# Chapitre 14 : Langages formels

## Table des matières

# List of Algorithms

# 1 Langages réguliers

## 1.1 Motivation

### 1.1.1 Introduction

On a souvent besoin de mettre en place une analyse de texte, même dans le cadre d'applications qui ne relèvent pas uniquement du traitement de texte.

Par exemple :

- La recherche d'un mot dans un texte (*cf* chap 11) ;
- analyser un document structuré afin de traiter de manière appropriée son contenu (exemple : compiler un programme, récupérer des données sérialisées (*cf* chap 11) dans un format particulier (ex : données brutes en CSV, fichiers de configuration en JSON ou en XML)) ;
- Reconnaître un encodage et le déchiffrer (exemple : QR-code).

Quelle que soit l'application, on a besoin d'un formalisme pour décrire la structure du texte et d'algorithmes efficaces capables d'analyser cette structure et d'extraire les données associées.

### 1.1.2 Exemple

Étant donné un fichier binaire, déterminer s'il contient la représentation binaire d'un entier non signé multiple de 3.

• Remarque : on n'utilise pas les types d'entiers natifs de C ou OCaml car ils ont une taille fixée qui peut être dépassée par le fichier.

Idée : on lit les bits un à un en effectuant les opérations associées modulo 3, en remarquant que

$$\begin{cases} \langle x0 \rangle_2 = 2 \langle x \rangle_2 \\ \langle x1 \rangle_2 = 2 \langle x \rangle_2 + 1 \end{cases}$$

On utilise cette table :

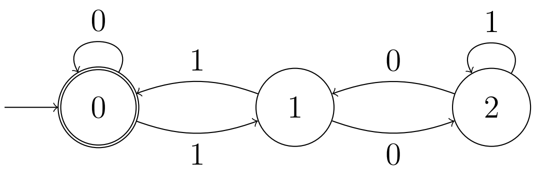
| | | | |
|---|---|---|---|
| $\langle x \rangle_2 \mod 3$ | 0 | 1 | 2 |
| $\langle x0 \rangle_2 \mod 3$ | 0 | 2 | 1 |
| $\langle x1 \rangle_2 \mod 3$ | 1 | 0 | 2 |

• Algorithme :

**Algorithm 1:** Exemple

$x \leftarrow 0$;
**for** chaque bit $b$ pris dans l'odre **do**
    **if** $b = 0$ **then**
        $x \leftarrow 2x \mod 3$;
    **else**
        $x \leftarrow 2x + 1 \mod 3$
**return** $x = 0$;

• Représentation graphique : $x$ ne peut prendre que trois valeurs différentes, appelées états, et on peut représenter les changements de valeur de $x$ dans un graphe orienté dont les sommets sont les états, et les arcs sont étiquetés par le bit qui produit le changement de valeur de l'état source vers l'état cible.




On distingue de plus la valeur initiale par une flèche, et la valeur finale atteinte par un double cercle.

Cette représentation correspond au formalisme des *automates*, que nous verrons en 2 (page 15).

## 1.2 Langages

### 1.2.1 Définition (*alphabet*)

Un *alphabet* est un ensemble fini non vide, dont les éléments sont appelés lettres ou symboles.

Notation usuelle : $\Sigma$.

### 1.2.2 Définition (*mot*)

Soit $\Sigma$ un alphabet.

Un *mot* sur $\Sigma$ est une suite finie de symboles $u = u_1 \cdots u_n$, potentiellement vide.

Si $n = 0$, on note $u = \varepsilon$.

On note $|u| = n$ la *longueur* du mot.

### 1.2.3 Définition (*concaténation*)

Soit $\Sigma$ un alphabet, et $u, v$ deux mots sur $\Sigma$.

On appelle *concaténation* de $u$ et $v$ le mot

$$uv = \begin{cases} v & \text{si } u = \varepsilon \\ u & \text{si } v = \varepsilon \\ u_1 \cdots u_n v_1 \cdots v_p & \text{si } \begin{cases} u = u_1 \cdots u_n \\ v = v_1 \cdots v_p \end{cases} \end{cases}$$

$\boxed{\text{Exo}}$

- $|uv| = |u| + |v|$
- La concaténation est une loi de composition interne associative et d'élément neutre $\varepsilon$ sur l'ensemble des mots sur $\Sigma$.

### 1.2.4 Définition (*préfixe, suffixe, facteur, sous-mot*)

Soit $\Sigma$ un alphabet, et $u, v$ deux mots sur $\Sigma$

- $v$ est un *préfixe* de $u$ ssi $\exists w$ mot tel que $u = vw$
- $v$ est un *suffixe* de $u$ ssi $\exists w$ mot tel que $u = wv$
- $v$ est un *facteur* de $u$ ssi $\exists x, y$ mots tels que $u = xvy$
- $v$ est un *sous-mot* de $u$ ssi $\exists i_1 < i_2 < \cdots < i_k$ tels que si $u = u_1 \cdots u_n$, alors $v = u_{i_1} \cdots u_{i_k}$

Exemple : si $u = abc$, $v = ac$ est un sous-mot de $u$, mais pas un facteur.

### 1.2.5 Langages

- Définition (*langage*) : un *langage* sur un alphabet $\Sigma$ est un ensemble de mots sur $\Sigma$.

- Exemples :
  - l'ensemble de tous les mots, noté $\Sigma^*$ (*cf* 1.3, page 6) ;
  - l'ensemble des écritures binaires de multiples des 3 ($\Sigma = \{0, 1\}$) ;
  - $\Sigma$ (si on voit les lettres comme des mots de longueur 1) ;
  - l'ensemble des code sources OCaml de programmes qui ne terminent pas.

- Problème : étant donné un langage $L$ sur un alphabet $\Sigma$, on veut disposer d'une représentation formelle de $L$ pour pouvoir étudier la question suivante : étant donné un mot $u$, a-t-on $u \in L$ ?

C'est une question importante car souvent, comme dans les exemples en 1.1.1 (page 3), il faut pouvoir vérifier la structure d'un élément avant d'en extraire des données.

Malheureusement, on ne peut pas toujours répondre algorithmiquement à cette question. (*cf* chap 16 et la notion de décidabilité et le problème de l'arêt), mais on peut y répondre pour une classe restreinte de langages.

## 1.3 Langages réguliers

### 1.3.1 Opérations sur les langages

Outre les opérations ensemblistes usuelles (intersection, union, complémentaire), on définit certaines opérations plus spécifiques aux langages.

• **Concaténation** : Soit $\Sigma$ un alphabet, et $L, L'$ deux langages sur $\Sigma$.
La concaténation de $L$ et $L'$ est le langage

$$LL' = \{uv \mid (u, v) \in L \times L'\}$$

Remarque : $L\varnothing = \varnothing L = \varnothing$.

• **Puissance** : Soit $\Sigma$ un alphabet, $L$ un langage sur $\Sigma$, et $n \in \mathbb{N}$.
La puissance $n$-ème de $L$ est le langage

$$L^n = \begin{cases} \{\varepsilon\} & \text{si } n = 0 \\ LL^{n-1} & \text{si } n > 0 \end{cases}$$

• **Étoile de Kleene** : Soit $\Sigma$ un alphabet, et $L$ un langage sur $\Sigma$.
L'*étoile de* Kleene de $L$ est le langage

$$L^* = \bigcup_{n \in \mathbb{N}} L^n$$

• Remarques :
  – $\Sigma^*$ est bien l'ensemble de tous les mots : tout mot $u = u_1 \cdots u_n$ est la caractérisation de ses lettres ($\forall i \in [\![1 \ ; \ n]\!]$, $u_i \in \Sigma$, donc $u = u_1 \cdots u_n \in \Sigma^n \subseteq \Sigma^*$).
  – On peut aussi définir la puissance $n$-ème d'un mot :

$$u^n = \begin{cases} \varepsilon & \text{si } n = 0 \\ uu^{n-1} & \text{si } n > 0 \end{cases}$$

  Attention : ne pas confondre $L^n$ et $\{u^n \mid u \in L\}$.
  – On note aussi

$$L^+ = \bigcup_{n \in \mathbb{N}^*} L^n$$

  $\boxed{\text{Exo}}$ $L^+ = L^* \iff \varepsilon \in L$.

### 1.3.2 Théorème (*Lemme d'*Arden) (H.P)

> Soit $\Sigma$ un alphabet, et $K, L$ deux langages sur $\Sigma$.

> (1) $K^*L$ est le minimum (pour l'ordre de l'inclusion) des solutions de l'équation
> $$X = KX \cup L$$
> d'inconnue un langage $X$.
>
> (2) Si $\varepsilon \notin K$, alors $K^*L$ est l'unique solution.

☐ Démonstration :

(1) On a :

$$\begin{aligned} K(K^*L) \cup L &= K\left(\bigcup_{n\in\mathbb{N}} K^n L\right) \cup L \\ &= K^+L \cup K^0 L \\ &= K^*L \end{aligned}$$

Donc $K^*L$ est une solution.

Puis soit $X$ une solution. Montrons que $K^*L \subset X$.

Soit $u \in K^*L$. Par définition,

$$\exists n \in \mathbb{N},\ \exists k_1, \dots, k_n \in K,\ \exists l \in L \mid u = k_1 \cdots k_n l$$

On montre par récurrence sur $n$ que $u \in X$ :

• $n = 0$ : $u \in L \subset KX \cup L = X$, donc $u \in X$.

• Hérédité : si $\forall k_1 \cdots k_n \in K,\ \forall l \in L,\ k_1 \cdots k_n l \in X$, considérons $u = k_1 \cdots k_{n+1} l$, avec $k_1 \cdots k_{n+1} \in K$ et $l \in L$.

$u = k_1 (\underbrace{k_2 \cdots k_{n+1} l}_{\in X \text{ par H.R}}) \in KX \subseteq KX \cup L = X$, donc $u \in X$.

Finalement, $K^*L \subset X$, et $K^*L$ est bien le minimum des solutions.

(2) On suppose $\varepsilon \notin K$. Soit $X$ une solution.

On sait par (1) que $K^*L \subseteq X$.

Il suffit de montrer que $X \subseteq K^*L$.

Soit $u \in X$ dont on note $n$ la longueur.

On montre par récurrence que

$$\forall k \in [\![0\ ;\ n]\!],\ X = \bigcup_{j=0}^{k} K^j L \cup K^{k+1} X$$

• $k = 0$ :

$$\bigcup_{j=0}^{0} K^j L \cup K^{0+1} X = L \cup KX = X$$

• Hérédité : Soit $k \in [\![0\ ;\ n-1]\!] \mid X = \bigcup_{j=0}^{k} K^j L \cup K^{k+1} X$

$$\begin{aligned} X &= KX \cup L \\ &= K\left(\bigcup_{j=0}^{k} K^j L \cup K^{k+1}L\right) \cup L \\ &= \bigcup_{j=0}^{k} K^{j+1} L \cup K^{k+1} L \cup L \\ &= \bigcup_{j=0}^{k+1} K^J L \cup K^{k+2} L \end{aligned}$$

En particulier,

$$X = \bigcup_{j=0}^{n} K^J L \cup K^{n+1} X$$

Or $\forall v \in k^{n+1}X,\ |v| \geqslant n+1$
En effet, $\exists k_1 \cdots k_{n+1} \in K,\ \exists x \in X \mid v = k_1 \cdots k_{n+1} x$.
Or $\forall i \in [\![1\ ;\ n+1]\!], k_i \neq \varepsilon$, donc $|k_i| \geqslant 1$, donc

$$|v| = \sum_{i=1}^{n+1} |k_i| + |x| \geqslant n+1+|x| \geqslant n+1$$

Donc, comme $|u| = n < n+1$,

$$\bigcup_{j=0}^{n} K^j L \subseteq \cup_{j\in\mathbb{N}} K^J L = K^* L$$

Donc $u \in K^*L$, et $X \subseteq K^*L$ ■

• Contre-exemple au point (2) si $\varepsilon \in K$ : on prend $K = \{\varepsilon\}$, et $L = \{a\}$. $K^*L = \{a\} = L$, et tout $X \mid a \in X$ est solution ($KX \cup L = X \cup \{a\}$). Par exemple : $\{a\} = K^*L$ ou $\{a, aa\}, \{a, aa, aaa\}$.

