Automates

David Hébert
hebert.iut@gmail.com
Yannick Henrio
yannick.henrio@univ-paris13.fr

2023

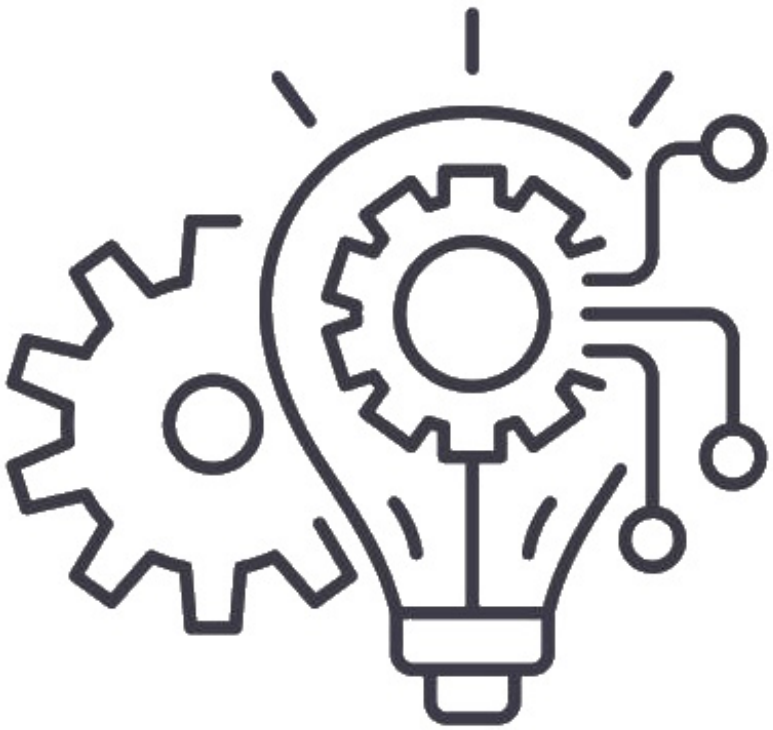


Table des matières

Introduction

Comment votre éditeur de document fait pour vous signaler la présence de faute d'orthographe ? Autrement dis : comment la machine fait pour reconnaitre la langue française ? Il existe plus de trois cent mille mots de la langue française avec toutes les déclinaisons de conjugaison et d'accord. Si on on a un texte de seulement 1000 mots, il faudrait pour vérifier que chaque mot est correctement orthographier faire environ $1\ 000 \times 300\ 000 = 300\ 000\ 000$ tests... trois cent millions de tests... c'est beaucoup (pour seulement un texte de mille mots).

Simplifions le problème et *parlons* en binaire. Nous ne tapons qu'une suite de 0 et 1 et le caractère espace pour espacer ces mots :

11 0 11111 00 1 0 1 0 11 1 0 1

Imaginons que les seuls mots de notre langage binaire (mot du dictionnaire) soit toutes les suites de 0 et de 1 qui ne commencent pas par 0 sauf 0 lui-même. Ainsi dans le texte précédent, il y a une faute : le mot 00 est une faute.

On cherche un processus automatique permettant d'identifier rapidement la forme d'un texte.

Les informaticiens reconnaitrons les REGEX.

```
^[a-zA-Z0-9]+@[a-zA-Z0-9]{2,}\.[a-z]{2,4}
```

1. Langages

1.1 Mots et langages

Définition 1.1.1

On appelle **alphabet** ou **vocabulaire** tout ensemble non vide fini. Les éléments d'un vocabulaire sont appelés **lettres** ou **caractères**.

L'ensemble $\Sigma = \{0, 1\}$ défini l'alphabet du langage binaire et 0 et 1 sont les lettres de cet alphabet.

Définition 1.1.2

Soit Σ un alphabet.

(i) Soit $n \in \mathbb{N}_{>0}$. On appelle **mot** de longueur n tout n-uplet $a = (a_1, \dots, a_n) \in \Sigma^n$. On note plus simplement $a = a_1 \dots a_n$. On note $n = \|a\|$.

(ii) On note ε le mot vide (*i.e.* qui ne contient aucune lettre) ; $\|\varepsilon\| = 0$.

(iii) Pour tout $n \in \mathbb{N}$, on note Σ^n l'ensemble de tous les mots de longueur n ; en particulier $\Sigma^0 = \{\varepsilon\}$ et $\Sigma^1 = \Sigma$.

(iv) On définit l'**étoile de Kleene** sur Σ, notée Σ^*, comme l'ensemble de tous les mots quelque soit leur longueur.

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

Sur l'alphabet $\Sigma = \{0, 1\}$ du langage binaire $\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$ et

$$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, ...\}$$

Si $\Sigma = \{a\}$ alors $\Sigma^* = \{\varepsilon, a, a^2, a^3, a^4, ...\}$.

Exercice 1

Si $\Sigma = \varnothing$, décrire Σ^*.

Exercice 2

Soit Σ un ensemble fini et non vide. Vérifier que l'application $\|\bullet\| : \Sigma^* \to \mathbb{N}$ est surjective. A quelle condition est-elle injective ?

Définition 1.1.3

Soit Σ un alphabet. Un **langage** sur Σ est une partie de L de Σ^*. Les éléments de L sont appelés les **mots du langage**.

Toujours avec $\Sigma = \{0, 1\}$, on considère le langage L formé de tous les mots de Σ^* ne commençant pas par 0 :

$$L = \{\varepsilon, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, ...\}$$

Proposition 1.1.4

Soient L_1 et L_2 deux langages sur un alphabet Σ. Alors $L_1 \cup L_2$, $L_1 \cap L_2$ et $\overline{L_1}$ sont des langages sur Σ.

Démonstration. Triviale □

Exercice 3

Est-ce que $L = \varnothing$ est un langage sur Σ ?

Concaténation

Étant donné un alphabet Σ on peut effectuer sur les langages les opérations ensemblistes courantes : l'intersection, l'union et la complémentation. On peut également concaténer les mots.

Définition 1.1.5

Soient α et β deux mots sur un alphabet Σ tel que $\|\alpha\| = n$ et $\|\beta\| = m$. On définit la **Concaténation** de α de β, noté $\alpha \circ \beta$ ou plus simplement $\alpha\beta$ par la règle :

$$\forall 1 \leqslant k \leqslant n+m, \qquad (\alpha\beta)_k = \begin{cases} \alpha_k & \text{si } k \leqslant n, \\ \beta_{k-n} & \text{sinon.} \end{cases}$$

on dit que α est le **facteur gauche** ou le **préfixe** de $\alpha\beta$ et β et le **facteur droit** ou le **suffixe**.

Sur l'alphabet binaire $001 \circ 101 = 001101$.

Proposition 1.1.6

Pour tout alphabet Σ, la Concaténation définit une opération interne sur Σ^*

$$\begin{array}{rcl} \circ : \Sigma^* \circ \Sigma^* & \longrightarrow & \Sigma^* \\ (\alpha, \beta) & \longmapsto & \alpha \circ \beta \end{array}$$

satisfaisant :

(Associativité) $\forall \alpha, \beta, \gamma \in \Sigma^*$, $(\alpha \circ \beta) \circ \gamma = \alpha \circ (\beta \circ \gamma)$.

(Élément neutre) $\forall \alpha \in \Sigma^*$, $\alpha \circ \varepsilon = \varepsilon \circ \alpha = \alpha$.

(Longueur) $\forall \alpha, \beta \in \Sigma^*$, $\|\alpha \circ \beta\| = \|\alpha\| + \|\beta\|$.

Démonstration. Vérifions l'associativité, le reste est trivial. Notons $\|\alpha\| = n$, $\|\beta\| = m$ et $\|\gamma\| = p$ et fixons un entier $1 \leqslant k \leqslant n+m+p$. Alors

$$((\alpha \circ \beta) \circ \gamma)_k = \begin{cases} (\alpha \circ \beta)_k & 1 \leqslant k \leqslant n+m \\ \gamma_{k-(n+m)} & n+m+1 \leqslant k \leqslant n+m+p \end{cases} = \begin{cases} \alpha_k & 1 \leqslant k \leqslant n \\ \beta_{k-n} & n+1 \leqslant k \leqslant n+m \\ \gamma_{k-(n+m)} & n+m+1 \leqslant k \leqslant n+m+p \end{cases}$$

$$(\alpha \circ (\beta \circ \gamma))_k = \begin{cases} \alpha_k & 1 \leqslant k \leqslant n \\ (\beta \circ \gamma)_{k-n} & n+1 \leqslant k \leqslant n+m+p \end{cases} = \begin{cases} \alpha_k & 1 \leqslant k \leqslant n \\ \beta_{k-n} & n+1 \leqslant k \leqslant n+m \\ \gamma_{(k-n)-m} & n+m+1 \leqslant k \leqslant n+m+p \end{cases}$$

On observe que les deux expressions sont identiques (puisque $k-(n+m) = (k-n)-m$). □

Exercice 4

A quelle condition sur un alphabet fini Σ, la concaténation dans Σ^* est commutative ?

Définition 1.1.7

Soient L_1 et L_2 deux langages sur un alphabet Σ. On défini l'ensemble L_1L_2 le langage des mots concaténés.

$$L_1L_2 = \left\{ m \in \Sigma^* \middle| \exists l_1 \in L_1,\ \exists l_2 \in L_2,\ m = l_1l_2 \right\}$$

Exercice 5

Déterminer $\varnothing L$ et $L\varnothing$ pour tout langage L sur un alphabet Σ.

Proposition 1.1.8

Soient L_1, L_2 et L_3 des langages sur un alphabet Σ.

(i). L_1L_2 est un langage sur Σ.

(ii). $L_1\{\varepsilon\} = \{\varepsilon\}L_1 = L_1$.

(iii). $L_1(L_2L_3) = (L_1L_2)L_3$

Définition 1.1.9

Soit L un langage sur un alphabet Σ. On définit l'**étoile de Kleene** de L, l'ensemble noté L^* définit de manière récursive par les règles

$$\begin{aligned} L^0 &= \{\varepsilon\} \\ \forall n \in \mathbb{N},\ L^{n+1} &= L^nL \\ L^* &= \bigcup_{n\in\mathbb{N}} L^n \end{aligned}$$

Exercice 6

Soit Σ un alphabet non vide tel que $a \in \Sigma$. Décrire L^* dans chacun des cas suivants :

1. $L_1 = \varnothing$
2. $L_2 = \{\varepsilon\}$
3. $L_3 = \{a\}$

1.2 Langages rationnels

Un langage sera appelé *rationnel* s'il découle d'un langage simple (rien, un mot, un ensemble fini de mot etc) et d'opérations sur les langages (union, concaténation, étoile de Kleene). Ce sont de ces langages dont nous allons chercher les REGEX.

