle \langle \mathrm{NM} \rangle \langle \mathrm{NF} \rangle \langle \mathrm{NF} \rangle \cdots$ `sont respectivement` $\langle \mathrm{AM} \rangle \langle \mathrm{AM} \rangle \langle \mathrm{AF} \rangle \langle \mathrm{AF} \rangle \cdots$

La structure $\{\langle \mathrm{NM} \rangle, \langle \mathrm{NF} \rangle\}^*$ `sont respectivement` $\{\langle \mathrm{AM} \rangle, \langle \mathrm{AF} \rangle\}^*$
ne peut être valide que si le nombre de noms masculins (resp. féminins)
est égal au nombre d'adjectifs masculins (resp. féminins).

Définissons donc le langage régulier suivant :

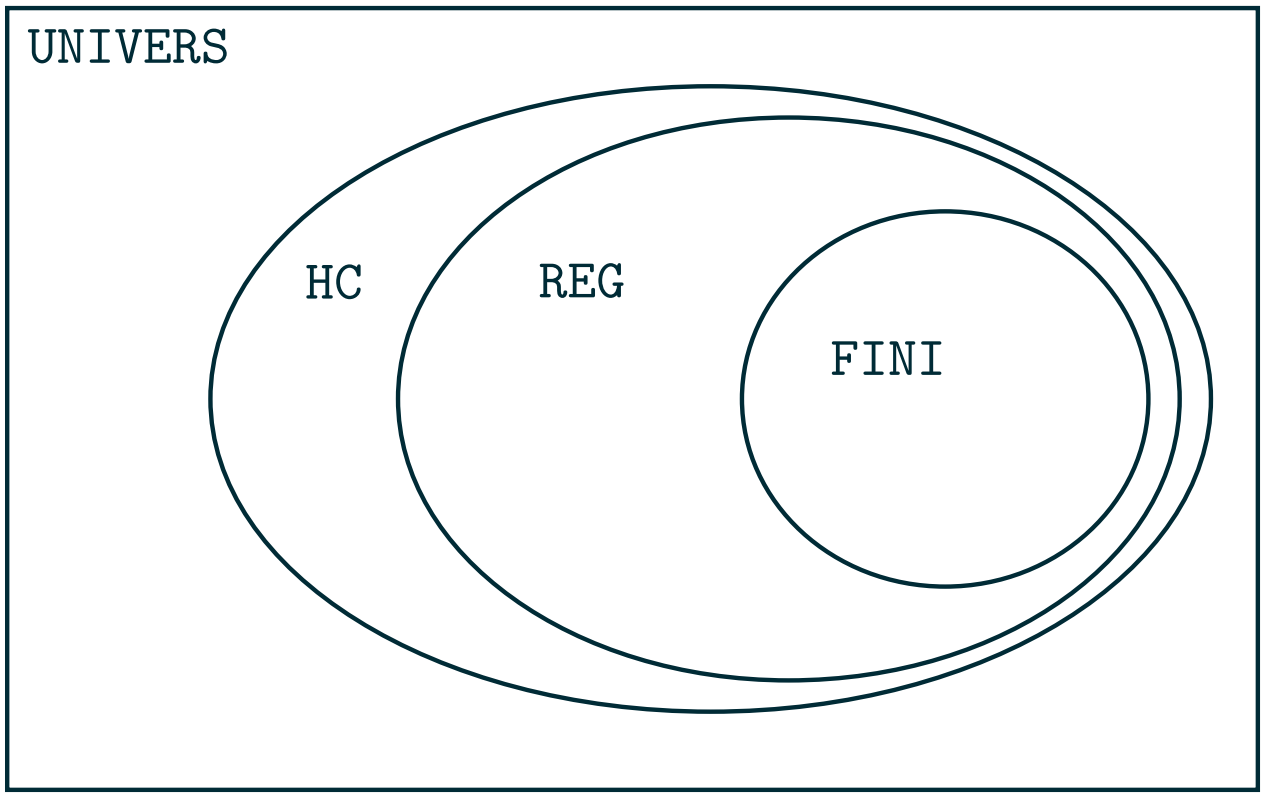
$$L = \{w \mid w = \langle \mathrm{NM} \rangle^n \langle \mathrm{NF} \rangle^m \texttt{ sont respectivement } \langle \mathrm{AM} \rangle^\ell \langle \mathrm{AF} \rangle^k, \\ n, m, \ell, k \geqslant 0\}$$

Si le français est hors-contexte, l'intersection du français avec L est également hors-contexte selon le théorème 4.105. Or, cette intersection est égale à :

$$L' = \{w \mid w = \langle \mathrm{NM} \rangle^n \langle \mathrm{NF} \rangle^m \texttt{ sont respectivement } \langle \mathrm{AM} \rangle^n \langle \mathrm{AF} \rangle^m, \\ n, m \geqslant 0\}$$

et on peut facilement démontrer avec le lemme du pompiste que ce langage n'est pas hors-contexte.