### 1.3.3 Langage régulier

La classe des *langages réguliers*, aussi appelés *langages rationnels*, est la famille des langages que l'on peut construire à partir de langages de base ($\varnothing$, $\{\varepsilon\}$, $\{a\}\ \forall a \in \Sigma$) et des opérations dites *régulières* : union, concaténation et étoile de Kleene. L'ensemble $\mathsf{Reg}(\Sigma)$ des langages réguliers sur l'alphabet $\Sigma$ est définit inductivement par :

$$\frac{}{\varnothing \in \mathsf{Reg}(\Sigma)} \qquad \frac{a \in \Sigma}{\{a\} \in \mathsf{Reg}(\Sigma)} \qquad \frac{L \in \mathsf{Reg}(\Sigma)}{L^* \in \mathsf{Reg}(\Sigma)}$$

$$\frac{L \in \mathsf{Reg}(\Sigma)\ L' \in \mathsf{Reg}(\Sigma)}{L \cup L' \in \mathsf{Reg}(\Sigma)} \qquad \frac{L \in \mathsf{Reg}(\Sigma)\ L' \in \mathsf{Reg}(\Sigma)}{LL' \in \mathsf{Reg}(\Sigma)}$$

• Remarque : $\{\varepsilon\} \in \mathsf{Reg}(\Sigma)$ car $\{\varepsilon\} = \varnothing^0 = \bigcup_{n\in\mathbb{N}} \varnothing^n = \varnothing^*$

$\Sigma \in \mathsf{Reg}(\Sigma)$ car, en notant $\Sigma = \{a_1 \cdots a_n\}$, on a :

$$\Sigma = \bigcup_{k=1}^{n} \{a_k\}$$

Donc une récurrence sur $n = |\Sigma|$ conclut.

De même, tout langage fini est régulier ($\boxed{\text{Exo}}$).

Il manque encore un formalisme pour décrire les langages réguliers.

### 1.3.4 Expressions régulières

Soit $\Sigma$ un alphabet, et $S = \{(,), |, *, \varnothing, \varepsilon\}$ un ensemble de symboles supposé disjoint de $\Sigma$.

L'ensemble $\mathsf{Regexp}(\Sigma)$ des *expressions régulières* sur $\Sigma$, aussi appelées *expressions rationnelles*, est l'ensemble des mots sur $\Sigma \cup S$ défini inductivement par

$$\frac{}{\varnothing \in \mathsf{Regexp}(\Sigma)} \qquad \frac{}{\varepsilon \in \mathsf{Regexp}(\Sigma)} \qquad \frac{a \in \Sigma}{a \in \mathsf{Regexp}(\Sigma)}$$

$$\frac{e \in \mathsf{Regexp}(\Sigma) \quad f \in \mathsf{Regexp}(\Sigma)}{(e|f) \in \mathsf{Regexp}(\Sigma)} \qquad \frac{e \in \mathsf{Regexp}(\Sigma) \quad f \in \mathsf{Regexp}(\Sigma)}{(ef) \in \mathsf{Regexp}(\Sigma)}$$

$$\frac{e \in \mathsf{Regexp}(\Sigma)}{(e^*) \in \mathsf{Regexp}(\Sigma)}$$

Remarques :

- $|$ est parfois noté $+$.
- On se passe de certaines parenthèses avec les règles de priorité :

$$* > \text{concaténation} > |$$

Par exemple, $((a\,(b^*))\,|b)$ s'écrit $ab^*|b$.

### 1.3.5 Définition (*langage dénoté par une expression régulière*)

Soit $\Sigma$ un alphabet.

Le *langage dénoté par une expression régulière* $e \in \mathsf{Regexp}(\Sigma)$ est le langage $\mathcal{L}(e)$ défini inductivement par :

$$\mathcal{L}(\varnothing) = \varnothing \qquad \mathcal{L}(\varepsilon) = \{\varepsilon\} \qquad \forall a \in \Sigma,\ \mathcal{L}(a) = \{a\}$$

$$\mathcal{L}(e|f) = \mathcal{L}(e) \cup \mathcal{L}(f) \qquad \mathcal{L}(ef) = \mathcal{L}(e)\mathcal{L}(f) \qquad \mathcal{L}(e^*) = \mathcal{L}(e)^*$$

Exemple :

$L_1 = \{\varepsilon\} = \mathcal{L}(\varnothing^*)$

$L_2 = \Sigma = \mathcal{L}(a_1|a_2|\cdots|a_n)$ en notant $\Sigma = \{a_k \mid k \in [\![1\ ;\ n]\!]\}$

$$\begin{aligned} L_3 &= \{u \in \Sigma^* \mid |u| \equiv 1\ [2]\} \\ &= \bigcup_{n\in\mathbb{N}} \{u \in \Sigma^* \mid |u| = 2n+1\} \\ &= \left(\bigcup_{n\in\mathbb{N}} \{u \in \Sigma^* \mid |u| = 2n\}\right)\Sigma \\ &= \mathcal{L}((\Sigma\Sigma)^*\Sigma) \end{aligned}$$

$L_4 = \{u \in \Sigma^* \mid a$ préfixe de $u$ et $b$ suffixe de $u\} = \mathcal{L}(a\Sigma^*b)$
$L_5 = \{u \in \Sigma^* \mid ab$ est facteur de $u\} = \mathcal{L}(\Sigma^*ab\Sigma^*)$

### 1.3.6 Proposition

> Soit $\Sigma$ un alphabet, et $L$ un langage sur $\Sigma$.
> Alors :
> $$L \text{ est régulier} \quad \Leftrightarrow \quad \exists e \in \mathsf{Regexp}(\Sigma) \mid L = \mathcal{L}(e)$$

☐ Démonstration :

$\boxed{\Leftarrow}$ $\boxed{\text{Exo}}$ (par induction sur $e$)

$\boxed{\Rightarrow}$ par induction sur une dérivation de $L \in \mathsf{Reg}(\Sigma)$, en faisant une disjonction de cas selon la dernière règle utilisée.

- $\dfrac{}{\varnothing \in \mathsf{Reg}(\Sigma)}$ : alors $L = \varnothing = \mathcal{L}(\varnothing)$
- $\dfrac{a \in \Sigma}{\{a\} \in \mathsf{Reg}(\Sigma)}$ : alors $L = \{a\} = \mathcal{L}(a)$
- $\dfrac{L_1 \in \mathsf{Reg}(\Sigma)\ \ L_2 \in \mathsf{Reg}(\Sigma)}{L_1 \cup L_2 \in \mathsf{Reg}(\Sigma)}$ : alors $L = L_1 \cup L_2$, et par H.I :

$$\exists e_1, e_2 \in \mathsf{Regexp}(\Sigma) \left| \begin{array}{l} L_1 = \mathcal{L}(e_1) \\ L_2 = \mathcal{L}(e_2) \end{array}\right.$$

D'où $L = L_1 \cup L_2 = \mathcal{L}(e_1) \cup \mathcal{L}(e_2)$

- $\dfrac{L_1 \in \mathsf{Reg}(\Sigma)\ \ L_2 \in \mathsf{Reg}(\Sigma)}{L_1L_2 \in \mathsf{Reg}(\Sigma)}$ : alors $L = L_1L_2$, et par H.I :

$$\exists e_1, e_2 \in \mathsf{Regexp}(\Sigma) \left| \begin{array}{l} L_1 = \mathcal{L}(e_1) \\ L_2 = \mathcal{L}(e_2) \end{array}\right.$$

D'où $L = L_1L_2 = \mathcal{L}(e_1)\mathcal{L}(e_2)$

- $\dfrac{L_0 \in \mathsf{Reg}(\Sigma)}{L_0^* \in \mathsf{Reg}(\Sigma)}$ : alors $L = L_0^*$, et par H.I :

$$\exists e \in \mathsf{Regexp}(\Sigma) \mid L_0 = \mathcal{L}(e)$$

D'où $L = L_0^* = \mathcal{L}(e)$ ■

Exemple : $L = \{ab, ac\}$

$$\dfrac{\dfrac{\dfrac{a \in \Sigma}{\{a\} \in \mathsf{Reg}(\Sigma)} \quad \dfrac{b \in \Sigma}{\{b\} \in \mathsf{Reg}(\Sigma)}}{\{ab\} \in \mathsf{Reg}(\Sigma)} \quad \dfrac{\dfrac{a \in \Sigma}{\{a\} \in \mathsf{Reg}(\Sigma)} \quad \dfrac{c \in \Sigma}{\{c\} \in \mathsf{Reg}(\Sigma)}}{\{ac\} \in \mathsf{Reg}(\Sigma)}}{\{ab, ac\} \in \mathsf{Reg}(\Sigma)}$$

$$ab|ac$$

$$\dfrac{\dfrac{a \in \mathsf{Reg}(\Sigma)}{\{a\} \in \mathsf{Reg}(\Sigma)} \quad \dfrac{\dfrac{b \in \Sigma}{\{b\} \in \mathsf{Reg}(\Sigma)} \quad \dfrac{c \in \Sigma}{\{c\} \in \mathsf{Reg}(\Sigma)}}{\{b, c\} \in \mathsf{Reg}(\Sigma)}}{\{ab, ac\} \in \mathsf{Reg}(\Sigma)}$$

$$a(b|c)$$

### 1.3.7 Définition (*équivalence d'expressions régulières*)

Soit $\Sigma$ un alphabet, et $e, f \in \mathsf{Regexp}(\Sigma)$.
Alors $e, f$ sont dits équivalents, noté $e \equiv f$, si et seulement si $\mathcal{L}(e) = \mathcal{L}(f)$.

### 1.3.8 Proposition

> Soit $\Sigma$ un alphabet, et $e \in \mathsf{Regexp}(\Sigma)$.
> (1) Si $e$ ne contient pas $\varnothing$, alors $\mathcal{L}(e) \neq \varnothing$
> (2) La réciproque est fausse
> (3) Si $\mathcal{L}(e) \neq \varnothing$, alors $\exists f \in \mathsf{Regexp}(\Sigma) \mid e \equiv f$ et $f$ ne contient pas $\varnothing$.

☐ Démonstration :

(1) Exo par induction sur $e$

(2) $a|\varnothing$ avec $a \in \Sigma$

(3) Par induction sur $e$ :

- $e = \varnothing$ : impossible
- $e = \Sigma$ ou $e = a$ avec $a \in \Sigma$, $f = e$ convient
- $e = e_1|e_2$ : $\mathcal{L}(e_1) = \varnothing = \mathcal{L}(e_2)$ est impossible car $\mathcal{L}(e) = \mathcal{L}(e_1) \cup \mathcal{L}(e_2)$
  - Si $\mathcal{L}(e_1) = \varnothing$ et $\mathcal{L}(e_2) \neq \varnothing$ (l'autre cas est symétrique), $\mathcal{L}(e) = \mathcal{L}(e_2)$, et par H.I, $\exists f_2 \in \mathsf{Regexp}(\Sigma) \mid f_2$ ne contient pas $\varnothing$ et $f_2 \equiv e_2$.
    Alors $e \equiv f_2$.
  - Si $\mathcal{L}(e_1) \neq \varnothing$ et $\mathcal{L}(e_2) \neq \varnothing$, par H.I, $\exists f_1, f_2 \in \mathsf{Regexp}(\Sigma) \mid f_1, f_2$ ne

contiennent pas $\varnothing$, et

$$\begin{cases} f_1 \equiv e_1 \\ f_2 \equiv e_2 \end{cases}$$

Alors $\mathcal{L}(e) = \mathcal{L}(e_1) \cup \mathcal{L}(e_2) = \mathcal{L}(f_1) \cup \mathcal{L}(f_2) = \mathcal{L}(f_1|f_2)$ et $e \equiv f_1|f_2$ qui ne contient pas $\varnothing$.

- $e = e_1 e_2$ : $\mathcal{L}(e_1) = \varnothing$ ou $\mathcal{L}(e_2) = \varnothing$ est impossible car $\mathcal{L}(e) = \mathcal{L}(e_1)\mathcal{L}(e_2)$.

De même, on applique l'H.I à $e_1$ et $e_2$ pour obtenir deux expressions régulières dont on prend la concaténation.

- $e = l_0^*$ :
  - Si $\mathcal{L}(e_0) = \varnothing$, alors $\mathcal{L}(e) = \{\varepsilon\} = \mathcal{L}(\varepsilon)$
  - Sinon, on applique l'H.I à $e_0$ pour obtenir une expression régulière dont on prend l'étoile de Kleene

■

### 1.3.9 Proposition

Soit $\Sigma$ un alphabet, et $e \in \mathsf{Regexp}(\Sigma)$.
On définit le terme constant de $e$ inductivement par

$$c(\varnothing) = 0 \qquad c(\varepsilon) = 1 \qquad c(a) = 0$$
$$c(ef) = \min(c(e), c(f)) \qquad c(e|f) = \max(c(e), c(f)) \qquad c(e^*) = 1$$

Alors :

$$\varepsilon \in \mathcal{L}(e) \iff c(e) = 1$$

□ Démonstration :

$\boxed{\text{Exo}}$ par induction sur $e$. ■

## 1.4 Expressions régulières étendues

### 1.4.1 Définition (*expressions régulières étendues*)

Soit $\Sigma$ un alphabet.
On étend la définition des expressions régulières en ajoutant deux symboles $\cap$ et $\setminus$ à $S$ et en ajoutant les règles d'inférence suivantes :

$$\frac{e \in \mathsf{Regexp}(\Sigma) \quad f \in \mathsf{Regexp}(\Sigma)}{e \cap f \in \mathsf{Regexp}(\Sigma)} \qquad \frac{e \in \mathsf{Regexp}(\Sigma) \quad f \in \mathsf{Regexp}(\Sigma)}{(e \setminus f) \in \mathsf{Regexp}(\Sigma)}$$

Les langages dénotés sont

$$\mathcal{L}(e \cap f) = \mathcal{L}(e) \cap \mathcal{L}(f)$$
$$\mathcal{L}(e \setminus f) = \mathcal{L}(e) \setminus \mathcal{L}(f)$$

Nous verrons que les expressions régulières étendues dénotent les mêmes langages que les expressions régulières.