Définition 1.2.1

Soit Σ un alphabet. On note $\mathrm{LR}(\Sigma)$ l'ensemble des **langages rationnels** construit de la manière suivante.

$$\begin{aligned} & \varnothing \in \mathrm{LR}(\Sigma) \\ & \{\varepsilon\} \in \mathrm{LR}(\Sigma) \\ \forall x \in \Sigma \quad & \{x\} \in \mathrm{LR}(\Sigma) \\ \forall L_1, L_2 \in \mathrm{LR}(\Sigma) \quad & L_1 \cup L_2 \in \mathrm{LR}(\Sigma) \\ \forall L_1, L_2 \in \mathrm{LR}(\Sigma) \quad & L_1 L_2 \in \mathrm{LR}(\Sigma) \\ \forall L \in \mathrm{LR}(\Sigma) \quad & L^* \in \mathrm{LR}(\Sigma) \end{aligned}$$

Remarque 1.2.2 : Une autre manière de lire cette définition est :

1. L'ensemble vide est un langage rationnel
2. Le mot vide est un langage rationnel
3. Toutes les lettres sont des langages rationnels
4. L'union de langage rationnel est un langage rationnel
5. La concaténation de langage rationnel est un langage rationnel
6. L'étoile de Kleene de langage rationnel est un langage rationnel.

Remarque 1.2.3 : ATTENTION : $\mathrm{LR}(\Sigma)$ est l'ensemble des langages rationnels. C'est donc un ensemble d'ensemble.

Exercice 7

Décrire $\mathrm{LR}(\varnothing)$, $\mathrm{LR}(\{\varnothing\})$ et $\mathrm{LR}(\{a\})$.

Proposition 1.2.4

Soient Σ un alphabet et $m \in \Sigma^*$ un mot alors $\{m\} \in \mathrm{LR}(\Sigma)$

L'Exercice suivant permet de réaliser la démonstration.

Exercice 8

Démontrer la proposition précédente par récurrence sur la taille du mot m.

Proposition 1.2.5

Soient Σ un alphabet et $L \subseteq \Sigma^*$ un langage de cardinalité fini alors $L \in \mathrm{LR}(\Sigma)$

L'Exercice suivant permet de réaliser la démonstration.

Exercice 9

Démontrer la proposition précédente par récurrence sur la cardinalité de L.

Exercice 10

La réciproque de cette proposition est-elle vraie ?

1.3 REGEX

Les REGEX, pour **REGular EXpressions**, sont un ensemble de commande permettant d'effectuer des recherches dans les chaines de caractères. Par exemple, nous avons demandé à un utilisateur d'entrer une adresse mail. Nous voulons nous assurer que la chaine de caractères saisi est au bon format

$$\underbrace{\texttt{hebert.iut}}_{\text{(Des caractères)}}\ \underbrace{\texttt{@}}_{\text{(un arobase)}}\ \underbrace{\texttt{gmail}}_{\text{(au moins 2 caractères)}}\ \underbrace{\texttt{.}}_{\text{(un point)}}\ \underbrace{\texttt{com}}_{\text{(entre 2 et 4 caractères)}}$$

Pour éviter de faire une recherche "à la main" à l'aide de boucle TANT QUE, on peut utiliser la REGEX suivante (en PHP par exemple) [1] :

```
preg_match("#^[a-zA-Z0-9]+@[a-zA-Z0-9]{2,}\.[a-z]{2,4}#",$chaine_saisi)
```

La fonction `preg_match` retourne vrai si la REGEX passé en paramètre apparait dans la chaine du second paramètre. Il existe beaucoup d'autre fonction permettant de jouer avec les REGEX permettant de récupérer ou substituer des informations.

Mémento sur les REGEX.

Ce n'est pas l'objectif de ce cours mais gardons en tête quelques commande.

REGEX	Renvoie `true` si la chaine
`#Euler#`	contient le mot `Euler`
`#Euler|euler#`	contient le mot `Euler` ou `euler`
`#[Ee]uler#`	contient le mot `Euler` ou `euler`
`#^Euler#`	commence par `Euler`
`#Euler$#`	finie par `Euler`
`#^Euler$#`	ne contient que le mot `Euler`
`#[a-z]#`	contient un lettre minuscule
`#[0-9]#`	contient un chiffre
`#[1-3a-gF-H]#`	contient un 1 ou 2 ou 3 ou une lettre minuscule parmi a, b, c, d, e, f, g ou une lettre majuscule parmi F, G, H
`#[^XYZ]#`	ne contient pas X ou Y ou Z
`#[^A-Z]#`	ne contient pas de lettre majuscule
`#^[^a-z]#`	ne commence pas par une lettre minuscule
`#[^Euler]$#`	ne finie pas E ou u ou l ou e ou r
`#Ha{2,4}#`	contient le mot `Haa` ou `Haa` ou `Haaa`
`#(Ha){2,4}#`	contient le mot `HaHa` ou `HaHaHa` ou `HaHaHaHa`
`#Ha+#`	contient le mot qui commence par `H` suivit d'au moins un `a`. Cette REGEX équivaut à `#Ha{1,}#`
`#Ha?#`	contient le mot `H` ou le mot `Ha`. Cette REGEX équivaut à `#Ha{0,1}#`
`#Ha*#`	contient le mot `H` ou `Ha` ou `Haa` ou... . Cette REGEX équivaut à `#Ha{0,}#`
`#Goo*gle#`	contient le mot `Gogle` ou `Google` ou `Gooogle` ...
`#ang?e#`	contient le mot `ange` ou `ane`
`#Allo\?#`	contient le mot `Allo?`

1. Le langage de REGEX utilisé ici est le **BRE** (Basic Regular Expression), il en existe d'autre comme par exemple le **ERE** (Extended Regular Expression), le **PCRE** (Perl Compatible Regular Expressions) fort de sa comptatibilité avec le langage java.

Liens entre langage rationnel et REGEX

L'idée est de déterminer une REGEX qui permet de reconnaitre tous les mots d'un langage rationnel. Comme nous l'avons survoler précédement, une REGEX est un mot sur un alphabet augmenter de quelques opération. Comme on le voit dans le tableau précédent, en plus des caractère alphanumérique de notre alphabet, on utilise un ensemble de caractère spéciaux (le dollar, le crochet, la parenthèse etc).

Dans le cadre théorique dans lequel nous allons aborder ces REGEX, nous allons légèrement simplifier.

Définition 1.3.1

Soit Σ un alphabet. On définit l'alphabet augmenté

$$\Sigma^+ = \Sigma \cup \{\varnothing, \varepsilon, (,), +, *\}$$

Remarque 1.3.2 : Pourquoi ces ajouts et pas d'autres ? L'idée est de trouver une correspondance parfaite entre les REGEX et les langages rationnels.

- L'ajout de $\varnothing$ et ε va permettre d'avoir une REGEX qui correspond aux langages rationnels $\varnothing$ et $\{\varepsilon\}$.
- L'ajout du $+$ correspondra à l'union des langages
- L'ajout de $*$ correspondra à l'étoile de Kleene
- Les parenthèses permettront de prioriser les opération (notamment entre la concaténation (équivalent d'une multiplication) et l'addition).

Définition 1.3.3

Soit Σ un alphabet. On note $\mathsf{RE}(\Sigma)$ l'ensemble des REGEX, mots sur Σ^+, construit de la manière suivante.

$$\begin{aligned}
&\varnothing \in \mathsf{RE}(\Sigma)\\
&\varepsilon \in \mathsf{RE}(\Sigma)\\
\forall x \in \Sigma \quad &x \in \mathsf{RE}(\Sigma)\\
\forall e_1, e_2 \in \mathsf{RE}(\Sigma) \quad &(e_1 + e_2) \in \mathsf{RE}(\Sigma)\\
\forall e_1, e_2 \in \mathsf{RE}(\Sigma) \quad &(e_1 e_2) \in \mathsf{RE}(\Sigma)\\
\forall e \in \mathsf{RE}(\Sigma) \quad &e^* \in \mathsf{RE}(\Sigma)
\end{aligned}$$

Remarque 1.3.4 : Une autre manière de lire cette définition est :

1. L'ensemble vide est une REGEX
2. Le mot vide est une REGEX
3. Toutes les lettres sont des REGEX
4. La somme de REGEX est une REGEX
5. La concaténation de REGEX est une REGEX
6. L'étoile de Kleene d'une REGEX est une REGEX

Remarque 1.3.5 : Attention, si e et f sont des REGEX alors $e + f$ n'est pas une REGEX. En effet, la règle de l'addition dis que la REGEX est $(e + f)$ (les parenthèses). Mais dans la pratique que nous allons en faire, la manipulation répétitive du parenthésage peut devenir très lourde.

Nous convenons donc de nous libérer de cette contrainte lorsqu'il n'y aura pas d'ambiguïté en considérant que la concaténation est prioritaire sur l'addition. Ainsi $ef + g$ aura un sens, celui de $((ef) + g)$ et $e(f + g)$ aura le sens de $(e(f + g))$.

Voici le résultat tant attendu.

Théorème 1.3.6

Soit Σ un alphabet. Il existe une unique application *langage associé*, noté L, satisfaisant les conditions suivantes :

$$\begin{array}{lrcl} & L : RE(\Sigma) & \longrightarrow & LR(\Sigma) \\ & \varnothing & \longmapsto & \varnothing \\ & \varepsilon & \longmapsto & \{\varepsilon\} \\ \forall x \in \Sigma, & x & \longmapsto & \{x\} \\ \forall e_1, e_2 \in RE(\Sigma), & (e_1 + e_2) & \longmapsto & L(e_1) \cup L(e_2) \\ \forall e_1, e_2 \in RE(\Sigma), & (e_1 e_2) & \longmapsto & L(e_1)L(e_2) \in RE(\Sigma) \\ \forall e \in RE(\Sigma), & e^* & \longmapsto & L(e)^* \in RE(\Sigma) \end{array}$$

De plus cette application est surjective.

Démonstration. Admise. □

Remarque 1.3.7 : Une autre manière de lire les règles de la fonction langage associé est :

1. $L(\varnothing) = \varnothing$
2. $L(\varepsilon) = \{\varepsilon\}$
3. $\forall x \in \Sigma,\ L(x) = \{x\}$
4. $\forall e_1, e_2 \in RE(\Sigma),\ L(e_1) \cup L(e_2) = L(e_1 + e_2)$
5. $\forall e_1, e_2 \in RE(\Sigma),\ L(e_1)\, L(e_2) = L(e_1 e_2)$
6. $\forall e \in RE(\Sigma),\ L(e)^* = L(e^*)$

Remarque 1.3.8 : La surjectivité de l'application langage associé se reformule en disant qu'il existe (au moins) une REGEX à tout langage rationnel.

Sur l'alphabet $\Sigma = \{a, b, c\}$ quel est le langage rationnel associé à la REGEX $ab + c$?
La question reviens à calculer $L(ab + c)$, en utilisant les règles de la fonction L on a

$$\begin{aligned} L(ab + c) &= L(ab) \cup L(c) \\ &= L(a)L(b) \cup L(c) \\ &= \{a\}\{b\} \cup \{c\} \\ &= \{ab\} \cup \{c\} \\ &= \{ab, c\} \end{aligned}$$

Exercice 11

On fixe l'alphabet $\Sigma = \{a, b, c\}$. Pour chacune des REGEX suivantes déterminer le langage rationnel.