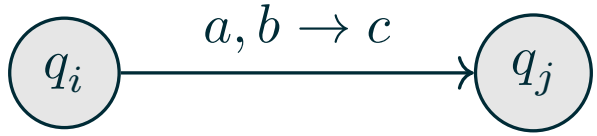
Hiérarchie



Automates à pile

Il existe aussi un type d'automate qui reconnaît les langages hors-contexte : les automates à pile (AP).

Un AP est essentiellement un AFN avec une pile pouvant contenir des lettres d'un alphabet Γ potentiellement différent de celui du langage. Chaque transition permet de dépiler et d'empiler une lettre. On a donc des transitions comme :



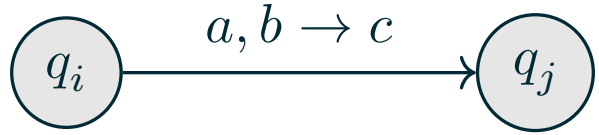
Dans cette transition, on lit le caractère $a \in \Sigma_\varepsilon$, on dépile $b \in \Gamma_\varepsilon$, et on empile $c \in \Gamma_\varepsilon$. Ici, $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$, voulant dire que ces symboles peuvent être le caractère vide ε.

Définition 4.106 (Automate à pile)

Un automate à pile est un tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, où Q, Σ, Γ et F sont des ensembles finis, et où

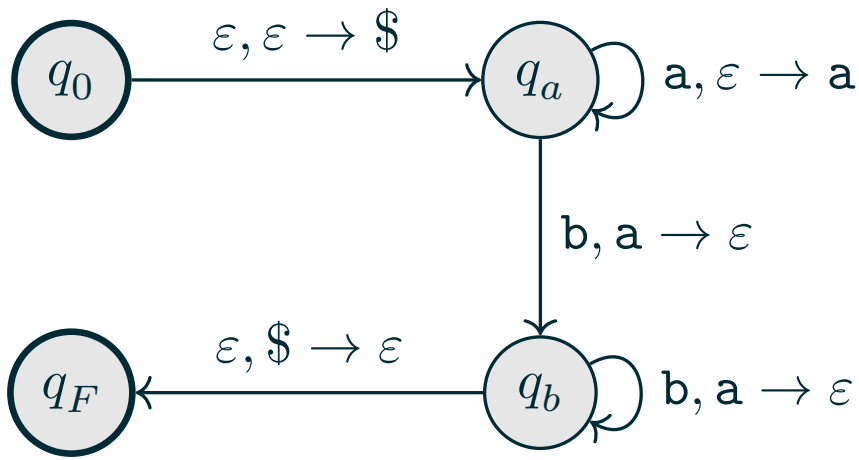
1. Q est l'ensemble des états
2. Σ est l'alphabet du langage
3. Γ est l'alphabet de la pile
4. $\delta : Q \times Q \to \mathcal{P}(\Sigma_\varepsilon \times \Gamma_\varepsilon \times \Gamma_\varepsilon)$ est la fonction de transition
5. $q_0 \in Q$ est l'état de départ
6. $F \subseteq Q$ est l'ensemble des états acceptants

$\delta(q_i, q_j)$ contient l'ensemble des transitions permises pour passer de q_i à q_j. Si $(a, b, c) \in \delta(q_i, q_j)$ alors la transition



est permise. Un mot est accepté par l'automate s'il existe une suite de transitions sur ce mot permettant d'arriver à un état acceptant.

L'automate suivant avec $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ et $\Gamma = \{\$, \mathtt{a}\}$ reconnaît $L = \{a^n b^n \mid n \geq 0\}$:



Théorème 4.107

Un langage L est hors-contexte si et seulement si il est reconnu par un automate à pile.

Langages contextuels

Les langages contextuels sont engendrés par une grammaire dont les règles permettent un *contexte* conditionnant leur application, par exemple

$$\alpha A \beta \to \alpha \gamma \beta$$

permet la transition $A \Rightarrow \gamma$ seulement si A est précédé de α et suivi de β.

Hiérarchie de Chomsky

Niveau	Classe	Règles	Exemples
Type-3	REG	$A \to a$, $A \to aB$	$\{\mathtt{a}^n \mid n \geqslant 0\}$
Type-2	HC	$A \to \gamma$	$\{\mathtt{a}^n\mathtt{b}^n \mid n \geqslant 0\}$
Type-1	Contextuels	$\alpha A \beta \to \alpha\gamma\beta$	$\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geqslant 0\}$
Type-0	Récursivement énumérables	$\alpha \to \gamma$	Peut simuler les MT