### 1.4.2 Application

Les expressions régulières (étendues) sont liées aux expressions régulières de la norme POSIX, utilisées par exemple dans la commande `grep`.
Exemple :

```
find | grep -E '.*/d[ms][1/]*\.(pdf|tex)$' | grep 'corrigé'
```

trouve les fichiers dont le nom contient `dm` ou `ds`, d'extension `.pdf` ou `.tex`, et dont le nom contient `corrigé`.

### 1.4.3 Définition (*langages des préfixes / suffixes / facteurs*)

Soit $L$ un langage sur un alphabet $\Sigma$.
On défini les langages des préfixes / suffixes de longueur 1 et des facteurs de longueur 2 des mots de $L$ par

$$\begin{aligned}P_1(L) &= \{a \in \Sigma \mid \exists u \in \Sigma^* \mid au \in L\}\\ S_1(L) &= \{a \in \Sigma \mid \exists u \in \Sigma^* \mid ua \in L\}\\ F_2(L) &= \{u \in \Sigma^2 \mid \exists x, y \in \Sigma^* \mid xuy \in L\}\end{aligned}$$

### 1.4.4 Langages locaux

• $\boxed{\text{Exo}}$ pour tout langage $L$ sur un alphabet $\Sigma$,

$$L \setminus \{\varepsilon\} \subseteq (P_1(L)\Sigma^* \cap \Sigma^* S_1(L)) \setminus \left(\Sigma^* \left(\Sigma^2 \setminus F_2(L)\right) \Sigma^*\right)$$

mais cette inclusion peut être stricte.

• Définition : Soit $\Sigma$ un alphabet, et $L$ un langage sur $\Sigma$.
$L$ est *local* si et seulement si

$$L \setminus \{\varepsilon\} = (P_1(L)\Sigma^* \cap \Sigma^* S_1(L)) \setminus \left(\Sigma^* \left(\Sigma^2 \setminus F_2(L)\right) \Sigma^*\right)$$

• Exemples :
  – $(ab)^*$ et $(ab)^+$ sont locaux
  – $P_1 = \{a\}$, $S_1 = \{b\}$, et $F_2 = \{a, b\}^2 \setminus \{ab, ba\}$ définissent les langages locaux $\varepsilon$ et $\varnothing$

• Proposition :

> Tout langage local est dénoté par une expression régulière étendue

☐ Démonstration :

En utilisant la définition et le fait qu'un langage fini est régulier $\boxed{\text{Exo}}$ ■

• $\boxed{\text{Exo}}$ : Donner un algorithme permettant de déterminer $P_1(L)$, $S_1(L)$, $F_2(L)$ pour tout langage régulier.

Indication : induction sur une expression régulière dénotant $L$.

### 1.4.5 Langages linéaires

• Définition (*expression régulière linéaire*) : une expression régulière est dite *linéaire* si et seulement si chaque lettre qui la compose n'y apparaît qu'une seule fois.

• Exemple : $(ab)^*|cd^*e$ mais pas $(ab^*)|ca$

• Définition (*langage linéaire*) : un langage est *linéaire* si et seulement si il est dénoté par une expression régulière linéaire.

• Proposition :

> Tout langage linéaire est local.

☐ Démonstration :

– Lemme 1 :

> *Lemme 1* : Soient $L_1, L_2$ deux langages locaux sur des alphabets disjoints. Alors $L_1 \cup L_2$ est local

$\boxed{\text{Exo}}$

– Lemme 2 :

> *Lemme 2* : Soient $L_1, L_2$ deux langages locaux sur des alphabets disjoints. Alors $L_1L_2$ est local.

$\boxed{\text{Exo}}$

Indication : distinguer quatre cas selon que $\varepsilon \in L_1$ ou $\varepsilon \in L_2$.

– Par induction sur une expression régulière linéaire dénotant le langage

∗ $\varnothing$ ou $\varepsilon$ : $P_1 = S_1 = F_2 = \varnothing$ conviennent

∗ $a \in \Sigma$ : $P_1 = S_1 = \{a\}$ et $F_2 = \varnothing$ convient

∗ $e|f$ : par H.I, $\mathcal{L}(e)$ et $\mathcal{L}(f)$ sont locaux.

De plus, come $e|f$ est linéaire, $e$ et $f$ sont exprimés sur des alphabets disjoints.

Donc le lemme 1 conclut.

∗ $ef$ : le même raisonnement avec le lemme 2.

$*$ $e^*$ : par H.I, $\mathcal{L}(e)$ est local, donc

$$\mathcal{L}(e) \setminus \{\varepsilon\} = (P_1\Sigma^* \cap \Sigma^* S_1) \setminus \left(\Sigma^* \left(\Sigma^2 \setminus F_2\right) \Sigma^*\right)$$

$P_1(e^*) = P_1(e)$
$S_1(e^*) = S_1(e)$
$F_2(e^*) = F_2(e) \cup S_1(e)P_1(e)$

■

• Contre-exemples :

– lemmes 1 et 2 : $L_1 = ab$ et $L_2 = a^*$

Si $L_1 \cup L_2$ était local, comme $P_1 = \{a\}$, $S_1 = \{a, b\}$, $F_2 = \{ab, aa\}$, on aurait $aab \in L_1 \cup L_2$ : absurde

Si $L_1L_2$ était local, comme $P_1 = \{a\}$, $S_1 = \{a, b\}$, $\{ab, ba, aa\}$, on aurait $aab \in L_1L_2$ : absurde

– $a(ba)^*$ est local mais non linéaire.

• Linéarisation : on peut transformer une expression régulière $e$ en expression régulière linéaire $\tilde{e}$ en numérotant ses lettres (l'alphabet devient $\Sigma \times [\![1\ ;\ n]\!]$)

Exemple : $aa(a|ab)^*b$ devient $a_1a_2(a_3|a_4b_1)^*b_2$

C'est une étape importante pour faire le lien entre automates et langages réguliers.

$\boxed{\text{Exo}}$ $\forall u,\ u \in \mathcal{L}(e) \iff \exists$ numérotation des lettres de $u$ donnant $\tilde{u} \in \mathcal{L}(\tilde{e})$

# 2 Automates finis

## 2.1 Automates finis déterministes

### 2.1.1 Introduction

La représentation visuelle de l'algorithme n°1 (page 4) est la representation graphique d'un automate fini déterministe. On peut voir un automate comme une machine à états, qui change d'état en fonction des caractères lus sur son entrée. L'entrée est considérée comme valide si l'état final de la machine est approprié.

Il existe plusieurs notions d'automates, et nous montrerons que toutes celles au programme sont équivalentes.

### 2.1.2 Définition (*automate fini déterministe (AFD)*)

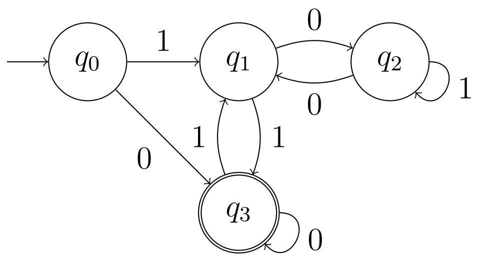
Un *automate fini déterministe* est un quintuplet $(\Sigma, Q, q_0, F, \delta)$, où :

- $\Sigma$ est un alphabet ;
- $Q$ est un ensemble fini non vide, appelé *ensemble d'états* ;
- $q_0 \in Q$ est l'état initial ;

- $F \in \mathcal{P}(Q)$ est l'ensemble des états acceptants / finaux / final / terminaux ;
- $\delta : Q \times \Sigma \longrightarrow Q$ est une fonction partielle (*i.e* définie sur une partie de $Q \times \Sigma$) appelée *fonction de transition.*

Représentation graphique :
- Tout état $q \in Q$ est représenté par un cercle ;
- L'état initial prend une flèche ;
- Les états acceptants sont cerclés deux fois ;
- $\forall q, q' \in Q,\ \forall a \in \Sigma \mid \delta(q, a) = q'$, on dessine un arc de $q$ à $q'$ étiqueté par $a$.

Exemple :




représente

$$(\{0, 1\}, \{q_0, \dots, q_3\}, q_0, \{q_3\}, \delta)$$

où $\delta$ est défini par la table

| | 0 | 1 |
|---|---|---|
| $q_0$ | $q_3$ | $q_1$ |
| $q_1$ | $q_2$ | $q_3$ |
| $q_2$ | $q_1$ | $q_2$ |
| $q_3$ | $q_3$ | $q_1$ |

### 2.1.3 Modèle de calcul associé

- Définition (*transition généralisées*) : Soit $M = (\Sigma, Q, q_0, F, \delta)$ un AFD.

On généralise la fonction de transition aux mots par :

$$\begin{cases} \forall q \in Q,\ \delta(q, \varepsilon) = q \\ \forall (q, u, a) \in Q \times \Sigma^* \times \Sigma \mid (q, a) \in \mathrm{dom}(\delta),\ \delta(q, au) = \delta(\delta(q, a), u) \end{cases}$$

$\boxed{\text{Exo}}$ Mq cette définition est équivalente à

$$\forall (q, u, a) \in Q \times \Sigma^* \times \Sigma,\ \delta(q, ua) = \delta(\delta(q, u), a)$$

ou alors

$$\forall q, q' \in Q,\ \forall u \in \Sigma^*,\ \delta(q, u) = q'$$

si et seulement si il existe un chemin dans le graphe de l'automate de $q$ à $q'$ tel que la concaténation des étiquettes sur ce chemin vaut $u$.

• Définition (*langage reconnu / reconnaissable*) : Soit $M = (\Sigma, Q, q_0, F, \delta)$ un AFD.
– $u \in \Sigma^*$ est *accepté* par $M$ ssi $\delta(q_0, u) \in F$
– Le langage *reconnu* par $M$ est

$$\mathcal{L}(M) = \{u \in \Sigma^* \mid u \text{ accepté par } M\}$$

– Un langage $L$ sur un alphabet $\Sigma$ est *reconnaissable* ssi il existe un AFD $M$ d'alphabet $\Sigma$ tel que $\mathcal{L}(M) = L$.
On note alors $\mathsf{Rec}(\Sigma)$ l'ensemble des langages reconnaissables sur $\Sigma$.

• Exemples :
– $\{u \in \{0,1\}^* \mid \langle u \rangle_2 \equiv 0\ [3]\} \in \mathsf{Rec}(\{0,1\})$ par 1.1.2 (page 3).
– On reprend l'AFD vu en 2.1.2 (page 15). Pour déterminer $\mathcal{L}(M)$, on peut utiliser le lemme d'Arden.
$\forall q \in Q$, on définit

$$L_q = \{u \in \Sigma^* \mid \delta(q, u) \in F\}$$

On cherche $\mathcal{L}(M) = L_{q_0}$ sachant :

$$\begin{cases} L_{q_0} = 0L_{q_3} \cup 1L_{q_1} \\ L_{q_1} = 0L_{q_2} \cup 1L_{q_3} \\ L_{q_2} = 0L_{q_1} \cup 1L_{q_2} \\ L_{q_3} = 0L_{q_3} \cup 1L_{q_1} \cup \{\varepsilon\} \end{cases}$$

$\boxed{\text{Exo}}$ Mq $\mathcal{L}(M) = (1(01^*0)^*1|0)^*$

### 2.1.4 Automate complet

• Définition (*Automate complet*) : un AFD est dit *complet* ssi sa fonction de transition est totale.

• Exemple : les automates en 1.1.2 (page 3) et 2.1.2 (page 15).

• Complétion : étant donné un AFD $M = (\Sigma, Q, q_0, F, \delta)$ incomplet, on peut construire $M'$ complet reconnaissant $\mathcal{L}(M)$ en ajoutant un état dit *poubelle* / *puits*, vers lequel vont toutes les transitions non définies de $\delta$.

$$M' = (\Sigma, Q \cup \{\Pi\}, q_0, F, \delta')$$

où $\Pi \notin Q$, et

$$\forall q \in Q \cup \{\Pi\},\ \forall a \in \Sigma,\ \delta'(q, a) = \begin{cases} \delta(q, a) & \text{si } (q, a) \in \mathrm{dom}(\delta) \\ \Pi & \text{sinon} \end{cases}$$

• Proposition :

> $M'$ est complet, et $\mathcal{L}(M') = \mathcal{L}(M)$

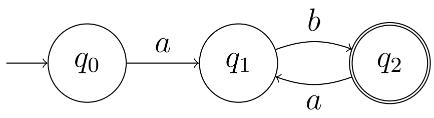
☐ Démonstration :

$M'$ est complet par définition.