1. ε
2. a
3. $a + b$
4. $ab + c$
5. $a(b + c)$
6. a^*
7. $(ab)^*$
8. $(a + b)^*$

Exercice 12

On fixe l'alphabet $\Sigma = \{a, b, c\}$. Déterminer, si possible, une REGEX associée aux langages suivantes.

1. $L = \{\varepsilon\}$
2. $L = \{a, b, c\}$
3. $L = \{a, ab, ac\}$
4. $L = \{a^n | n \in \mathbb{N}\}$
5. $L = \{a^{2n} | n \in \mathbb{N}\}$
6. $L = \{b, ab, aab, aaab, aaaab, \ldots\}$
7. $L = \{c, b, ab, aab, aaab, aaaab, \ldots\}$

2. Automates

A présente l'idée est de déterminer un processus *automatique* pour reconnaitre un langage. On ne s'intéressera pas à tous les langages mais seulement aux langages rationnels.

D'après les résultats précédents, il suffira de déterminer un automate reconnaissant une REGEX, c'est à dire un mot sur l'alphabet augmenté.

2.1 Automates d'états finis déterministes

Définition 2.1.1

Un **automate d'état finis déterministe** $\mathcal{A}$, plus simplement appelé ADEF, est la donnée de

- un alphabet $\Sigma_{\mathcal{A}}$, appelé l'**alphabet de l'automate**,
- un ensemble fini $E_{\mathcal{A}}$ d'**état**,
- un singleton $I_{\mathcal{A}} \subset E_{\mathcal{A}}$ d'**état initiale**,
- un ensemble $F_{\mathcal{A}} \subset E_{\mathcal{A}}$ d'**états finaux**,
- une **fonction de transition** $\tau_{\mathcal{A}} : E_{\mathcal{A}} \times \Sigma_{\mathcal{A}} \longrightarrow E_{\mathcal{A}}$.

Considérons l'ADEF ayant $\{0,1\}$ comme alphabet, $\{A,B,C\}$ comme état, $\{A\}$ comme état initial, $\{A,C\}$ comme états finaux et ayant pour fonction de transition la fonction

$$\begin{array}{rcl} \tau : E_{\mathcal{A}} \times \Sigma_{\mathcal{A}} & \longrightarrow & E_{\mathcal{A}} \\ (A,0) & \longmapsto & B \\ (B,1) & \longmapsto & C \\ (B,0) & \longmapsto & B \\ (C,1) & \longmapsto & A \end{array}$$

Représentation d'un automate

On dispose de deux moyens pour représenter un automate. Le premier, la représentation matricielle, est plus satisfaisante pour l'informaticien qui pourra programmer un automate par l'intermédiaire de cette matrice. Le second moyen, la représentation sagittale, va permettre une visualisation simple.

Définition 2.1.2

Soit $\mathcal{A}$ un ADEF. On définit la matrice de l'automate $M_{\mathcal{A}}$ indexée par les états en ligne et l'alphabet en colonne par :

$$(M_{\mathcal{A}})_{e,x} = e' \qquad \text{si } \tau_{\mathcal{A}}(e,x) = e'$$

La matrice suivante est celle de l'automate de l'exemple précédent en précédant (resp. succédant) les états initiaux (resp. finaux) d'une petite flèche.

$$M_{\mathcal{A}} = \begin{array}{c|cc} & 0 & 1 \\ \hline \to A \to & B & \\ B & B & C \\ C \to & & A \end{array}$$

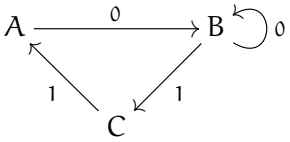
Définition 2.1.3

Soit $\mathcal{A}$ un ADEF. On définit le graphe valué de l'automate $\mathcal{G}_{\mathcal{A}}$ par :

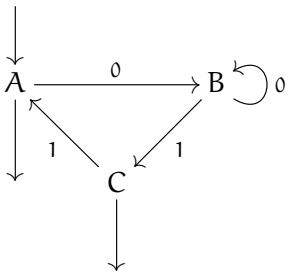
$$\mathbf{Som}(\mathcal{G}_{\mathcal{A}}) = E_{\mathcal{A}}, \qquad \mathbf{Arc}(\mathcal{G}_{\mathcal{A}}) = \left\{ (e,e') \in E_{\mathcal{A}}^2 \middle| \exists x \in \Sigma,\ \tau_{\mathcal{A}}(e,x) = e' \right\}$$

où la valuation est donné par la la fonction de transition.

Le graphe de l'automate de notre exemple est :



On spécifie les états initiaux par des flèches entrantes dans les états concernés et les états finaux par des flèches sortantes



Complétion

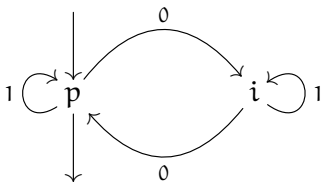
Définition 2.1.4

Un ADEF des appelé **complet** si sa fonction de transition est une application.

Considérons $\Sigma_{\mathcal{A}} = \{0, 1\}$, $E_{\mathcal{A}} = \{p, i\}$, $I_{\mathcal{A}} = \{p\}$, $F_{\mathcal{A}} = \{p\}$ (p pour paire et i pour impaire) et la fonction de transition

$$\begin{aligned} \tau : E_{\mathcal{A}} \times \Sigma_{\mathcal{A}} &\longrightarrow E_{\mathcal{A}} \\ (p, 0) &\longmapsto i \\ (p, 1) &\longmapsto p \\ (i, 0) &\longmapsto p \\ (i, 1) &\longmapsto i \end{aligned}$$

Dans un ADEF complet, tous les coefficients de la matrice sont remplis (la matrice est complète). Le graphe de cet automate est :



Théorème 2.1.5 Complétion d'un automate

Soit $\mathcal{A}$ un automate d'état finis déterministe. Considérons l'automate $\overline{\mathcal{A}}$ défini en rajoutant un nouvel état e tel que :

- Alphabet : $\Sigma_{\overline{\mathcal{A}}} = \Sigma_{\mathcal{A}}$
- États : $E_{\overline{\mathcal{A}}} = E_{\mathcal{A}} \cup \{e\}$
- États initiaux : $I_{\overline{\mathcal{A}}} = I_{\mathcal{A}}$
- États finaux : $F_{\overline{\mathcal{A}}} = F_{\mathcal{A}}$
- Fonction de transition :

$$\begin{aligned} \tau_{\overline{\mathcal{A}}} : E_{\overline{\mathcal{A}}} \times \Sigma_{\overline{\mathcal{A}}} &\longrightarrow E_{\overline{\mathcal{A}}} \\ (x, \sigma) &\longmapsto \begin{cases} \tau_{\mathcal{A}}(x, \sigma) & \text{si c'est défini} \\ e & \text{sinon} \end{cases} \end{aligned}$$

Alors $\overline{\mathcal{A}}$, appelé le **complété** de $\mathcal{A}$, est un automate d'état finis déterministe complet.

Démonstration. C'est une conséquence triviale de la construction de $\overline{\mathcal{A}}$. □

Considérons l'ADEF (non complet) $\mathcal{A}$ suivant :

- $\Sigma_{\mathcal{A}} = \{0, 1\}$
- $E_{\mathcal{A}} = \{A, B, C\}$
- $I_{\mathcal{A}} = \{A, B\}$
- $F_{\mathcal{A}} = \{A, C\}$
- Fonction de transition :

$$\begin{array}{rcl} \tau : E_{\mathcal{A}} \times \Sigma_{\mathcal{A}} & \longrightarrow & E_{\mathcal{A}} \\ (A, 0) & \longmapsto & B \\ (B, 1) & \longmapsto & C \\ (B, 0) & \longmapsto & B \\ (C, 1) & \longmapsto & A \end{array}$$

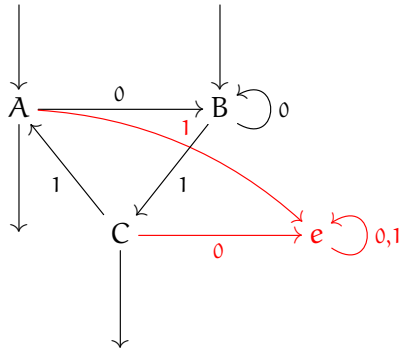
Le complété de cette automate est :

- $\Sigma_{\overline{\mathcal{A}}} = \{0, 1\}$
- $E_{\overline{\mathcal{A}}} = \{A, B, C, e\}$
- $I_{\overline{\mathcal{A}}} = \{A, B\}$
- $F_{\overline{\mathcal{A}}} = \{A, C\}$
- Fonction de transition :

$$\begin{array}{rcl} \tau : E_{\overline{\mathcal{A}}} \times \Sigma_{\overline{\mathcal{A}}} & \longrightarrow & E_{\overline{\mathcal{A}}} \\ (A, 0) & \longmapsto & B \\ (A, 1) & \longmapsto & C \\ (B, 0) & \longmapsto & B \\ (B, 1) & \longmapsto & e \\ (C, 0) & \longmapsto & e \\ (C, 1) & \longmapsto & A \\ (e, 0) & \longmapsto & e \\ (e, 1) & \longmapsto & e \end{array}$$

$M_{\overline{\mathcal{A}}} =$

	0	1
A	B	e
B	B	C
C	e	A
e	e	e



Exercice 13

Pour chacun des automates suivants identifier l'alphabet, l'état initiale, les états finaux et donner la représentation sagittale. Compléter l'automate.

1.

	a	b
A	E	C
B →	E	A
C	C	A
D	A	E
→ E	E	B

2.

	u	v
A	A	
B →		
C	B	B
D	C	D
E	D	
→ F	E	D

3.

	0	1	2
A		D	D
B		D	E
→ C →	B	B	E
D	D	C	C
E	C	D	A

2.2 Automates d'états finis non déterministes

Définition 2.2.1

Un **automate d'état finis** $\mathcal{A}$, plus simplement appelé AEF, est la donnée de

- un alphabet $\Sigma_{\mathcal{A}}$, appelé l'**alphabet de l'automate**,
- un ensemble fini $E_{\mathcal{A}}$ d'**état**,
- un ensemble $I_{\mathcal{A}} \subset E_{\mathcal{A}}$ d'**états initiaux**,
- un ensemble $F_{\mathcal{A}} \subset E_{\mathcal{A}}$ d'**états finaux**,
- une **fonction de transition** $\tau_{\mathcal{A}} : E_{\mathcal{A}} \times (\Sigma_{\mathcal{A}} \cup \{\varepsilon\}) \longrightarrow \mathcal{P}(E_{\mathcal{A}})$.

La différence entre un ADEF et un AEF est que les passages d'un état à un autre ne sont uniquement déterminés par l'alphabet. Un même caractère peut mener à différents états.

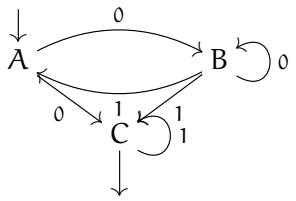
Considérons l'AEF sur l'alphabet $\Sigma = \{0, 1\}$ ayant comme état $\{A, B, C\}$, $\{A\}$ comme état initial, $\{C\}$ comme état final et ayant pour fonction suivante pour fonction de transition :

$$\begin{array}{rcl} \tau : E_{\mathcal{A}} \times \Sigma_{\mathcal{A}} & \longrightarrow & \mathcal{P}(E_{\mathcal{A}}) \\ (A, 0) & \longmapsto & \{B, C\} \\ (B, 0) & \longmapsto & \{B\} \\ (B, 1) & \longmapsto & \{A, C\} \\ (C, 1) & \longmapsto & \{C\} \end{array}$$

De la même manière on représente un automate de deux manières : matricielle et sagittale.

Dans notre exemple nous avons :

$$M_{\mathcal{A}} = \begin{array}{c|cc} & 0 & 1 \\ \hline \to A & B, C & \\ B & B & A, C \\ C \to & & C \end{array}$$



Proposition 2.2.2

Tout ADEF est un AEF.

Démonstration. On compose la fonction de transition d'un ADEF par le plongement $X \hookrightarrow \mathcal{P}(X)$. □

Il est également possible de rendre un AEF déterministe.

Proposition 2.2.3 Détermination d'un AEF

Soit $\mathcal{A}$ un AEF. Considérons l'automate $\mathcal{A}_{\text{det}}$ défini par les règles suivantes :

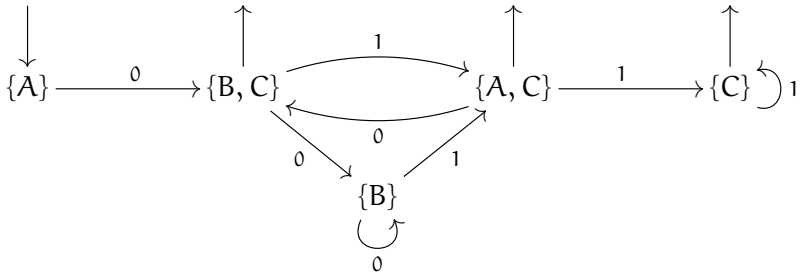
- $\Sigma_{\mathcal{A}_{\text{det}}} = \Sigma_{\mathcal{A}}$
- $E_{\mathcal{A}_{\text{det}}} = \mathcal{P}(E_{\mathcal{A}}) - \{\varnothing\}$
- $I_{\mathcal{A}_{\text{det}}} = \{I_{\mathcal{A}}\}$
- $F_{\mathcal{A}_{\text{det}}} = \left\{ X \subset E_{\mathcal{A}} \middle| \exists f \in F_{\mathcal{A}}, f \in X \right\}$
- $$\begin{array}{rcl} \tau_{\mathcal{A}_{\text{det}}} : E_{\mathcal{A}_{\text{det}}} \times \Sigma_{\mathcal{A}_{\text{det}}} & \longrightarrow & E_{\mathcal{A}_{\text{det}}} \\ (X, \sigma) & \longmapsto & \bigcup_{x \in X} \tau_{\mathcal{A}}(x, \sigma) \end{array}$$

Alors $\mathcal{A}_{\text{det}}$ est un automate déterministe.

Démonstration. Il suffit de voir que la fonction de transition ainsi défini est bien une fonction ce qui se déduit trivialement de la construction. □

Remarque 2.2.4 : Si l'AEF est un automate à n états alors l'ADEF associé a $2^n - 1$ états. Cet automate est donc exponentiellement plus grand que le premier. Cependant il conserve un avantage qui sera résumé plus tard à travers le théorème de Rabin-Scott.

L'ADEF correspondant à la détermination d'un AEF possède parfois de nombreux états inaccessibles comme nous pouvons le voir sur l'exemple précédent dont l'ADEF associé est :



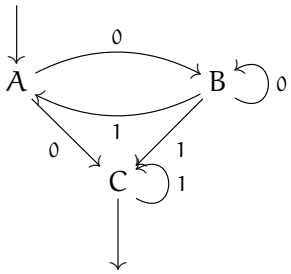
Les ensembles $\{A, B\}$ et $\{A, B, C\}$ ne sont pas utilisés.

L'algorithme suivant permet de construire progressivement l'ADEF associé à un AEF.

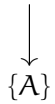
§2.2.5 : `Détermination d'un AEF`

`Initialiser` $E_{\mathcal{A}_{det}}$ `à` $\{I_{\mathcal{A}}\}$
`Tant que` $E_{\mathcal{A}_{det}}$ `contient des éléments non marqué`
  `Sélectionner un élément` X `de` $E_{\mathcal{A}_{det}}$ `non marqué.`
  `Marquer` X.
  `Pour chaque` $\sigma \in \Sigma_{\mathcal{A}_{det}}$
    `Considérer` $X' = \bigcup_{x \in X} \tau_{\mathcal{A}}(x, \sigma)$.
    `Ajouter` X' `à` $E_{\mathcal{A}_{det}}$ `si ce n'est pas déjà le cas.`
    `Ajouter un arc entre` X `et` X' `valué par` σ.
  `Fin Pour`
`Fin Tant Que`

Faisons tourner cet algorithme sur l'AEF

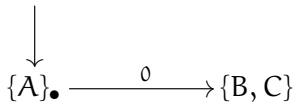


Initialisation.

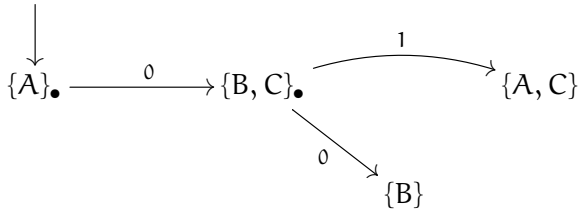


On choisis $\{A\}$**.** On le marque (en mettant un petit point par exemple). Pour chaque élément composant cet ensemble (ici uniquement A), on construit l'image par 0 par $\tau_{\mathcal{A}} : \{B, C\}$. Cet ensemble n'existant pas on le rajoute et on met un arc entre $\{A\}$ et $\{B, C\}$ valué par 0. On fait de même pour 1 mais il

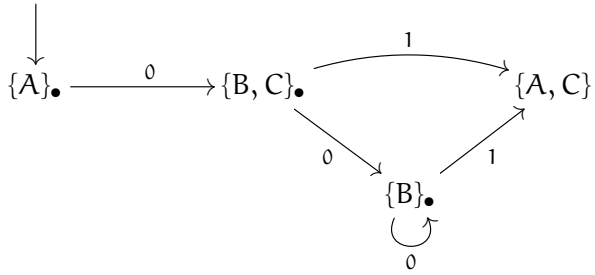
n'y a pas d'image. On arrive donc à



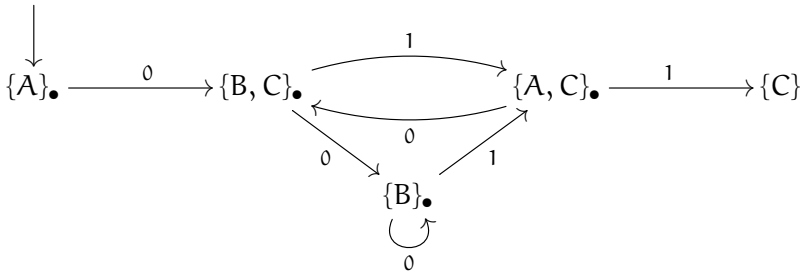
On choisis {B, C}**.** On le marque. Pour chaque élément composant cet ensemble (à savoir B et C), on construit l'image par 0 par $\tau_{\mathcal{A}}$: B donne B et C ne donne rien. On arrive donc à l'ensemble {B}. Cet ensemble n'existant pas on le rajoute et on met un arc entre {B, C} et {B} valué par 0. On fait de même pour 1 : B donne A et C et C donne C, il s'agit donc de l'ensemble {A, C}. Au final :



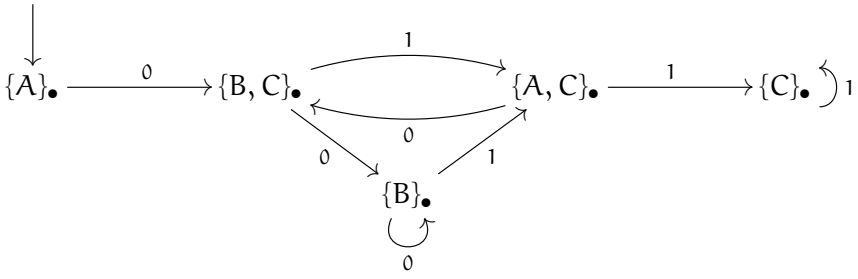
On choisis {B}**.** (on aurait également pu choisir {A, C}). On le marque. On construit l'image de B par 0 via $\tau_{\mathcal{A}}$: {B}. Par 1 on arrive {A, C}



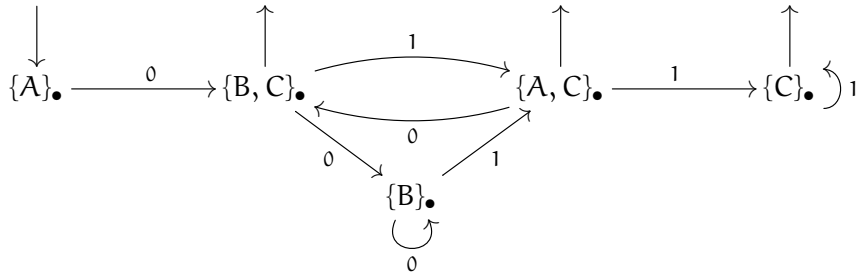
On choisis {A, C}**.** On le marque. On construit l'image de A et C par 0 via $\tau_{\mathcal{A}}$: {B, C}. Par 1 on arrive {C}



On choisis {C}**.** On le marque. L'image de C par 0 via $\tau_{\mathcal{A}}$ n'existe pas et l'image de C par 1 via $\tau_{\mathcal{A}}$ donne {C}



FIN. Tous les sommets sont marqués. C'est fini. On rajoute les états de sortie dès qu'un éléments d'un sommet possède un élément de $F_{\mathcal{A}}$; ici il n'y a que C.



Exercice 14

Pour chacun des automates suivants, donner une représentation sagittale et construire l'automate déterministe associé.

1.

	0	1
A →	C	C
→ B	A, B, C	B
C	B, C	A, C

2.

	0	1
A →	A	C
B	B, C	A
→ C	A, B, C	B

3.

	0	1
A	B, C	A
→ B	B	A
C →	A	B

4.

	0	1
→ A	B	C
B	C	C
C →	A	B, C

5.

	0	1
→ A →	C	A
B	C	C
C	A	A, B

6.