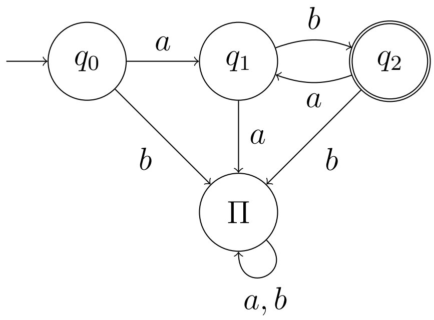
Si $u \in \mathcal{L}(M)$, on remarque que toutes les transitions suivies dans $M$ lors de la lecture de $u$ sont définies, et ce sont les mêmes dans $M'$, donc $u \in \mathcal{L}(M')$ (récurrence finie, en notant $u = u_1 \cdots u_n$, et $\forall i \in [\![1\ ;\ n]\!]\,,\ q_i = \delta(q_0, u_1 \cdots, u_i)$ $\boxed{\text{Exo}}$).

Si $u \in \mathcal{L}(M')$, on remarque qu'il n'existe pas de transition étiquetée par un préfixe de $u$ menant à l'état $\Pi$ (sinon $\delta'(q_0, u) = \Pi \notin F$) donc toutes ces transitions sont aussi définies dans $M$ et $u \in \mathcal{L}(M)$. ■

• Exemple :




Devient :




### 2.1.5 Automate standard

• Définition : Un AFD est dit *standard* ssi aucune transistion ne mène à l'état initial.

• Exemple : l'automate en 2.1.2 (page 15), mais pas celui en 1.1.2 (page 3).

• Standardisation : si $M = (\Sigma, Q, q_0, F, \delta)$ est un AFD non standard, on construit un AFD $M'$ standard reconnaissant $\mathcal{L}(M)$ en ajoutant un nouvel état initial qui ne sert que pour la première transition.

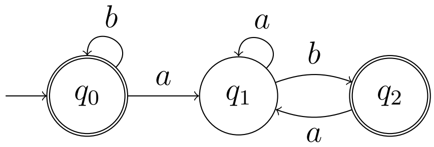
On définit $M' = (\Sigma, Q \cup \{\iota\}, \iota, F', \delta')$, où $\iota \notin Q$,

$$F' = \begin{cases} F & \text{si } q_0 \notin F \\ F \cup \{\iota\} & \text{sinon} \end{cases}$$

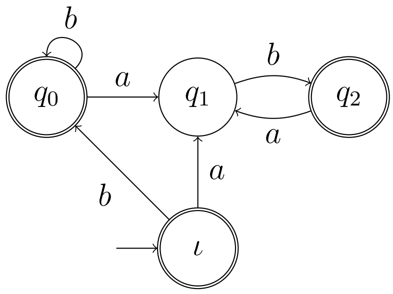
$$\forall (q, a) \in (Q \cup \{\iota\}) \times \Sigma,\ \delta'(q, a) = \begin{cases} \delta(q, a) & \text{si } (q, a) \in \mathrm{dom}(\delta) \\ \delta(q_0, a) & \text{si } \iota = q \text{ et } (q_0, a) \in \mathrm{dom}(\delta) \\ \text{non définie} & \text{sinon} \end{cases}$$

$\boxed{\text{Exo}}$ $M'$ est standard, et $\mathcal{L}(M') = \mathcal{L}(M)$.

• Exemple :




Devient :




### 2.1.6 Automate émondé

• Idée : certains états peuvent être inutiles du point de vue du modèle de calcul.

Par exemple, l'état puits d'un complété n'apporte rien au langage reconnu. De même un état qui ne serait pas accessible depuis l'état initial est inutile.

• Définition (*états accessibles / co-accessibles, utiles*) : Soit $M = (\Sigma, Q, q_0, F, \delta)$ un AFD, et $q \in Q$.

- $q$ est dit *accessible* ssi $\exists u \in \Sigma^* \mid \delta(q_0, u) = q$
- $q$ est dit *co-accessible* ssi $\exists u \in \Sigma^* \mid \delta(q, u) \in F$
- $q$ est dit *utile* ssi $q$ est accessible et co-accessible.

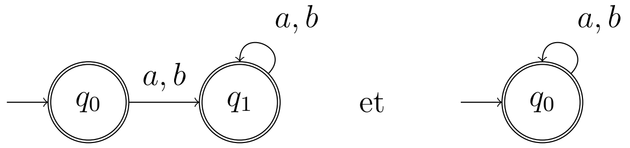
• Remarque : il est possible de déterminer les états accessibles grâce à un parcours depuis l'état initial, les états co-accessibles grâce à un parcours du graphe transposé du graphe de l'automate à partir des états acceptants, et les états utiles grâce à deux parcours.

• Définition (*automate émondé*) : Un automate est dit *émondé* ssi tous ses états sont utiles.

• $\boxed{\text{Exo}}$ Supprimer les états inutiles donne un AFD émondé reconnaissant le même langage.

• Remarque : Disposer d'un automate émondé reconnaissant un langage donné n'assure pas d'avoir l'information minimale permettant de caractériser ce langage.

Exemple :



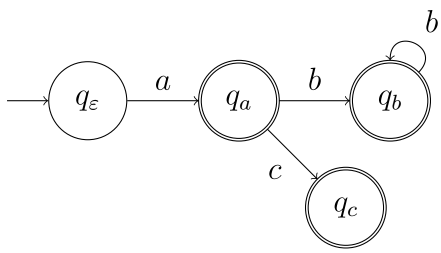

sont deux automates émondés reconnaissant le même langage ($\Sigma^*$, avec $\Sigma = \{a, b\}$)

• (H.P) Un automate reconnaissant un langage donné et ayant un nombre minimal d'états est appelé *automate minimal* du langage et on peut montrer que l'automate minimal d'un langage donné est émondé et unique (à isomorphisme près).

### 2.1.7 Automate local

• Remarque : Un langage local est caractérisé par la connaissance des lettres qui peuvent débuter / terminer un mot du langage et des lettres qui peuvent succéder à une lettre donnée dans un mot de langage. En particulier, la connaissance de la dernière lettre lue permet de connaître les lettres qui peuvent suivre, ce qui fait des langages locaux de bon candidats pour l'appartenance à $\mathsf{Rec}(\Sigma)$.

• Exemple : $a(b^*|c)$, $P_1 = \{a\}$, $S_1 = \{a, b, c\}$, $F_2 = \{ab, ac, bb\}$
Idée : on associe un état à chaque lettre lue.




• Définition (*automate local*) : Soit $M = (\Sigma, Q, q_0, F, \delta)$ un AFD.
$M'$ est dit *local* ssi

$$\forall a \in \Sigma,\ \exists q \in Q \mid \forall q' \in Q,\ (q', a) \in \mathrm{dom}(\delta) \Rightarrow \delta(q', a) = q$$

• Proposition :

> Soit $\Sigma$ un alphabet, et $L$ un langage sur $\Sigma$.
>
> $$L \text{ est local } \Leftrightarrow \exists M = (\Sigma, Q, q_0, F, \delta) \text{ un AFD local} \mid \mathcal{L}(M) = L$$

□ Démonstration :

$\boxed{\Rightarrow}$ On sait que

$$L \setminus \{\varepsilon\} = (P_1(L)\Sigma^* \cap \Sigma^* S_1(L)) \setminus \left(\Sigma^* \left(\Sigma^2(F_2(L))\Sigma^*\right)\right) \qquad (*)$$

On définit $M = (\Sigma, Q, q_\varepsilon, F, \delta)$, où

$$Q = \{q_\varepsilon\} \cup \{q_a \mid a \in \Sigma\}$$

$$F = \begin{cases} \{q_a \mid a \in S_1(L)\} & \text{si } \varepsilon \notin L \\ \{q_a \mid a \in S_1(L)\} \cup \{q_\varepsilon\} & \text{sinon} \end{cases}$$

$$\forall (q, a) \in Q \times \Sigma,\ \delta(q, a) = \begin{cases} q_a & \text{si } q = q_\varepsilon \text{ et } a \in P_1(L) \\ q_a & \text{si } q = q_b \text{ et } ba \in F_2(L) \\ \text{non définie} & \text{sinon} \end{cases}$$

Alors $M$ est local (même standard) et $\mathcal{L}(M) = L$
On le montre par double inclusion en utilisant l'égalité $(*)$ $\boxed{\text{Exo}}$.

$\boxed{\Leftarrow}$ On sait que $L = \mathcal{L}(M)$ où $M = (\Sigma, Q, q_0, F, \delta)$ est un AFD local.

$$\forall a \in \Sigma,\ \exists q_a \in Q \mid \forall q' \in Q,\ (q', a) \in \mathrm{dom}(\delta) \Rightarrow \delta(q', a) = q_a$$

On définit

$$\begin{aligned} P_1 &= \{a \in \Sigma \mid (q_0, a) \in \mathrm{dom}(\delta)\} = \{a \in \Sigma \mid \delta(q_0, a) = q_a\} \\ S_1 &= \{a \in \Sigma \mid qa \in F\} \\ F_2 &= \{ab \in \Sigma^2 \mid (qa, b) \in \mathrm{dom}(\delta)\} \end{aligned}$$

On vérifie alors que

$$L \setminus \{\varepsilon\} = (P_1\Sigma^* \cap \Sigma^* S_1) \setminus \left(\Sigma^* \left(\Sigma^2 \setminus F_2\right) \Sigma^*\right)$$

$\boxed{\subseteq}$ Soit $u = u_1 \cdots u_n \in L \setminus \{\varepsilon\}$.
$u \in \mathcal{L}(M)$, donc $\forall i \in [\![1\ ;\ n]\!]$, $\delta(q_0, u_1 \cdots u_i)$ est bien défini, notons le $q_i$.
$(q_0, q_i) \in \mathrm{dom}(\delta)$, donc $u_1 \in P_1$.
$q_{u_n} = \delta(q_0, u_1 \cdots u_n) \in F$, donc $u_n \in S_1$.
$\forall i \in [\![1\ ;\ n-1]\!]$,

$$\begin{aligned} \delta(q_0, u_1 \cdots u_{i+1}) &= \delta(\delta(q_0, u_1 \cdots u_i), u_{i+1}) \\ &= \delta(q_i, u_{i+1}) \\ &= q_i \end{aligned}$$

Donc $u_i u_{i+1} \in F_2$, donc

$$u \in (P_1\Sigma^* \cap \Sigma^* S_1) \setminus \left(\Sigma^* \left(\Sigma^2 \setminus F_2\right) \Sigma^*\right)$$

$\boxed{\supseteq}$ par récurrence sur la taille du mot $\boxed{\text{Exo}}$. ■

## 2.2 Automates finis non déterministes (AFND)

### 2.2.1 Introduction

Il peut être complexe, ou simplement fastidieux, de concevoir un AFD reconnaissant un langage donné.

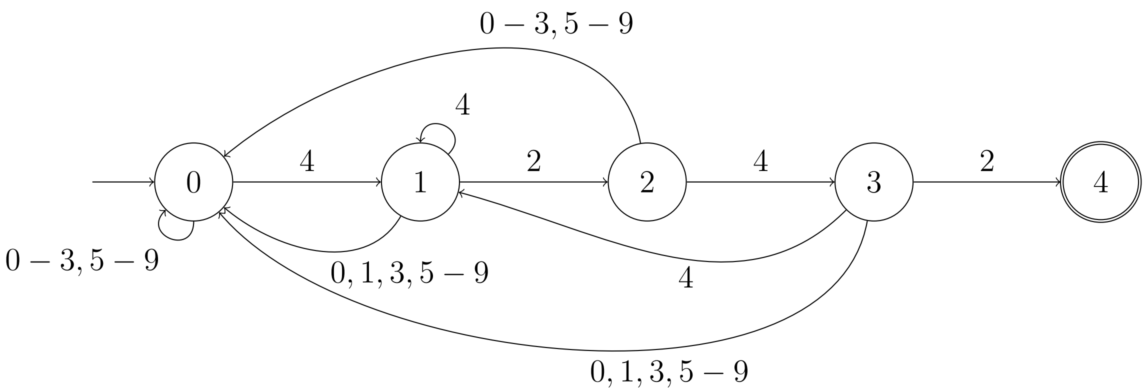
Exemple : digicode : on entre une série de chiffres, et si elle se termine par le code enregistré, la porte se débloque. Cela correspond au fonctionnement d'un automate, mais la difficulté vient fu fait que même si le début du code est entré, il faut pouvoir revenir en arrière si la suite est invalide.

Idée : on compte le nombre de chiffres valides lus consécutivement et on met à jour ce nombre en fonction des chiffres lus.

Par exemple, pour le code 4242, on peut écrire le programme suivant :

```
let nb_valides = ref 0 in
while !nb_valides < 4 do
    let n = read_int () in
    match !nb_valides, n with
    | 0, 4 | 1, 2 | 2, 4 | 3, 2 -> incr nb_valides
    | _, 4 -> nb_valides := 1
    | _ -> nb_valides := 0
done;
unlock_door()
```

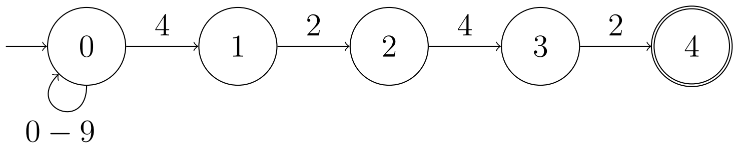
En associant un état à chaque valeur possible de `nb_valides`, ce code correspond à l'automate




Cet automate contient de nombreuses informations "parasites", liées uniquement à la gestion des erreurs de l'entrée. On peut simplifier sa présentation en changeant le modèle de calcul et en exploitant le principe de retour sur trace.

Idée : depuis l'état 0, si on lit un 4, cela peut être le début d'une séquence valide donc on passe dans l'état 1, mais si une erreur survient dans la suite de la séquence, on revient sur ce choix et on décide que la lecture de ce 4 conserve l'état 0.

Graphiquement, cela simplifie l'automate en :




L'algorithme de retour sur trace revient à parcourir ce graphe selon les lettres lues en essayant de passer dans l'état 1 dès que possible et en revenant sur ce choix en cas d'impossibilité. Ce type d'automate est appelé *non déterministe* car on peut le voir comme une machine pouvant être dans plusieurs états à la fois ( par exemple 0 et 1 après la lecture d'un 4 depuis l'état 0). Lorsqu'une transition est impossible, la machine n'est plus dans l'état associé (idée du retour sur trace), et l'objectif est que l'un des états de la machine sont acceptant en fin de lecture.

### 2.2.2 Définition (*Automate fini non déterministe*)

Un *automate fini non déterministe* (AFND) est un quintuplet $M = (\Sigma, Q, I, F, \delta)$

- $\Sigma$ est un alphabet ;
- $Q$ est un ensemble fini non vide d'états ;
- $I \subseteq Q$ est l'ensemble des états initiaux ;
- $F \subseteq Q$ est l'ensemble des états acceptants ;
- $\delta : Q \times \Sigma \longrightarrow \mathcal{P}(Q)$

Remarque : $\delta$ est totale, mais son image peut être vide pour certains couples.

Représentation graphique : comme les AFD, avec un arc $q \xrightarrow{a} q'$ $\forall (q, q', a) \in Q^2 \times \Sigma \mid q' \in \delta(q, a)$.

### 2.2.3 Modèle de calcul associé

• Définition (*transition généralisées*) : Soit $M = (\Sigma, Q, I, F, \delta)$ un AFND.
On généralise $\delta$ aux mots en définissant $\forall q \in Q,\ \delta(q, \varepsilon) = \{q\}$, et

$$\forall (q, u, a) \in Q \times \Sigma^* \times \Sigma,\ \delta(q, au) = \bigcup_{q' \in \delta(q,a)} \delta(q', u)$$

$\boxed{\text{Exo}}$ Montrer que c'est équivalent à :

$$\forall (q, u, a) \in Q \times \Sigma^* \times \Sigma,\ \delta(q, ua) = \bigcup_{q' \in \delta(q,u)} \delta(q', a)$$

• Définition (*langage reconnu*) : Soit $M = (\Sigma, Q, I, F, \delta)$ un AFND.

- $\forall u \in \Sigma^*$, $u$ est accepté par $M$ si et seulement si

$$\left( \bigcup_{q_0 \in I} \delta(q_0, u) \right) \cap F \neq \varnothing$$

– Le langage reconnu par $M$ est

$$\mathcal{L}(M) = \{u \in \Sigma^* \mid M \text{ acceptable } u\}$$

• Proposition :

> Soit $M = (\Sigma, Q, I, F, \delta)$ un AFND, et $u \in \Sigma^*$
> Alors $u \in \mathcal{L}(M) \iff$ il existe un chemin $q_0 \cdots q_n$ dans le graphe de $M$ avec $q_0 \in I,\ q_n \in F$, et la concaténation des étiquettes sur ce chemin est $u$.

☐ Démonstration :

$\boxed{\Leftarrow}$ On montre que $\forall i \in [\![0\ ;\ n]\!],\ q_i \in \delta(q_0, u_1 \cdots u_i)$ (on note $u = u_1 \cdots u_i$)

– $i = 0$ : $\delta(q_0, u_1 \cdots u_0) = \delta(q_0, \varepsilon) = \{q_0\} \ni q_0$

– Hérédité : si $i \in [\![1\ ;\ n-1]\!] \mid q_i \in \delta(q_0, u_1 \cdots u_i)$, alors

$$\bigcup_{q \in \delta(q_0, u_1 \cdots u_i)} \delta(q, u_{i+1}) \supseteq \delta(q_i, u_{i+1}) \ni q_{i+1}$$

(car $q_i \xrightarrow{u_{i+1}} q_{i+1}$)
Finalement, $q_n \in F \cap \delta(\underbrace{q_0}_{\in I}, \underbrace{u_1 \cdots u_n}_{u})$, donc $u \in \mathcal{L}(M)$.

$\boxed{\Rightarrow}$ $\exists q_0 \in I,\ \exists q_n \in F \mid q_n \in \delta(q_0, u)$.
On montre qu'il existe, $\forall i \in [\![0\ ;\ n]\!]$ un chemin $q_i \cdots q_n$ étiqueté par $u_{i+1} \cdots u_n$ avec $q_i \in \delta(q_0, u_1 \cdots u_i)$.

– $i = n$, $q_n \in \delta(q_0, u_1 \cdots u_n)$, et le chemin vide étiqueté par $\varepsilon = u_{n+1} u_n$.

– Hérédité : si $i \in [\![1\ ;\ n]\!] \mid \exists q_i \cdots q_n$ un chemin étiqueté par $u_{i+1} \cdots u_n$ avec $q_i \in \delta(q_0, u_1 \cdots u_i)$, alors

$$q_i \in \delta(q_0, u_1 \cdots u_i) = \bigcup_{q \in \delta(q_0, u_1 \cdots u_{i-1})} \delta(q, u_i)$$

Donc $\exists q_{i-1} \in \delta(q_0, u_1 \cdots u_{i-1}) \mid q_i \in \delta(q_{i-1}, u_i)$.
Le chemin $q_{i-1}, q_n$ est étiqueté par $u_i \cdots u_n$.
Finalement, on a construit un chemin $q_0 \cdots q_n$ convenable. ■

### 2.2.4 Déterminisation

• Théorème :

> Soit $\Sigma$ un alphabet, et $L$ un langage sur $\Sigma$.
>
> $$L \in \mathsf{Rec}(\Sigma) \iff \exists M = (\Sigma, Q, I, F, \delta) \text{ un AFND} \mid \mathcal{L}(M) = L$$

☐ Démonstration :

$\boxed{\Rightarrow}$ Si $M = (\Sigma, Q, q_0, F, \delta)$ est un AFD tel que $L = \mathcal{L}(M)$, on construit $M' = (\Sigma, Q, \{q_0\}, F, \delta')$, où $\forall q \in Q, \forall a \in \Sigma$,

$$\delta'(q,a) = \begin{cases} \{\delta(q,a)\} & \text{si } (q,a) \in \text{dom}(\delta) \\ \varnothing & \text{sinon} \end{cases}$$

Les graphes de $M$ est $M'$ sont identiques, donc $L = \mathcal{L}(M) = \mathcal{L}(M')$.

$\boxed{\Leftarrow}$ On construit l'automate des parties

$$M' = (\Sigma, \mathcal{P}(Q), I, F', \delta')$$

où

$$F' = \{E \in \mathcal{P}(Q) \mid E \cap F \neq \varnothing\}$$
$$\forall E \in \mathcal{P}(Q), \forall a \in \Sigma,\ \delta'(E,a) = \bigcup_{q \in E} \delta(q,a)$$

$M'$ est un AFD.
On montre par récurrence que

$$\forall (n,u) \in \mathbb{N} \times \Sigma^* \mid\ |u| = n,\ \delta'(I,u) = \bigcup_{q_0 \in I} \delta(q_0,u)$$

– $n = 0$ : $u = \varepsilon$ est le seul mot possible, et

$$\delta'(I,\varepsilon) = I = \bigcup_{q_0 \in I} \{q_0\} = \bigcup_{q_0 \in I} \delta(q_0,\varepsilon)$$

– Hérédité : Soit $n \in \mathbb{N} \mid \forall u \in \Sigma^* \mid n = |u|,\ \delta'(I,u) = \bigcup_{q_0 \in I} \delta(q_0,u)$.
Soit $u \in \Sigma^* \mid\ |u| = n+1$. Alors $u$ s'écrit $u = va$ avec $v \in \Sigma^*, a \in \Sigma \mid\ |v| = n$.

$$\begin{aligned} \delta'(I,u) &= \delta'(I,va) \\ &= \bigcup_{q \in \delta'(I,v)} \delta'(q,a) \\ &= \bigcup_{q_0 \in I} \bigcup_{q \in \delta(q_0,u)} \delta(q,a) \\ &= \bigcup_{q_0 \in I} \delta(q_0,\varepsilon) \end{aligned}$$

Soit $u \in \Sigma^*$.

$$\begin{aligned} u \in L &\Leftrightarrow u \in \mathcal{L}(M) \\ &\Leftrightarrow \bigcup_{q_0 \in I} \delta(q_0,u) \cap F \neq \varnothing \\ &\Leftrightarrow \delta'(I,u) \cap F \neq \varnothing \\ &\Leftrightarrow \delta'(I,u) \in F' \\ &\Leftrightarrow u \in \mathcal{L}(M') \end{aligned}$$

■

• Remarque : l'automate des parties est complet et possède un nombre d'états exponentiel en le nombre d'états de l'AFND associé. On peut réduire ce nombre d'états en ne considérant que les ensembles d'états de l'AFND qui sont accessibles depuis $I$. L'algorithme de déterminisation accessible s'exprime ainsi :

**Algorithm 2:** Déterminisation accessible

1 $E \leftarrow \{I\}$;
2 **while** $\exists P \in E$ pas encore traité **do**
3 $\quad \forall a \in \Sigma$, construire $\delta'(P, a)$ et l'ajouter à $E$;

Exemple : $L = (a|b)^*baa$ est reconnu par l'AFND

$$
\begin{tikzcd}
\arrow[r] & q_0 \arrow[loop above, "{a,b}"] \arrow[r, "b"] & q_1 \arrow[r, "a"] & q_2 \arrow[r, "a"] & q_3
\end{tikzcd}
$$

| | | | | |
|---|---|---|---|---|
| | $\{q_0\}$ | $\{q_0, q_1\}$ | $\{q_0, q_2\}$ | $\{q_0, q_3\}$ |
| $a$ | $\{q_0\}$ | $\{q_0, q_2\}$ | $\{q_0, q_3\}$ | $\{q_0\}$ |
| $b$ | $\{q_0, q_1\}$ | $\{q_0, q_1\}$ | $\{q_0, q_1\}$ | $\{q_0, q_1\}$ |

L'AFD obtenu est :

$$
\begin{tikzcd}
\arrow[r] & \{q_0\} \arrow[loop above, "a"] \arrow[r, "b"] & \{q_0, q_1\} \arrow[loop above, "b"] \arrow[d, bend right, "a"'] \\
& \{q_0, q_3\} \arrow[u, "a"] \arrow[ur, "b"] & \{q_0, q_2\} \arrow[l, "a"] \arrow[u, bend right, "b"']
\end{tikzcd}
$$

$\boxed{\text{Exo}}$ le langage $(a|b)^*a(a|b)^n$ est reconnu par un AFND à $n+2$ états mais un AFD reconnaissant ce langage a au moins $2^{n+1}$ états.