	0	1
→ A	C	B
B →	C	A, C
C	A	A

7.

	a	b	c
A	C	D	D
B →	A, D	A, B, C, D	A, B, D
C	A, B, C, D	C	A
→ D	A	A	A

8.

	a	b	c
→ A →	C	A, C, D	D
B	A, B	C	A, B, C, D
C	D	C	B, C
D	B	A, B, D	B

2.3 Automates d'états finis non déterministes à ε-transition

Nous allons à présent assouplir davantage la notion d'automate en permettant des transitions par le mot vide.

Définition 2.3.1

Un **automate d'état finis à ε-transition** $\mathcal{A}$, plus simplement appelé AEFε, est la donnée de

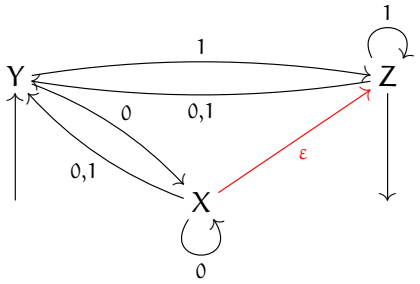
- un alphabet $\Sigma_{\mathcal{A}}$, appelé l'**alphabet de l'automate**,
- un ensemble fini $E_{\mathcal{A}}$ d'**état**,
- un ensemble $I_{\mathcal{A}} \subset E_{\mathcal{A}}$ d'**états initiaux**,
- un ensemble $F_{\mathcal{A}} \subset E_{\mathcal{A}}$ d'**états finaux**,
- une **fonction de transition** $\tau_{\mathcal{A}} : E_{\mathcal{A}} \times \Sigma_{\mathcal{A}} \cup \varepsilon \longrightarrow \mathcal{E}_{\mathcal{A}}$.

En définitive, c'est exactement la même définition que pour un automate à ceci près que la fonction de transition, permet des transitions par le mot ε.

Comme pour les ADEF et les AEF, on représente les AEFε par une matrice ou un graphe augmenté.

$M_{\mathcal{A}} =$

	0	1	ε
X	X, Y	Y	Z
→ Y	X	Z	
Z →	Y	Y, Z	



Proposition 2.3.2

Tout AEF est un AEFε.

Démonstration. Trivial. □

L'idée à présent est de supprimer les ε-transition (pour qu'il n'y ai pas d'ambiguïté sur les mots reconnus - nous détaillerons ce point au prochain chapitre) et donc de transformer un AEFε en AEF.

Définition 2.3.3

Soit $\mathcal{A}$ une AEFε et X un état de $\mathcal{A}$. On définit $\overline{X}$, la **ε-clôture** de X comme l'ensemble des états qui *lisent* le mot vide.

$$\overline{X} = \left\{ Y \in E_{\mathcal{A}} \middle| \tau_{\mathcal{A}}(X, \varepsilon) = Y \right\}$$

La notion de *lecture* sera détailler dans le prochain chapitre.

Avec l'automate précédent, on a

$$\overline{X} = \{X, Z\} \qquad \overline{Y} = \{Y\} \qquad \overline{Z} = \{Z\}$$

Définition 2.3.4

Soit $\mathcal{A}$ un AEFε. L'automate **ε-libéré** $\mathcal{A}_\varepsilon$ est l'AEF définit par les règles suivantes :

- $\Sigma_{\mathcal{A}_\varepsilon} = \Sigma_{\mathcal{A}}$
- $E_{\mathcal{A}_\varepsilon} = E_{\mathcal{A}}$
- $I_{\mathcal{A}_\varepsilon} = I_{\mathcal{A}}$
- $F_{\mathcal{A}_\varepsilon} = \left\{ X \in E_{\mathcal{A}_\varepsilon} \middle| \overline{X} \cap F_{\mathcal{A}} \neq \varnothing \right\}$
- La fonction de transition

$$\begin{array}{rccc} \tau_{\mathcal{A}_\varepsilon} : & E_{\mathcal{A}_\varepsilon} \times \Sigma_{\mathcal{A}_\varepsilon} & \longrightarrow & \mathcal{P}(E_{\mathcal{A}_\varepsilon}) \\ & (X, x) & \longmapsto & \bigcup_{Y \in \overline{X}} \tau_{\mathcal{A}}(Y, x) \end{array}$$

Avec l'automate précédent on a

$$M_{\mathcal{A}_\varepsilon} = \begin{array}{c|cc} & 0 & 1 \\ \hline X \to & X, Y & Y, Z \\ \to Y & X & Z \\ Z \to & Y & Y, Z \end{array}$$

Exercice 15

Déterminer l'automate ε-libéré de chacun des AEFε suivant.

1.

	x	y	ε
A	A	A, B, C	
→ B →	A	A, C	
C	A, B, C	A, B	A

2.

	x	y	ε
A →	A, C	A, C	A, C
B	A, B, C, D	A, C, D	
→ C	A, B, C, D	C, D	
D	D	A, B	C

3.

	α	β	γ	ε
A	A, C	B	A, C	
B →	A, B, C	B, C	C, D	B, D
→ C	A, C	D	A, C, D	
D	A, C, D	D	B, C	

4.

	0	1	2	ε
A	C	A, B, C	C	A, C
→ B →	B, C	A, B, C	B	B
C	A, B, C	C	A	B

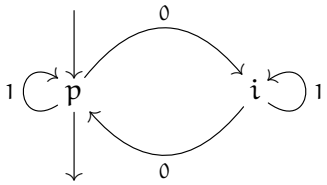
2.4 Langage d'un automate

Définition 2.4.1

Soient $\mathcal{A}$ un AEFε, $\sigma \in \Sigma_{\mathcal{A}}^*$ tel que $\|\sigma\| = n$. On dira que σ est **accepté** par $\mathcal{A}$ si il existe une suite $e_0, e_1, \ldots, e_n$ de $E_{\mathcal{A}}$ tel que

1. $e_0 \in I_{\mathcal{A}}$,
2. $e_n \in F_{\mathcal{A}}$,
3. $\forall k \in [\![1, n]\!],\ \tau_{\mathcal{A}}(e_{k-1}, \sigma_k) = e_k$

Autrement dit : un mot de longueur n est accepté par un automate s'il existe un chaine de longueur $n-1$ dans le graphe représentant l'automate ayant un état initial pour origine et un état final pour aboutissement.

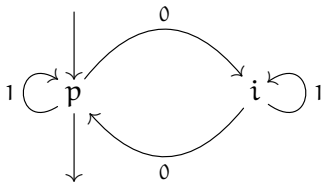


Les mots ε, 001, 100, 1100, 010 sont acceptés par l'automate ci-contre.

Définition 2.4.2

Le **langage** d'un AEFε $\mathcal{A}$ est l'ensemble des mots acceptés.

$$L(\mathcal{A}) = \left\{ \sigma \in \Sigma_{\mathcal{A}}^* \middle| \sigma \text{ est accepté par } \mathcal{A} \right\}$$

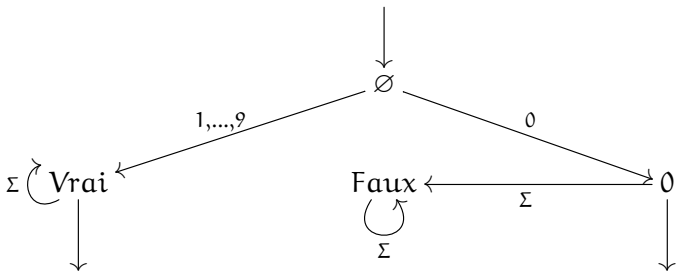


Les mots reconnus par cet automate sont tous les mots de $\{0,1\}$ possédant un nombre paire de 0.

Définition 2.4.3

Soient Σ un alphabet et $L \subset \Sigma^*$ un langage. On dira que L est un **langage d'états finis**, s'il existe un ADEF $\mathcal{A}$ tel que $\Sigma = \Sigma_{\mathcal{A}}$ et $L(\mathcal{A}) = L$. On dit que l'automate $\mathcal{A}$ **reconnait** le langage L.

Considérons l'alphabet $\Sigma = \{0,1,2,3,4,5,6,7,8,9\}$ et le langage $L = \mathbb{N}$ des nombres. Il s'agit d'un langage d'états finis. En effet les nombres (ne commençant pas par 0 sauf 0 lui même) sont reconnus par l'A(D)EF suivant :



Théorème 2.4.4 Rabin-Scott

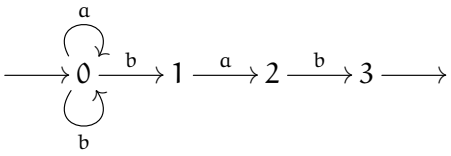
Soit $\mathcal{A}$ un AEF.

$$L(\mathcal{A}) = L(\mathcal{A}_{\text{det}})$$

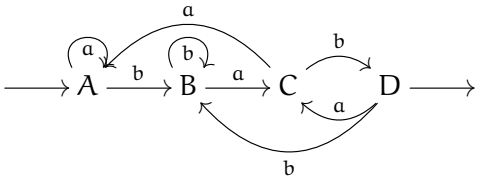
Démonstration. Admise □

Il est parfois plus naturel de construire une AEF qui traduit un langage puis de le déterminer (et le compléter) pour pouvoir le programmer.

Considérons l'AEF qui reconnait les mots de l'alphabet $\Sigma = \{a, b\}$ qui se termine par bab :



Le déterminé (complété) de cet AEF est



Théorème 2.4.5

Soit $\mathcal{A}$ un AEFε.

$$L(\mathcal{A}) = L(\mathcal{A}_\varepsilon)$$

Démonstration. Admise. □

Exercice 16

Décrire un automate déterministe et complet permettant de reconnaitre les langages suivants.

1. Les mots finissant par bab sur l'alphabet $\Sigma = \{a, b\}$.
2. Les mots commençant par aba sur l'alphabet $\Sigma = \{a, b\}$.
3. Les mots ne contenant pas de 00 sur l'alphabet $\Sigma = \{0, 1\}$.
4. Les nombres entiers ($\mathbb{N}$) sur $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
5. Les nombres entiers paires ($2\mathbb{N}$) sur $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
6. Les nombres entiers divisible par 3 ($3\mathbb{N}$) sur $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
7. Les entiers relatifs ($\mathbb{Z}$) sur l'alphabet $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -\}$.
8. Les expressions mathématiques correctes sur l'alphabet $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -\}$.

3.1 Lemme d'Arden

Théorème 3.1.1 Lemme d'Arden

Soient A et B deux langages sur un alphabet Σ.

(i) Une solution de l'équation $L = AL \cup B$ est $L = A^*B$.

(ii) Si $\varepsilon \notin A$, $L = A^*B$ est l'unique solution de l'équation $L = AL \cup L$

Démonstration. Admise □

Ce théorème est très puissant et va permettre de déterminer le langage reconnu par un automate.