### 2.2.5 Propriétés de clôture sur $\mathsf{Rec}(\Sigma)$

• Clôture par complémentation : Soit $L \in \mathsf{Rec}(\Sigma)$. Alors

$$\Sigma^* \setminus L \in \mathsf{Rec}(\Sigma)$$

☐ Démonstration :

$\exists M = (\Sigma, Q, q_0, F, \delta)$ un AFD complet tel que $L = \mathcal{L}(M)$.
On considère $M' = (\Sigma, Q, q_0, Q \setminus F, \delta)$.

$$\begin{array}{rcll} u \in \mathcal{L}(M') & \Leftrightarrow & \delta(q_0, u) \in Q \setminus F & \\ & \Leftrightarrow & \delta(q_0, u) \notin F & \text{car } (q_0, u) \text{ existe car } M \text{ est complet} \\ & \Leftrightarrow & u \notin \mathcal{L}(M) & \\ & \Leftrightarrow & u \in \Sigma^* \setminus L & \end{array}$$

■

• Clôture par union : Soient $L_1, L_2 \in \mathsf{Rec}(\Sigma)$.
Alors :

$$L_1 \cup L_2 \in \mathsf{Rec}(\Sigma)$$

☐ Démonstration :

Idée : on lit le mot dans les deux automates en parallèle et on accepte si l'un des deux accepte.

$$\forall i \in \{1, 2\}\,,\ \exists M_i = (\Sigma, Q_i, I_i, F_i, \delta_i) \mid \mathcal{L}(M_i) = L_i$$

On construit

$$M = (\Sigma,\ Q_1 \uplus Q_2,\ I_1 \uplus I_2,\ F_1 \uplus F_2,\ \delta)$$

où $\forall q \in Q_1 \uplus Q_2,\ \forall a \in \Sigma,$

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{si } q \in Q_1 \\ \delta_2(q, a) & \text{si } q \in Q_2 \end{cases}$$

$\boxed{\text{Exo}}$ $\mathcal{L}(M) = L_1 \cup L_2$ ■

• Clôture par intersection : Soient $L_1, L_2 \in \mathsf{Rec}(\Sigma)$.
Alors

$$L_1 \cap L_2 \in \mathsf{Rec}(\Sigma)$$

☐ Démonstration :

On a :

$$\forall i \in \{1, 2\}\,,\ \exists M_i = (\Sigma, Q_i, I_i, F_i, \delta_i) \mid \mathcal{L}(M_i) = L_i$$

On construit l'automate produit $M = (\Sigma, Q_1 \times Q_2, I_1 \times I_2, F_1 \times F_2, \delta)$
où $\forall (q_1, q_2) \in Q_1 \times Q_2,\ \forall a \in \Sigma,\ \delta((q_1, q_2), a) = \delta_1(q_1, a) \times \delta_2(q_2, a)$
$\boxed{\text{Exo}}$ $\mathcal{L}(M) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$ ■

• Clôture par concaténation : Soient $L_1, L_2 \in \mathsf{Rec}(\Sigma)$.
Alors :

$$L_1 L_2 \in \mathsf{Rec}(\Sigma)$$

☐ Démonstration :

Idée : on met bout à bout les automates en passant du premier au second de manière déterministe.

$$\forall i \in \{1, 2\}\,,\ \exists M_i = (\Sigma, Q_i, q_i, F_i, \delta_i) \mid \mathcal{L}(M_i) = L_i$$

On suppose $M_1$ et $M_2$ complets et $M_2$ standard.
On construit $M = (\Sigma, Q_1 \cup Q_2 \setminus \{q_2\}, \{q_1\}, F, \delta)$, où :

$$F = \begin{cases} F_2 & \text{si } q_2 \notin F_2 \\ F_1 \cup F_2 \setminus \{q_2\} & \text{sinon} \end{cases}$$

et

$$\forall (q, a) \in (Q_1 \cup Q_2 \setminus \{q_2\}) \times \Sigma,\ \ \delta(q, a) = \begin{cases} \{\delta_1(q, a)\} & \text{si } q \in Q_1 \setminus F_1 \\ \{\delta_2(q, a)\} & \text{si } q \in Q_2 \setminus \{q_2\} \\ \{\delta_1(q, a), \delta_2(q_2, a)\} & \text{sinon} \end{cases}$$

$\boxed{\text{Exo}}$ $\mathcal{L}(M) = L_1 L_2$ ■

## 2.3 Transitions spontanées

### 2.3.1 Introduction

On ajoute de l'indéterminisme aux automates en autorisant un changement d'état sans lire de lettre.

Ce nouveau type de transition s'appelle *transition spontanée / instantanée*, ou $\varepsilon$-transition, qui correspond à une transition étiquetée par le mot vide.

### 2.3.2 Définition (*automate fini non déterministe à transitions spontanées*)

Un AFND à transitions spontanées ($\varepsilon$-AFND) est un quintuplet $(\Sigma, Q, I, F, \delta)$, où

- $\Sigma$ est un alphabet ;
- $Q$ est un ensemble fini non vide d'états ;
- $I \subseteq Q$ est l'ensemble des états initiaux ;
- $F \subseteq Q$ est l'ensemble des états acceptants ;
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \longrightarrow \mathcal{P}(Q)$ est la fonction de transition.

### 2.3.3 Modèle de calcul associé

• Principe : un $\varepsilon$-AFND fonctionne comme un AFND mais à tout moment dans la lecture du mot il est possible de suivre un nombre quelconque d'$\varepsilon$-transitions. Il faut donc déterminer pour chaque état les états accessibles depuis ce dernier *via* des transitions spontanées. C'est un problème de graphe simple à résoudre.

• Définition (*ε-clôture*) : Soit $M = (\Sigma, Q, I, F, \delta)$ un $\varepsilon$-AFND, et $q \in Q$.
On appelle *ε-clôture* de $q$ l'ensemble

$$E(q) = \left\{ q' \in Q \mid \exists n \in \mathbb{N},\ \exists (q_0, \dots, q_n) \in Q^{n+1} \left| \begin{array}{l} q_0 = q \\ q_n = q' \\ \forall i \in [\![0\ ;\ n-1]\!],\ q_{i+1} \in \delta(q_i, \varepsilon) \end{array} \right. \right\}$$

• Définition (*transitions généralisées*) : Soit $M = (\Sigma, Q, I, F, \delta)$ un $\varepsilon$-AFND.
On généralise $\delta$ aux mots par :

$$\forall q \in Q,\ \hat{\delta}(q, \varepsilon) = E(q)$$

$$\forall (q, u, a) \in Q \times \Sigma^* \times \Sigma,\ \hat{\delta}(q, au) = \bigcup_{q' \in E(q)} \bigcup_{q" \in \delta(q', a)} \hat{\delta}(q", u)$$

$\boxed{\text{Exo}}$ Mq c'est équivalent à

$$\forall (q, u, a) \in Q \times \Sigma^* \times \Sigma,\ \hat{\delta}(q, ua) = \bigcup_{q' \in \hat{\delta}(q, u)} \bigcup_{q" \in \delta(q', a)} E(q")$$

• Définition (*langage reconnu*) : Soit $M = (\Sigma, Q, I, F, \delta)$ un $\varepsilon$-AFND.
– $u \in \Sigma^*$ est accepté par $M$ ssi

$$\bigcup_{q_0 \in I} \hat{\delta}(q_0, u) \cap F \neq \varnothing$$

– Le langage reconnu par $M$ est

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

• Proposition

> Soit $M = (\Sigma, Q, I, F, \delta)$ un $\varepsilon$-AFND, et $u \in \Sigma^*$.
> Alors
>
> $$u \in \mathcal{L}(M) \ \Leftrightarrow$$
>
> $$\exists q_0, \dots, q_n \in Q \left| \begin{array}{l} q_0 \in I \\ q_n \in F \\ \exists a_1, \dots, a_n \in \Sigma \cup \{\varepsilon\} \left| \begin{array}{l} \forall i \in [\![0\ ;\ n-1]\!],\ q_{i+1} \in \delta(q_i, a_{i+1}) \\ u = a_1 \cdots a_n \end{array} \right. \end{array} \right.$$

☐ Démonstration :

$\boxed{\Leftarrow}$ On montre par récurrence que $\forall i \in [\![0\ ;\ n]\!],\ q_i \in \hat{\delta}(q_0, a_1 \cdots a_i)$ :
– $i = 0$ : $a_1 \cdots a_i = \varepsilon$ et $q_0 \in E(q_0) = \hat{\delta}(q_0, \varepsilon)$.
– Si $\forall j \leqslant i$, la propriété est vraie, deux cas :
  * $a_{i+1} \neq \varepsilon$ : $q_{i+1} \in \delta(q_i, a_{i+1})$ et $q_i \in \hat{\delta}(q_0, a_1 \cdots a_i)$.

Or $\hat{\delta}(q_0, a_1 \cdots a_{i+1}) = \bigcup\limits_{q \in \hat{\delta}(q_0, a_1 \cdots a_i)} \bigcup\limits_{q' \in \delta(q, a_{i+1})} E(q')$.

Donc $q_{i+1} \in E(q_{i+1}) \subseteq \bigcup\limits_{q \in \delta(q_i, a_{i+1})} E(q') \subseteq \hat{\delta}(q_0, a_1 \cdots a_{i+1})$

$*$ $a_{i+1} = \varepsilon$ : deux cas :

$+$ $\forall j \leqslant i,\ a_j = \varepsilon$. Dans ce cas, $q_{i+1} \in E(q_0) = \hat{\delta}(q_0, \varepsilon) = \hat{\delta}(q_0, a_1 \cdots a_{i+1})$

$+$ $\exists j \leqslant i \mid a_j \neq \varepsilon$ : on considère $j = \max\{k \leqslant i \mid a_k \neq \varepsilon\}$

On sait que $a_1 \cdots a_{i+1} = a_1 \cdots a_j$.

$$q_{i+1} \in E(q_j) \subseteq \bigcup_{q' \in \delta(q_{j-1}, a_j)} E(q') \subseteq \bigcup_{q \in \hat{\delta}(q_0, a_1 \cdots a_{j-1})} \bigcup_{q' \in \delta(q, a_j)} E(q') = \hat{\delta}(q_0, a_1 \cdots a_j)$$

$\boxed{\Rightarrow}$ On montre par récurrence sur $n \in \mathbb{N}$ qur $\forall u \in \Sigma^* \mid |u| = n,\ \forall q, q' \in Q$, si $q' \in \hat{\delta}(q, u)$, alors il exste un chemin $q_0 \cdots q_n$ dont les étiquettes $a_1 \cdots a_n$ vérifient $u = a_1 \cdots a_n$ et $q_0 = q, q_n = q'$.

$-$ $n = 0$ : $u = \varepsilon$ et $q' \in E(q)$ donc la définition de l'$\varepsilon$-clôture donne le chemin.

$-$ Si la propriété est vraie pour $n \in \mathbb{N}$, soit $u \in \Sigma^* \mid |u| = n+1$, soit $q, q' \in Q \mid q' \in \hat{\delta}(q, u)$.

$\exists v \in \Sigma^* \mid |v| = n,\ \exists a \in \Sigma \mid u = av$.

$q' \in \hat{\delta}(q, av) = \bigcup\limits_{q'' \in E(q)} \bigcup\limits_{q''' \in \delta(q'', q)} \hat{\delta}(q''', v)$

Première union : chemin étiqueté par $\varepsilon \cdots \varepsilon$ de $q$ à $q'$

Deuxième union : avec transition étiquetée par $a$ de $q''$ à $q'''$

$\hat{\delta}(q'', v)$ : H.R : le chemin étiqueté par $a_1 \cdots a_n$ tel que $a_1 \cdots a_n = v$ de $q'''$ à $q'$.

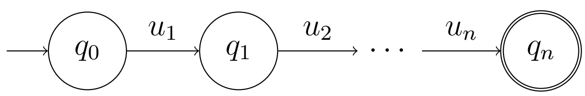
Donc chemin de $q$ à $q'$ étiqueté par $\varepsilon \cdots \varepsilon$ de $a_1 \cdots a_n = av = u$.

$-$ Si $u \in \mathcal{L}(M)$, alors $\exists q_0 \in I, \exists q_n \in F \mid q_n \in \hat{\delta}(q_0, u)$

donc la propriété démontrée donne le chemin voulu. ■

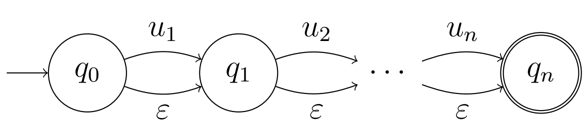
- Exemple : si $u = u_1 \cdots u_n$,

$-$ l'AFD




reconnait $\{u\}$.