Définition 3.1.2

Soient i et j des états d'un ADEF $\mathcal{A}$ et L un langage sur $\Sigma_{\mathcal{A}}$. On note

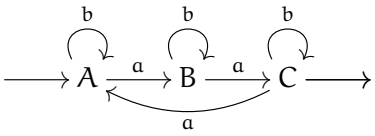
$\Sigma_{i,j}$ l'ensemble des lettres de l'alphabet qui permettent de passer de l'état i à l'état j.

$$\Sigma_{i,j} = \left\{ x \in \Sigma \middle| \tau_{\mathcal{A}}(i,x) = j \right\}$$

$\mathcal{A}_i$ l'automate identique à l'automate $\mathcal{A}$ où l'état initial a été remplacé par i.

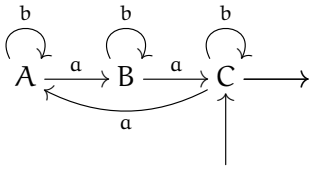
$$\sigma_{\mathcal{A}_i} = \Sigma_{\mathcal{A}},\ E_{\mathcal{A}_i} = E_{\mathcal{A}},\ I_{\mathcal{A}_i} = \{i\},\ F_{\mathcal{A}_i} = F_{\mathcal{A}},\ \tau_{\mathcal{A}_i} = \tau_{\mathcal{A}}$$

Considérons l'automate



On a $\Sigma_{A,A} = \{b\}$, $\Sigma_{A,B} = \{a\}$, $\Sigma_{A,C} = \varnothing$.

De même $\mathcal{A}_A = \mathcal{A}$ et $\mathcal{A}_C$ est l'automate



Proposition 3.1.3

Soit $\mathcal{A}$ un automate.

$$\forall i \in E_{\mathcal{A}} - F_{\mathcal{A}} \quad L(\mathcal{A}_i) = \bigcup_{j \in E_{\mathcal{A}}} \Sigma_{i,j} L(\mathcal{A}_j)$$

$$\forall i \in F_{\mathcal{A}} \quad L(\mathcal{A}_i) = \{\varepsilon\} \cup \bigcup_{j \in E_{\mathcal{A}}} \Sigma_{i,j} L(\mathcal{A}_j)$$

Démonstration. Notons $L_i = L(\mathcal{A}_i)$.

On observe pour commencer que si ε est un mot reconnu alors l'état initial est un état final. En particulier $\varepsilon \in L_f$ si f est un état final, la réciproque est trivialement vrai. Ainsi pour démontrer les deux cas de la proposition, il suffit de montrer que $L_i - \{\varepsilon\} = \bigcup_{j \in E_{\mathcal{A}}} \Sigma_{i,j} L_j$.

$\bigcup_{j\in E_{\mathcal{A}}} \Sigma_{i,j}L_j \subseteq L_i - \{\varepsilon\}$. Si $a \in \Sigma_{i,j}$ et $m \in L_j$ alors $am \in L_i$ donc $\Sigma_{i,j}L_j \subset L_j$ et à fortiori $\bigcup_{j\in E_{\mathcal{A}}} \Sigma_{i,j}L_j \subseteq L_i - \{\varepsilon\}$.

$L_i - \{\varepsilon\} \subseteq \bigcup_{j\in E_{\mathcal{A}}} \Sigma_{i,j}L_j$. Soit $m \in L_i - \{\varepsilon\}$ et $a \in \Sigma_{\mathcal{A}}$ la première lettre de m : $m = am'$. Posons $j = \tau_{\mathcal{A}}(i, a)$ alors $m' \in L_j$ ainsi $am = \Sigma_{i,j}L_j$ ce qui prouve l'inclusion.

□

Reprenons l'automate précédent. Notons $L_X = L(\mathcal{A}_X)$.

$$L_A = \Sigma_{A,A}L_A \cup \Sigma_{A,B}L_B \cup \Sigma_{A,C}L_C$$

Nous avons $\Sigma_{A,A} = \{b\}$, $\Sigma_{A,B} = \{a\}$ et $\Sigma_{A,C} = \varnothing$. Sachant que pour tout langage L on a $\varnothing L = \varnothing$, l'égalité précédente se simplifie en

$$L_A = \{b\}L_A \cup \{a\}L_B$$

Le lemme d'Arden permet alors de trouver L_A : $L_A = \{b\}^*\{a\}L_B$

De la même manière, on a $L_B = \{b\}L_B \cup \{a\}L_C$ qui permet, par le lemme d'Arden, d'obtenir $L_B = \{b\}^*\{a\}L_C$ ce qui donne en jumelant avec l'égalité précédente :

$$L_A = \{b\}^*\{a\}\left(\{b\}^*\{a\}L_C\right) = \left(\{b\}^*\{a\}\right)^2 L_C$$

Enfin, puisque C est final, on a $L_C = \{\varepsilon\} \cup \{b\}L_C \cup \{a\}L_A$ ce qui donne par le lemme d'Arden

$$\begin{aligned} L_C &= \{b\}^*\left(\{\varepsilon\} \cup \{a\}L_A\right) \\ &= \{b\}^*\{\varepsilon\} \cup \{b\}^*\{a\}L_A \\ &= \{b\}^* \cup \{b\}^*\{a\}L_A \end{aligned}$$

En jumelant nos deux dernières égalités nous arrivons a

$$\begin{aligned} L_A &= \left(\{b\}^*\{a\}\right)^2 L_C \\ &= \left(\{b\}^*\{a\}\right)^2 \left(\{b\}^* \cup \{b\}^*\{a\}L_A\right) \\ &= \left(\left(\{b\}^*\{a\}\right)^2\{b\}^*\right) \cup \left(\left(\{b\}^*\{a\}\right)^2\{b\}^*\{a\}L_A\right) \\ &= \left(\left(\{b\}^*\{a\}\right)^2\{b\}^*\right) \cup \left(\left(\{b\}^*\{a\}\right)^3 L_A\right) \end{aligned}$$

Encore un petit coup de lemme d'Arden pour pouvoir conclure (puisque $L = L_A$) :

$$L = \left(\left(\{b\}^*\{a\}\right)^3\right)^* \left(\left(\{b\}^*\{a\}\right)^2\{b\}^*\right)$$

De plus, il est évident que la REGEX associée est $\left((b^*a)^3\right)^*\left((b^*a)^2 b^*\right)$.

Exercice 17

Pour chacun des automates suivants, calculer une REGEX reconnaissant le langage de l'automate.

1.

	α	β
A	B	C
B $\to$	C	C
$\to$ C	B	C

2.

	0	1
A	B	A
$\to$ B	A	C
C $\to$	C	A

3.

	u	v
$\to$ A	A	A
B $\to$	B	A
C	B	A

4.

	u	v
$\to$ A	C	A
B $\to$	B	A
C	B	A

5.

	u	v
$\to$ A	C	A
B $\to$	B	A
C	B	A

6.

	x	y
A	C	A, B, C
$\to$ B	B	A, C
C $\to$	B	B, C

3.2 Théorème de Kleene

Dans le précédent paragraphe, nous avons à chaque automate, associé un langage : c'est le langage reconnu par cet automate.

Nous nous intéressons à la réciproque. Dans notre contexte, on se limitera au langage rationnel permettant de travailler avec les REGEX.

Définition 3.2.1

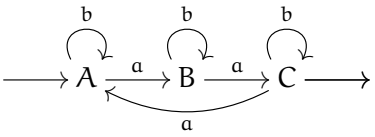
On dira qu'un automate $\mathcal{A}$ est un **automate rationnel** si $L(\mathcal{A}) \in LR(\Sigma_{\mathcal{A}})$.

Théorème 3.2.2 Kleene

Un langage est rationnel si et seulement si il existe un automate rationnel qui le reconnait.

Démonstration. Admise. □

Considérons par exemple l'ADEF $\mathcal{A}$ suivant



Nous avons montré, à l'aide du lemme d'Arden, que la REGEX qui reconnait ce langage est $\left((b^*a)^3\right)^*(b^*a)^2b^*$.

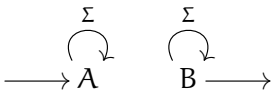
A présent la problématique est inverse. On dispose d'une REGEX r qui, par l'intermédiaire de l'application L (langage associé), donne un langage relationnel $L(r)$. D'après le théorème de Kleene, nous devrions trouver un automate (rationnel) permettant de reconnaitre ce langage.

Nous allons prendre chacune des conditions construisant l'application *langage associé* et déterminer des algorithmes permettant de construire un automate reconnaissant le langage.

3.3 Condition 1 : automate reconnaissant $\varnothing$

Le premier critère de construction de l'application *langage associé* est que $L(\varnothing) = \varnothing$. Déterminons donc un automate qui permet de reconnaitre le langage $\varnothing$ sur un alphabet quelconque Σ.

Considérons l'automate suivant



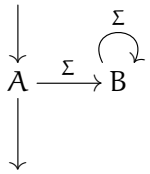
	Σ
→ A	A
B →	B

Exercice 18

En utilisant le lemme d'Arden, calculer le langage reconnu par cet automate ainsi qu'une REGEX associée.

3.4 Condition 2 : automate reconnaissant $\{\varepsilon\}$

Le second critère de construction de l'application *langage associé* est que $L(\varepsilon) = \{\varepsilon\}$.
Considérons l'automate suivant pour un alphabet quelconque Σ.



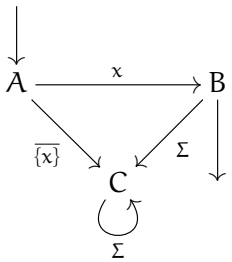
	Σ
→ A →	B
B	B

Exercice 19

En utilisant le lemme d'Arden, calculer le langage reconnu par cet automate ainsi qu'une REGEX associée.

3.5 Condition 3 : automate reconnaissant un caractère

Le troisième critère de construction de l'application *langage associé* est que $L(x) = \{x\}$ pour n'importe quel caractère x d'un alphabet Σ.
Considérons l'automate suivant



	{x}	$\overline{\{x\}}$
→ A	B	C
B →	C	C
C	C	C

Exercice 20

En utilisant le lemme d'Arden, calculer le langage reconnu par cet automate ainsi qu'une REGEX associée.

3.6 Condition 4 : automate reconnaissant une union

Définition 3.6.1

Soient $\mathcal{A}$ et $\mathcal{B}$ des automates sur un langage Σ. On définit $\mathcal{A}+\mathcal{B}$ comme l'automate avec les règles suivantes.

Alphabet. $\Sigma_{\mathcal{A}+\mathcal{B}} = \Sigma$

États. $E_{\mathcal{A}+\mathcal{B}} = E_{\mathcal{A}} \times E_{\mathcal{B}}$

État initial. $I_{\mathcal{A}+\mathcal{B}} = I_{\mathcal{A}} \times I_{\mathcal{B}}$

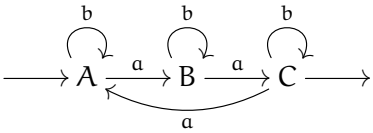
États finaux. $F_{\mathcal{A}+\mathcal{B}} = (F_{\mathcal{A}} \times E_{\mathcal{B}}) \cup (E_{\mathcal{A}} \times F_{\mathcal{B}})$

Transitions. $\forall (A,B) \in E_{\mathcal{A}+\mathcal{B}}, \forall x \in \Sigma, \tau_{\mathcal{A}+\mathcal{B}}((A,B),x) = (\tau_{\mathcal{A}}(A,x), \tau_{\mathcal{B}}(B,x))$.