$-$ l'$\varepsilon$-AFND



reconnait $\{v \in \Sigma^* \mid v \text{ sous-mot de } u\}$

### 2.3.4 Théorème

> Soit $\Sigma$ un alphabet, et $L$ un langage sur $\Sigma$.
>
> $$L \in \mathsf{Rec}(\Sigma) \Leftrightarrow \exists M = (\Sigma, Q, I, F, \delta) \text{ un } \varepsilon\text{-AFND} \mid \mathcal{L}(M) = L$$

☐ Démonstration :

$\boxed{\Rightarrow}$ par 2.2.4 (page 24), $\exists M = (\Sigma, Q, I, F, \delta)$ un AFND tel que $\mathcal{L}(M) = L$.
On construit l'$\varepsilon$-AFND $M' = (\Sigma, Q, I, F, \delta')$, où

$$\forall (q, a) \in Q \times (\Sigma \cup \{\varepsilon\}),\ \delta'(q, a) = \begin{cases} \delta(q, a) & \text{si } a \neq \varepsilon \\ \varnothing & \text{sinon} \end{cases}$$

Les graphes de $M, M'$ sont identiques, donc par 2.3.3 (page 28), $\mathcal{L}(M') = \mathcal{L}(M) = L$

$\boxed{\Leftarrow}$ On élimine les $\varepsilon$-transitions en construisant l'AFND

$$M' = \left(\Sigma, Q, \bigcup_{q_0 \in I} E(q_0), F, \delta'\right)$$

où $\forall (q, a) \in Q \times \Sigma$,

$$\delta'(q, a) = \bigcup_{q' \in \delta(q, a)} E(q')$$

Soit $u \in \Sigma^*$. On note $u = u_1 \cdots u_n$.

$$u \in \mathcal{L}(M') \Leftrightarrow \exists q_0 \cdots q_n \text{ un chemin } \left| \begin{array}{l} q_0 \in \bigcup\limits_{q \in I} E(q) \\ q_n \in F \\ \forall i \in [\![0 \ ;\ n-1]\!],\ q_{i+1} \in \delta'(q_i, u_{i+1}) \end{array} \right.$$

$$\Leftrightarrow \exists q_0 \cdots q_n \text{ un chemin } \left| \begin{array}{l} q_0 \in \bigcup\limits_{q \in I} E(q) \\ q_n \in F \\ \forall i \in [\![0 \ ;\ n-1]\!],\ q_{i+1} \in \bigcup\limits_{q \in \delta(q_i, u_{i+1})} E(q) \end{array} \right.$$

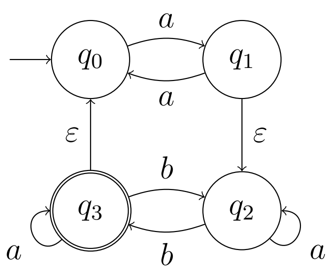
$$\Leftrightarrow \exists q_{-m} \cdots q_0 q_1^{(0)} \cdots q_1^{(m_1)} q_2^{(0)} \cdots q_2^{(m_2)} \cdots q_n^{(0)} \cdots q_n^{(m_n)} \text{ un chemin tel que}$$

$$\left| \begin{array}{l} \forall i \in [\![-m \ ;\ -1]\!],\ q_{i+1} \in \delta(q_i, \varepsilon) \\ \forall i \in [\![1 \ ;\ n]\!],\ q_i^{(0)} \in \delta(q_{i-1}^{(m_{i-1})}, u_i) \\ \forall j \in [\![0 \ ;\ m_i - 1]\!],\ q_i^{(j+1)} \in \delta(q_i^{(j)}, \varepsilon) \\ q_0 \in I \\ q_n^{(m_n)} \in F \end{array} \right.$$

$$\Leftrightarrow u \in \mathcal{L}(M) \ \blacksquare$$

### 2.3.5 Exemple

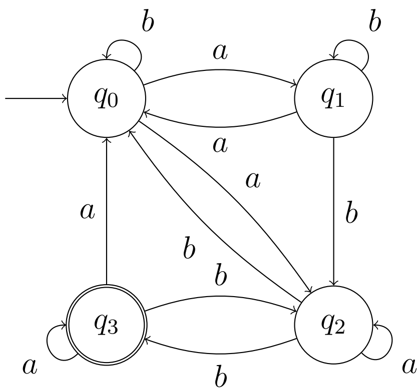
$\Sigma = \{a, b\}$, et $\forall c \in \Sigma$, on note $L_c = \{u \in \Sigma^* \mid |u|_c \equiv 1\ [2]\}$
$(L_a L_b)^+$ est reconnu par :




$E(q_0) = \{q_0\}$
$E(q_1) = \{q_1, q_2\}$
$E(q_2) = \{q_2\}$
$E(q_3) = \{q_3, q_0\}$

$a(bb)$
$\to (ab)b \in \mathcal{L}(M)$

$(L_a L_b)^+$ est donc reconnu par




### 2.3.6 Retour aux propriétés de clôture sur $\mathsf{Rec}(\Sigma)$

- Théorème :

$$\mathsf{Reg}(\Sigma) \subseteq \mathsf{Rec}(\Sigma)$$

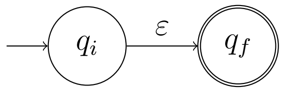
□ Démonstration :

On procède par induction structurelle sur une expression régulière dénotant le langage en appliquant la construction de THOMPSON :

– $\varnothing$ :

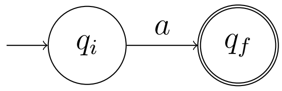

$M_\varnothing = (\Sigma, \{q_i, q_f\}, \{q_i\}, \{q_f\}, \varnothing)$

– $\varepsilon$ :



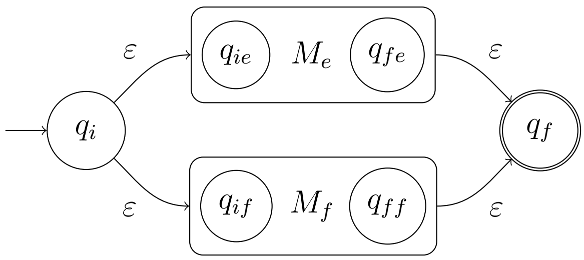

$M_\varepsilon = (\Sigma, \{q_i, q_f\}, \{q_i\}, \{q_f\}, \{(q_i, \varepsilon) \longmapsto \{q_f\}\})$

– $a \in \Sigma$




$M_a = (\Sigma, \{q_i, q_f\}, \{q_i\}, \{q_f\}, \{(q_i, a) \longmapsto \{q_f\}\})$

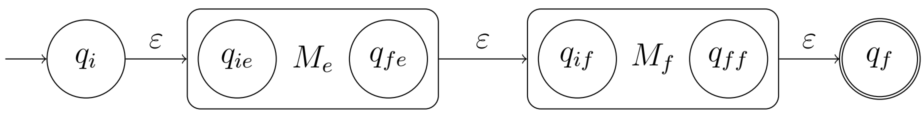
– $e|f$ :




$$M_{e|f} = \left(\Sigma,\ \{q_i, q_f\} \cup Q_e \cup Q_f,\ \{q_i\},\ \{q_f\},\ \delta_e \cup \delta_f \cup \left| \begin{array}{l} (q_i, \varepsilon) \longmapsto \{q_{ie}, q_{if}\} \\ (q_{fe}, \varepsilon) \longmapsto \{q_f\} \\ (q_{ff}, \varepsilon) \longmapsto \{q_f\} \end{array} \right. \right)$$

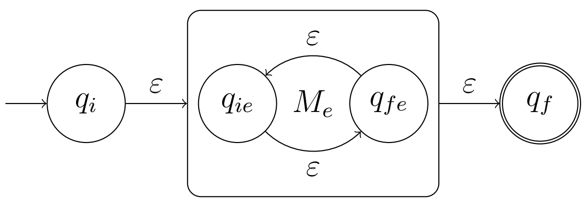
$u \in \mathcal{L}(M_{e|f}) \ \Leftrightarrow$
il existe un chemin de $q_i$ à $q_f$ dont les étiquettes donnent $u$
$\Leftrightarrow$ il existe un chemin de $q_{ie}$ à $q_{fe}$ dont les étiquettes donnent $u$
ou il existe un chemin de $q_{if}$ à $q_{ff}$ dont les étiquettes donnent $u$
$\Leftrightarrow$ $u \in \mathcal{L}(M_e)$ ou $u \in \mathcal{L}(M_f)$
$\Leftrightarrow$ $u \in \mathcal{L}(M_e) \cup \mathcal{L}(M_f) = \mathcal{L}(e) \cup \mathcal{L}(f) = \mathcal{L}(e|f)$.

– $ef$




$$M_{ef} = \left(\Sigma,\ Q_e \cup Q_f \cup \{q_i, q_f\},\ \{q_i\},\ \{q_f\},\ \delta_e \cup \delta_f \cup \left| \begin{array}{l} (q_i, \varepsilon) \longmapsto \{q_{ie}\} \\ (q_{fe}, \varepsilon) \longmapsto \{q_{if}\} \\ (q_{ff}, \varepsilon) \longmapsto \{q_f\} \end{array} \right. \right)$$

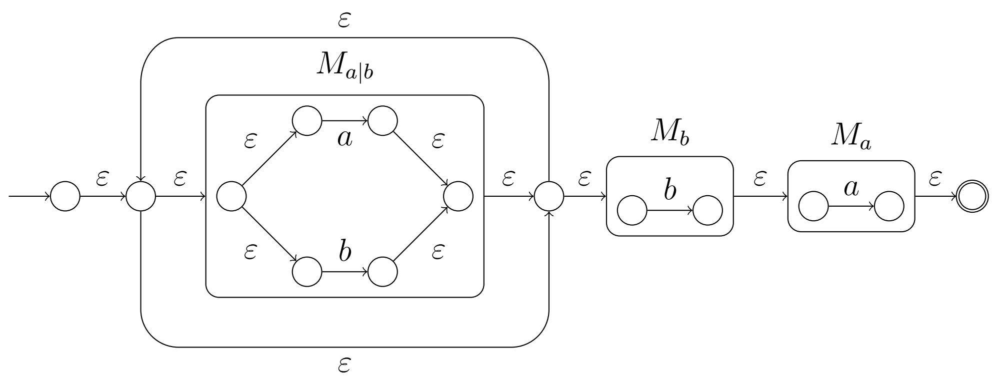
– $e^*$ :



$$M_{e^*} = \left( \Sigma,\ Q_e \cup \{q_i, q_f\}\ ,\ \{q_i\}\ ,\ \{q_f\}\ ,\ \delta_e \cup \left| \begin{array}{l} (q_i, \varepsilon) \longmapsto \{q_{ie}\} \\ (q_{ie}, \varepsilon) \longmapsto \{q_{fe}\} \\ (q_{fe}, \varepsilon) \longmapsto \{q_{ie}, q_f\} \end{array} \right. \right)$$

$\boxed{\text{Exo}}$ Comme $M_e$ est standard, $\mathcal{L}(M_{e^*}) = \mathcal{L}(e^*)$ ■

- Exemple : $((a|b)^*)(ba)$




## 2.4 Théorème de Kleene

### 2.4.1 Introduction

Le théorème de Kleene énonce le fait que les langages réguliers et les automates finis ont la même expressivité : les langages réguliers sont les mêmes langages que les langages reconnaissables.

Remarque : le théorème est déjà démontré : la construction de Thompson (*cf* 2.3.5, page 32) montre que les langages réguliers sont reconnaissables et on a vu qu'il était possible de déterminer le langage reconnu par un AFD à l'aide du lemme d'Arden (*cf* 2.1.3, page 16). Ce langage est construit à l'aide d'opérations régulières, donc il est régulier.

Rappel : Soit $M = (\Sigma, Q, q_0, F, \delta)$ un AFD.
$\forall q \in Q$, on définit le système d'équations d'inconnues $L_q$ suivant :

$$\forall q \in Q,\ L_q = \begin{cases} \displaystyle\bigcup_{q' \in Q} \Sigma_{q,q'} L_{q'} & \text{si } q \notin F \\ \{\varepsilon\} \cup \displaystyle\bigcup_{q' \in Q} \Sigma_{q,q'} L_{q'} & \text{sinon} \end{cases}$$

où $\Sigma_{q,q'} = \{a \in \Sigma \mid \delta(q,a) = q'\}$

On remarque que $\forall q \in Q$, $\Sigma_{q,q'}$ ne contient pas le mot vide ($\Sigma_{q,q'} \subseteq \Sigma$), donc le lemme d'Arden permet d'obtenir l'expression de $L_q$ en fonction des $L_{q'}$, $\forall q' \in Q \setminus q$.
Réinjecter cette expression dans les autres équations permet d'obtenir un système avec une équation et une inconnue de moins, que l'on peut résoudre récursivement.

Nous allons maintenant voir une autre démonstration du théorème de Kleene , au programme cette fois.

### 2.4.2 Théorème (Kleene, sens direct)

> Soit $\Sigma$ un alphabet.
>
> $$\mathsf{Reg}(\Sigma) \subseteq \mathsf{Rec}(\Sigma)$$

☐ Démonstration :

Soit $L \in \mathsf{Reg}(\Sigma)$. On a donc

$$\exists e \in \mathsf{Regexp}(\Sigma) \mid L = \mathcal{L}(e)$$

On construit un automate reconnaissant $L$ grâce à l'algorithme de Berry-Sethi :

**Algorithm 3:** Berry-Sethi

1 Linéariser $e$ en une expression régulière $e'$ sur $\Sigma \times [\![1\ ;\ n]\!]$ (*cf* 1.4.5, page 14);
2 Construire l'AFD local $M = (\Sigma \times [\![1\ ;\ n]\!], Q, q_0, F, \delta)$ reconnaissant $\mathcal{L}(e')$ comme en 2.1.7 (page 20) (rappel : $\mathcal{L}(e')$ est local, *cf* 1.4.5);
3 Effacer les indices des symboles pour obtenir un AFND $M' = (\Sigma, Q, \{q_0\}, F, \delta')$, où :

$$\forall (q,a) \in Q \times \Sigma,\ \ \delta'(q,a) = \{q' \in Q \mid \exists i \in [\![1\ ;\ n]\!] \mid \delta(q,(a,i)) = q'\}$$

$M'$ est appelé l'automate de Glushkov associé à $e$, et on peut, de manière optionnelle, le déterminiser (*cf* 2.2.4, page 24)

La correction de l'algorithme s'exprime ainsi :

$\mathcal{L}(M') = \mathcal{L}(e)$.

Soit $u \in \Sigma^*$. On le note $u = u_1 \cdots u_m$.