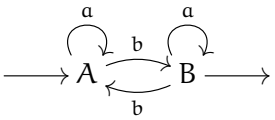
Exercice 21

Donner la représentation sagittale de l'automate $\mathcal{A}+\mathcal{B}$ sur $\Sigma = \{a,b\}$ où

L'automate $\mathcal{A}$ est



L'automate $\mathcal{B}$ est



Proposition 3.6.2

Soient $\mathcal{A}$ et $\mathcal{B}$ des automates sur un langage Σ.

$$L(\mathcal{A}+\mathcal{B}) = L(\mathcal{A}) \cup L(\mathcal{B})$$

Démonstration. C'est une conséquence triviale de la définition. □

Exercice 22

Soit $\Sigma = \{a,b\}$.

1. Donner un automate reconnaissant les langages $\{a\}$ et $\{b\}$.
2. Donner un automate qui reconnait le langage Σ. Donnez une REGEX reconnaissant ce langage.

3.7 Condition 5 : automate reconnaissant une concaténation

Définition 3.7.1

Soient $\mathcal{A}$ et $\mathcal{B}$ deux automates sur un alphabet Σ où l'on renomme les états de $\mathcal{B}$ de sorte que $E_{\mathcal{A}} \cap E_{\mathcal{B}} = \varnothing$. On définit $\mathcal{AB}$ l'automate à ε-transition, par les règles suivantes.

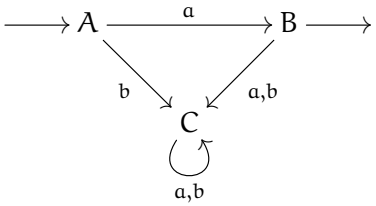
- $\Sigma_{\mathcal{AB}} = \Sigma$
- $E_{\mathcal{AB}} = E_{\mathcal{A}} \cup E_{\mathcal{B}}$
- $I_{\mathcal{AB}} = I_{\mathcal{A}}$
- $F_{\mathcal{AB}} = F_{\mathcal{B}}$
- La fonction de transition

$$\begin{aligned} \tau_{\mathcal{AB}} : E_{\mathcal{AB}} \times \Sigma \cup \varepsilon &\longrightarrow \mathcal{P}(E_{\mathcal{AB}}) \\ \forall A \in E_{\mathcal{A}}, \forall x \in \Sigma,\ (A,x) &\longmapsto \tau_{\mathcal{A}}(A,x) \\ \forall B \in E_{\mathcal{B}}, \forall x \in \Sigma,\ (B,x) &\longmapsto \tau_{\mathcal{B}}(B,x) \\ \forall f \in F_{\mathcal{A}}, \forall i \in I_{\mathcal{B}},\ (f,\varepsilon) &\longmapsto i \end{aligned}$$

En définitive, on *connecte*, par une ε-transition, les deux automates que l'on souhaite concaténer.

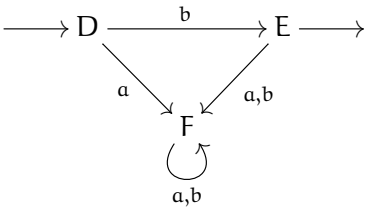
Sur l'alphabet $\Sigma = \{a, b\}$, considérons les automates reconnaissant chacun des caractères.

Automate reconnaissant $\{a\}$:



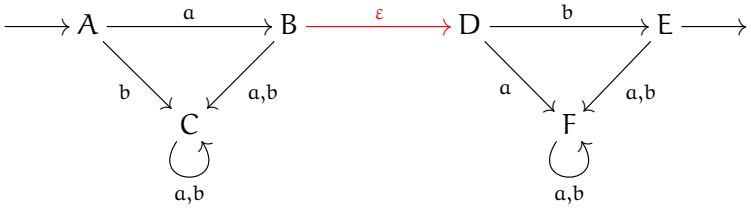
	a	b
→ A	B	C
B →	C	C
C	C	C

Automate reconnaissant $\{b\}$; c'est le même que précédemment en changeant le rôle de a et b et en prenant soin de nommer les états différemment de ceux de l'automate précédent :



	a	b
→ D	F	E
E →	F	F
F	F	F

L'automate concatené est alors



Sa matrice est

	a	b	ε
→ A	B	C	
B	C	C	D
C	C	C	
D	F	E	
E →	F	F	
F	F	F	

dont l'ε-libéré est l'AEF

	a	b
→ A	B	C
B	C, F	C, E
C	C	C
D	F	E
E →	F	F
F	F	F

dont l'automate déterministe associé est

	a	b
→ {A}	{B}	{C}
{B}	{C, F}	{C, E}
{C}	{C}	{C}
{C, F}	{C, F}	{C, F}
{C, E} →	{C, F}	{C, F}

dont, pour y voir plus claire, nous pouvont renommer les états

	a	b
→ U	V	W
V	X	Y
W	W	W
X	X	X
Y →	X	X

Proposition 3.7.2

Soient $\mathcal{A}$ et $\mathcal{B}$ deux automates.

$$L((\mathcal{A}\mathcal{B})_\varepsilon) = L(\mathcal{A}\mathcal{B}) = L(\mathcal{A})L(\mathcal{B})$$

Démonstration. C'est une conséquence triviale de la définition. □

Exercice 23

Pour chacune des paires d'automates suivants $\mathcal{A}$ et $\mathcal{B}$, déterminer l'ADEF, représentant la concaténation $\mathcal{A}\mathcal{B}$ et $\mathcal{B}\mathcal{A}$.

1. $\mathcal{A} =$

	a	b
$\to A \to$	A	B
B	B	B

$\mathcal{B} =$

	a	b
$\to A$	A	B
$B \to$	B	B

2. $\mathcal{A} =$

	a	b
$\to A$	B	A
$B \to$	B	A

$\mathcal{B} =$

	a	b
$\to A$	C	B
B	B	B
$C \to$	C	B

3.8 Condition 6 : automate reconnaissant une étoile

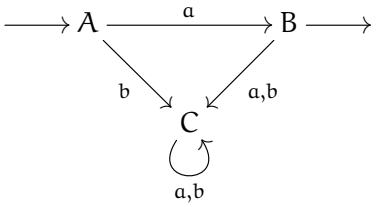
Définition 3.8.1

Soit $\mathcal{A}$ un automate sur un alphabet Σ. On définit $\mathcal{A}^*$, l'automate a ε-transition, par les règles suivantes.

- $\Sigma_{\mathcal{A}^*} = \Sigma$
- $E_{\mathcal{A}^*} = E_{\mathcal{A}}$
- $I_{\mathcal{A}^*} = I_{\mathcal{A}}$
- $F_{\mathcal{A}^*} = I_{\mathcal{A}}$
- La fonction de transition

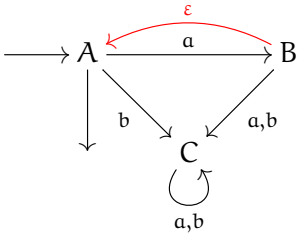
$$\begin{aligned} \tau_{\mathcal{A}^*} : E_{\mathcal{A}^*} \times \Sigma \cup \varepsilon &\longrightarrow \mathcal{P}(E_{\mathcal{A}^*}) \\ \forall x \in \Sigma \ (E, x) &\longmapsto \tau_{\mathcal{A}}(E, x) \\ \forall f \in F_{\mathcal{A}}, \forall i \in I_{\mathcal{A}}, \ (f, \varepsilon) &\longmapsto i \end{aligned}$$

Sur l'alphabet $\Sigma = \{a, b\}$, considérons l'automate reconnaissant $\{a\}$:



	a	b
$\to A$	B	C
$B \to$	C	C
C	C	C

Alors l'automate étoile est



Sa matrice est

	a	b	ε
→ A →	B	C	
B	C	C	A
C	C	C	

dont l'ε-libéré est l'AEF

	a	b
→ A →	B	C
B →	B, C	C
C	C	C

dont l'automate déterministe est

	a	b
→ {A} →	{B}	{C}
{B} →	{B, C}	{C}
{C}	{C}	{C}
{B, C} →	{B, C}	{C}

dont, pour y voir plus claire, nous pouvons renommer les états

	a	b
→ U →	V	W
V →	X	W
W	W	W
X →	X	W

Proposition 3.8.2

Soit $\mathcal{A}$ un automate.

$$L((\mathcal{A}^*)_\varepsilon) = L(\mathcal{A}^*) = L(\mathcal{A})^*$$

Démonstration. C'est une conséquence triviale de la définition □

Exercice 24

Pour chacun des automates suivants déterminer l'ADEF représentant l'automate étoile.

1.

	a	b
→ A →	A	B
B	B	B

2.

	a	b
→ A	A	B
B →	B	B

3.

	a	b
→ A	B	A
B →	B	A

4.

	a	b
→ A	C	B
B	B	B
C →	C	B

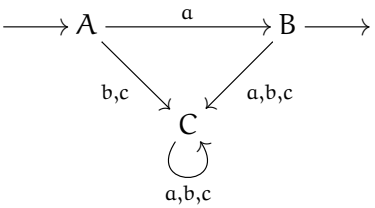
3.9 Un exemple

Considérons l'alphabet $\Sigma = \{a, b, c\}$ et déterminons un automate reconnaissant la REGEX $(a + b)c^*$. Le parenthèsage permet d'identifier la priorité des calculs. Il nous faut donc

1. Déterminer un automate reconnaissant a
2. Déterminer un automate reconnaissant b
3. En déduire un automate reconnaissant la somme $a + b$
4. Déterminer un automate reconnaissant c
5. En déduire un automate reconnaissant l'étoile c^*
6. Finalement, en déduire un automate reconnaissant le concaténé $(a + b)c^*$

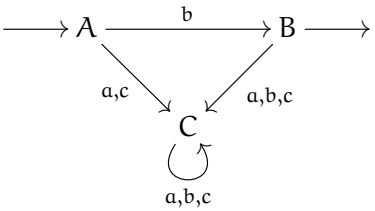
Pour les caractères nous pouvons utiliser la condition 3 :

On note $\mathcal{A}$ l'automate



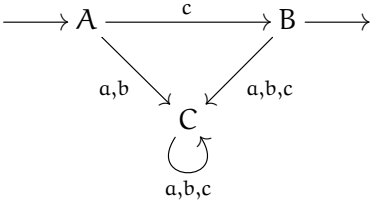
	a	b	c
$\to$ A	B	C	C
B $\to$	C	C	C
C	C	C	C

On note $\mathcal{B}$ l'automate



	a	b	c
$\to$ A	C	B	C
B $\to$	C	C	C
C	C	C	C

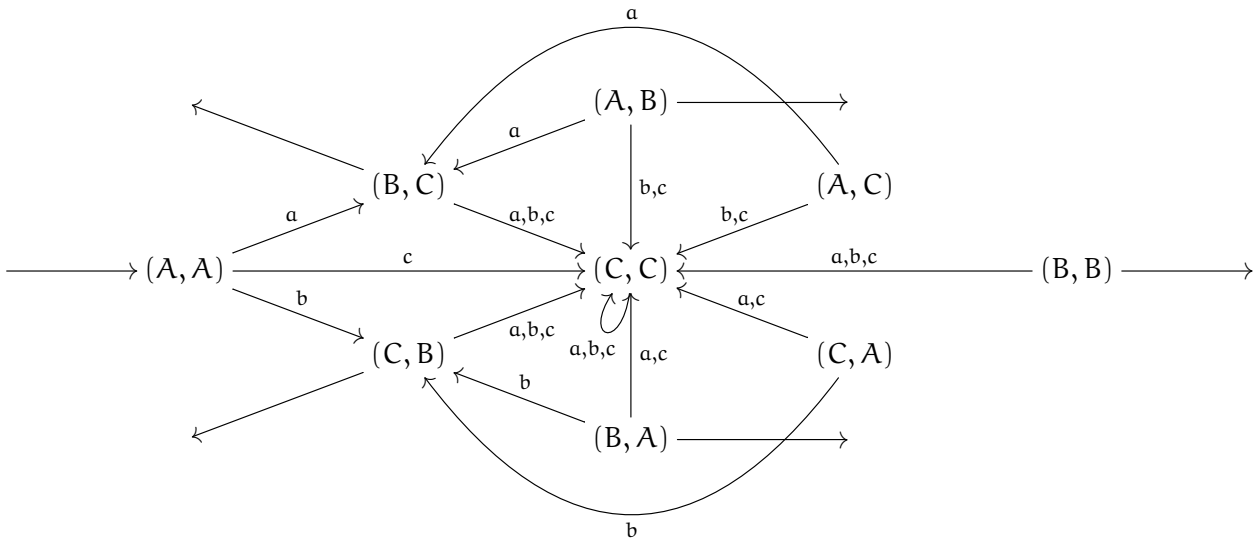
On note $\mathcal{C}$ l'automate



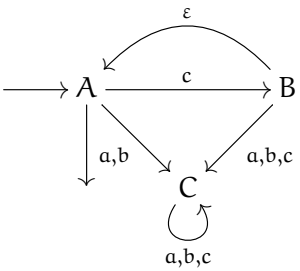
	a	b	c
$\to$ A	C	C	B
B $\to$	C	C	C
C	C	C	C

Nous pouvons alors réaliser la somme $\mathcal{A} + \mathcal{B}$ en utilisant les outils développés à la condition 4.

	a	b	c
$\to$ (A, A)	(B, C)	(C, B)	(C, C)
(A, B) $\to$	(B, C)	(C, C)	(C, C)
(A, C)	(B, C)	(C, C)	(C, C)
(B, A) $\to$	(C, C)	(C, B)	(C, C)
(B, B) $\to$	(C, C)	(C, C)	(C, C)
(B, C) $\to$	(C, C)	(C, C)	(C, C)
(C, A)	(C, C)	(C, B)	(C, C)
(C, B) $\to$	(C, C)	(C, C)	(C, C)
(C, C)	(C, C)	(C, C)	(C, C)



On à présent l'automate $\mathcal{C}^*$ en utilisant la construction 6.



de matrice

	a	b	c	ε
→ A →	C	C	B	
B	C	C	C	A
C	C	C	C	

d'ε-libéré

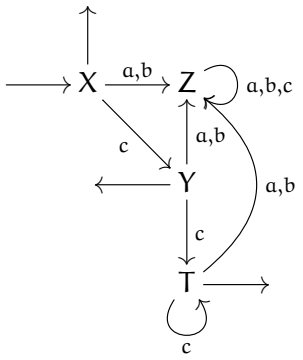
	a	b	c
→ A →	C	C	B
B →	C	C	B, C
C	C	C	C

d'automate déterministe associé

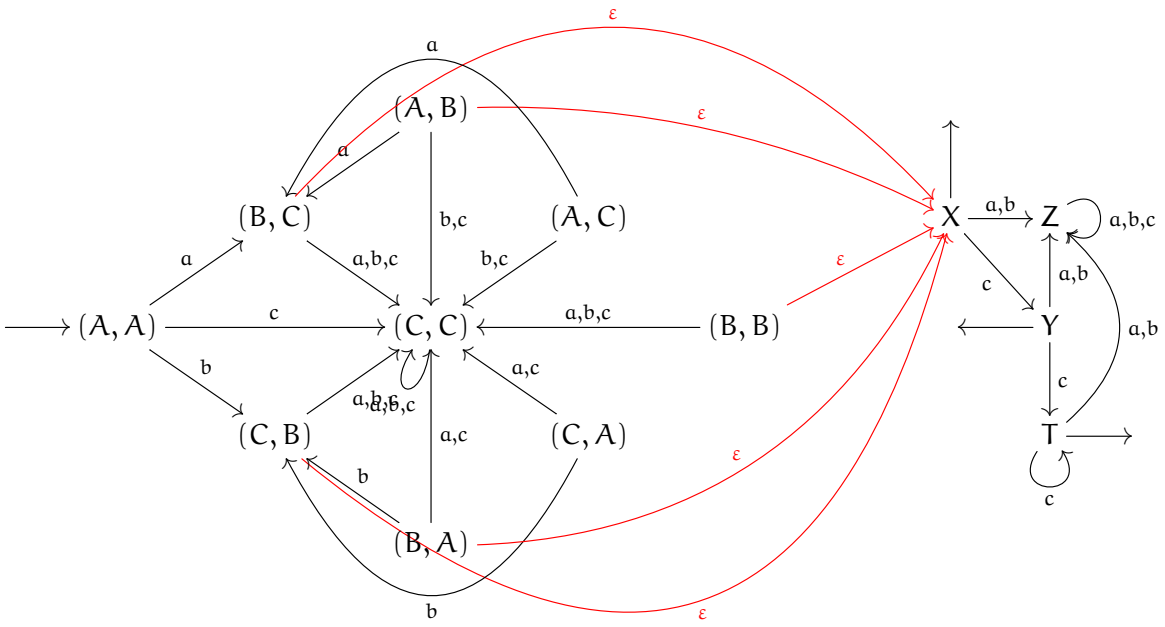
	a	b	c
→ {A} →	{C}	{C}	{B}
{B} →	{C}	{C}	{B, C}
{C}	{C}	{C}	{C}
{B, C} →	{C}	{C}	{B, C}

En renommant les sommets, on arrive à l'automate

	a	b	c
→ X →	Z	Z	Y
Y →	Z	Z	T
Z	Z	Z	Z
T →	Z	Z	T



Finalement, on concatène l'automate $\mathcal{A} + \mathcal{B}$ et $\mathcal{C}^*$ par des ε-transition :



Sa matrice est

	a	b	c	ε
$\to (A, A)$	(B, C)	(C, B)	(C, C)	
(A, B)	(B, C)	(C, C)	(C, C)	X
(A, C)	(B, C)	(C, C)	(C, C)	
(B, A)	(C, C)	(C, B)	(C, C)	X
(B, B)	(C, C)	(C, C)	(C, C)	X
(B, C)	(C, C)	(C, C)	(C, C)	X
(C, A)	(C, C)	(C, B)	(C, C)	
(C, B)	(C, C)	(C, C)	(C, C)	X
(C, C)	(C, C)	(C, C)	(C, C)	
$X \to$	Z	Z	Y	
$Y \to$	Z	Z	T	
Z	Z	Z	Z	
$T \to$	Z	Z	T	

L'ε-libéré est (on note simplement AB au lieu de (A, B) pour alléger les notations)

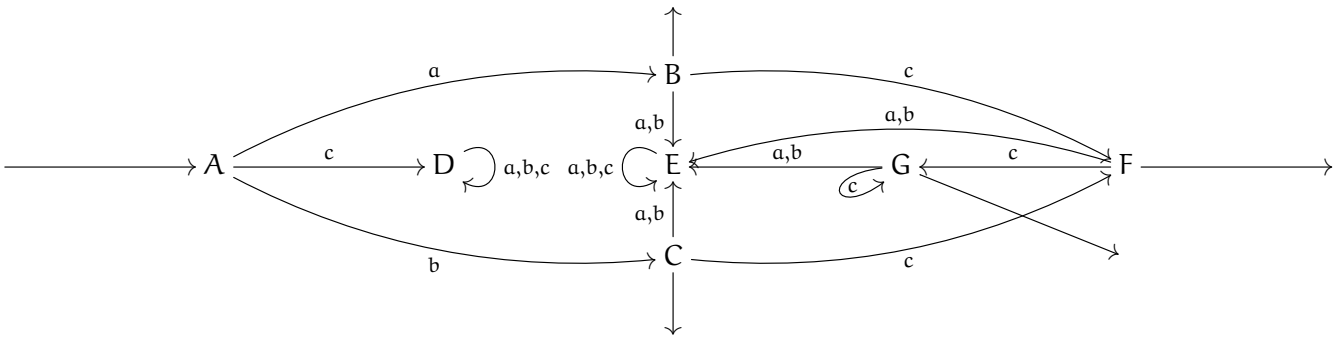
	a	b	c
$\to AA$	BC	CB	CC
$AB \to$	BC, Z	CC, Z	CC, Y
AC	BC	CC	CC
$BA \to$	CC, Z	CB, Z	CC, Y
$BB \to$	CC, Z	CC, Z	CC, Y
$BC \to$	CC, Z	CC, Z	CC, Y
CA	CC	CB	CC
$CB \to$	CC, Z	CC, Z	CC, Y
CC	CC	CC	CC
$X \to$	Z	Z	Y
$Y \to$	Z	Z	T
Z	Z	Z	Z
$T \to$	Z	Z	T

Finalement l'automate déterministe est

	a	b	c
$\to \{AA\}$	$\{BC\}$	$\{CB\}$	$\{CC\}$
$\{BC\} \to$	$\{CC, Z\}$	$\{CC, Z\}$	$\{CC, Y\}$
$\{CB\} \to$	$\{CC, Z\}$	$\{CC, Z\}$	$\{CC, Y\}$
$\{CC\}$	$\{CC\}$	$\{CC\}$	$\{CC\}$
$\{CC, Z\}$	$\{CC, Z\}$	$\{CC, Z\}$	$\{CC, Z\}$
$\{CC, Y\} \to$	$\{CC, Z\}$	$\{CC, Z\}$	$\{CC, T\}$
$\{CC, T\} \to$	$\{CC, Z\}$	$\{CC, Z\}$	$\{CC, T\}$

En renommant les sommets, on arrive à

	a	b	c
$\to A$	B	C	D
$B \to$	E	E	F
$C \to$	E	E	F
D	D	D	D
E	E	E	E
$F \to$	E	E	G
$G \to$	E	E	G



Exercice 25

Sur l'alphabet $\Sigma = \{a, b\}$ déterminer des automates reconnaissant les REGEX suivantes.

1. ab^*
2. $(ab)^*$
3. $a + b^*$
4. $(a + b)^*$