$u \in \mathcal{L}(M') \ \Leftrightarrow\ \exists$ un chemin $q_0 \xrightarrow{u_1} q_1 \xrightarrow{u_2} \cdots \xrightarrow{u_m} q_m$ avec $q_m \in F$ dans le graphe $M'$

$\Leftrightarrow\ \exists i_1, \dots, i_m,\ \exists$ un chemin $q_0 \xrightarrow{(u_1,i_1)} q_1 \xrightarrow{(u_2,i_2)} \cdots \xrightarrow{(u_m,i_m)} q_m$ avec $q_m \in F$ dans le graphe de $M$.

$\Leftrightarrow\ \exists i_1, \dots, i_m \mid (u_1,i_1), \dots, (u_m,i_m) \in \mathcal{L}(M) = \mathcal{L}(e')$

$\Leftrightarrow\ u = u_1 \cdots u_m \in \mathcal{L}(e)$ (*cf* exo en 1.4.5, page 14) ■

### 2.4.3 Exemple

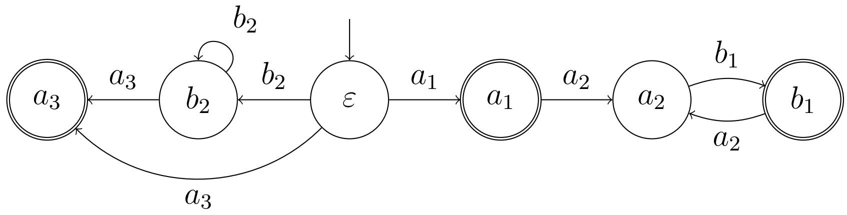
On considère $e = a(ab)^*|b^*a$.

(1) On linéarise $e$ en $e' = a_1(a_2b_1)^*|b_2^*a_3$

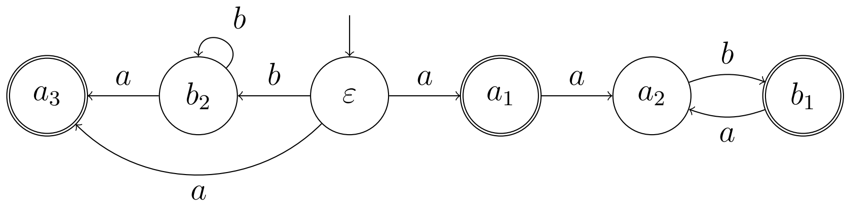
(2) On détermine

$$P_1 = \{a_1, b_2, a_3\}$$
$$S_1 = \{a_1, b_1, a_3\}$$
$$F_2 = \{a_1a_2, a_2b_1, b_1a_2, b_2a_3, b_2b_2\}$$

D'où l'automate $M$ :

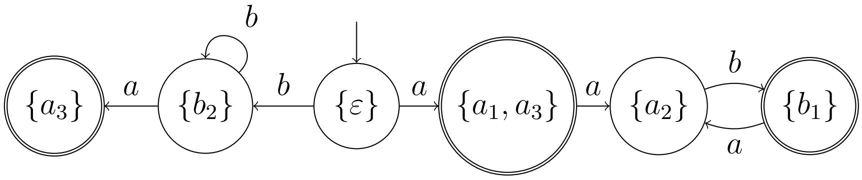



(3) L'automate de Glushkov associé à $e$ est donc




Que l'on peut déterminiser :

| | $\{\varepsilon\}$ | $\{a_1, a_3\}$ | $\{b_2\}$ | $\{a_2\}$ | $\{a_3\}$ | $\{b_1\}$ |
|---|---|---|---|---|---|---|
| $a$ | $\{a_1, a_3\}$ | $\{a_2\}$ | $\{a_3\}$ | $\varnothing$ | $\varnothing$ | $\{a_2\}$ |
| $b$ | $\{b_2\}$ | $\varnothing$ | $\{b_2\}$ | $\{b_1\}$ | $\varnothing$ | $\varnothing$ |




### 2.4.4 Théorème (Kleene, sens réciproque)

> Soit $\Sigma$ un alphabet.
> Alors
> $$\mathsf{Rec}(\Sigma) \subseteq \mathsf{Reg}(\Sigma)$$

☐ Démonstration :

Soit $L \in \mathsf{Rec}(\Sigma)$. Il existe un AFD $M = (\Sigma, Q, q_0, F, \delta) \mid \mathcal{L}(M) = L$.

On construit à partir de $M$ une expression régulière $e$ telle que $\mathcal{L}(e) = \mathcal{L}(M) = L$, en appliquant l'algorithme de Brzozavski et Mc Cuskey dont le principe est le suivant :

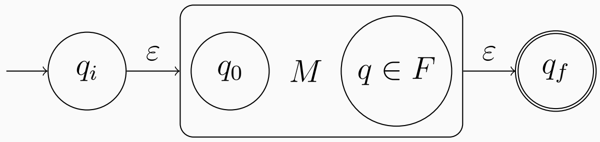
**Algorithm 4:** Brzozavski et Mc Cuskey

On construit l'automate généralisé

$$M' = (\mathsf{Regexp}(\Sigma), Q \cup \{q_i, q_f\}, q_i, \{q_f\}, \delta')$$

où $\forall (q, e) \in Q \times \mathsf{Regexp}(\Sigma)$,

$$\delta'(q, e) = \begin{cases} q_0 & \text{si } e = \varepsilon \text{ et } q = q_i \\ q_f & \text{si } e = \varepsilon \text{ et } q \in F \\ \delta(q, a) & \text{si } e = a \in \Sigma \text{ et } (q, a) \in \mathrm{dom}(\delta) \\ \text{non défini} & \text{sinon} \end{cases}$$



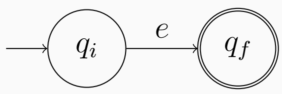

**while** $Q \neq \varnothing$ **do**

Éliminer des transitions multiples : si l'automate contient deux transitions $q \xrightarrow{e} q'$ et $q \xrightarrow{f} q'$, on les remplace par $q \xrightarrow{e|f} q'$ ;

Éliminer un état $q \in Q$ : pour toute transition $p \xrightarrow{e} q$ et $q \xrightarrow{f} r$, on ajoute la transition $p \xrightarrow{ef} r$ s'il n'existe pas de transition de $q$ vers lui-même et $p \xrightarrow{eg^*f} r$ s'il existe une transition $q \xrightarrow{g} q$;

Supprimer $q$ et les transitions associées ;

Après une éventuelle élimination des transitions multiples, l'automate est de la forme

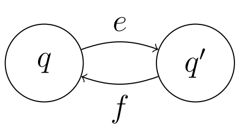



L'expression régulière $e$ est le résultat de l'algorithme;

La preuve de correction de cet algorithme est H.P. Elle nécessiterait de définir le modèle de calcul associé aux automates généralisés. L'idée de la preuve consiste à remarquer que l'algorithme termine ($|Q|$ est un variant de boucle), et démontrer l'invariant $\mathcal{L}(M') = \mathcal{L}(M)$.

Pour cela, on remarque que les transformations conservent le langage de l'automate :

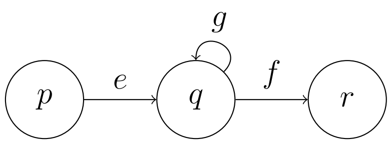
– Si



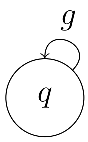

et si la lecture du mot $u$ depuis $q$ mène à $q'$, c'est que $u \in \mathcal{L}(e)$ ou $u \in \mathcal{L}(f)$ donc

$u \in \mathcal{L}(e) \cup \mathcal{L}(f) = \mathcal{L}(e|f)$.

– Si



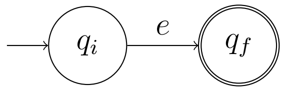

et si la lecture de $u$ mène de $p$ à $r$ *via* $q$, alors en notant $n \in \mathbb{N}$ le nombre de fois que cette lecture passe par la transition




Alors $\exists (x, y, z) \in \mathcal{L}(e) \times \mathcal{L}(g)^n \times \mathcal{L}(f) \mid u = xyz$
Donc $\exists n \in \mathbb{N} \mid u \in \mathcal{L}(e)\mathcal{L}(g)^n\mathcal{L}(f)$, donc

$$u \in \mathcal{L}(e)\mathcal{L}(g)^*\mathcal{L}(f) = \mathcal{L}(eg^*f)$$

Remarque : on obtient un automate de la forme

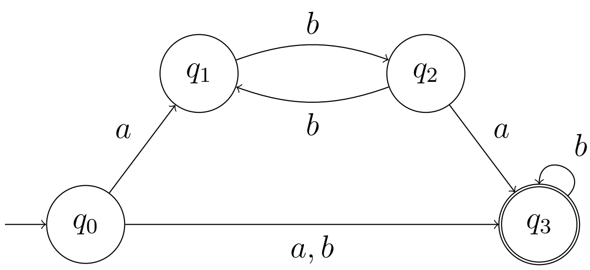



que dans le cas où il existe un chemin de $q_0$ à un état $q \in F$, $i.e$ $q_0$ est co-accessible, $i.e$ $\mathcal{L}(M) \neq \varnothing$.

Si $\mathcal{L}(M) = \varnothing$, ce que l'on peut vérifier en faisant un parcours du graphe de $M$ à partir de $q_0$, il est inutile d'appliquer l'algorithme de Brzozwski et Mc Cuskey (algorithme n°4, page 37) : l'expression régulière $\varnothing$ convient. ■

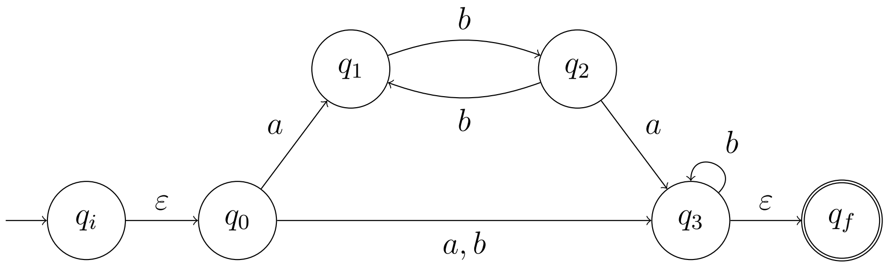
### 2.4.5 Exemple

L'algorithme de Brzozwski et Mc Cuskey (algorithme n°4, page 37) fonctionne également sur les AFND :

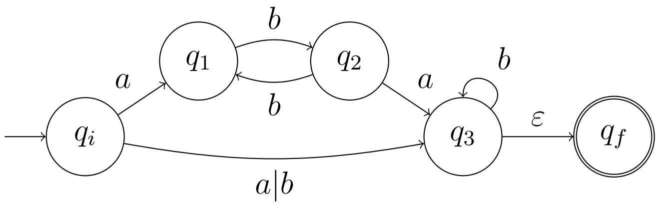



$\mathcal{L}(M) = (a|b|ab(bb)^*a)b^*$

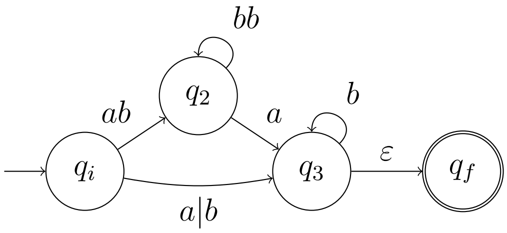
(1) On construit l'automate généralisé :



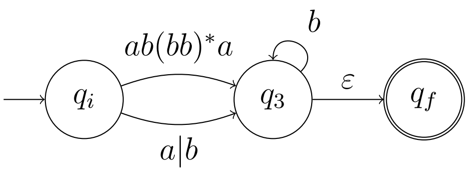

(2) On élimine les transitions multiples de $q_0$ à $q_3$, puis on élimine $q_0$



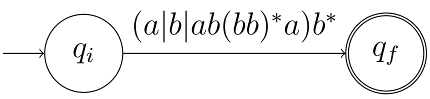

Pas de transition multiples, on élimine $q_1$ :




Pas de transitions multiples, on élimine $q_2$ :




On élimine les transitions multiples entre $q_i$ et $q_3$, puis on élimine $q_3$ :




On conclut que $\mathcal{L}(M) = (a|b|ab(bb)^*a)b^*$.

Remarques :

– L'ordre d'élimination a un impact sur l'expression régulière obtenue.

$\boxed{\text{Exo}}$ en éliminant les sommets dans l'ordre $q_3, q_2, q_1, q_0$, on aurait obtenu

$$(a|b)b^*|a(bb)^*bab^*$$

– L'automate de Glushkov associé à une expression régulière a toujours un nombre d'états de l'ordre du nombre de symboles dans l'expression régulière (mais le déterminiser peut donner un nombre d'états exponentiel), mais l'expression régulière obtenue par l'algorithme de Brzozwski et Mc Cuskey (algorithme n°4, page 37) peut être de taille exponentielle en le nombre d'états de l'automate $\boxed{\text{Exo}}$.