Théorie des Langages

Cours 3 : Opérations, expressions régulières, implémentation

Lionel Rieg

Grenoble INP - Ensimag, 1re année

Une propriété

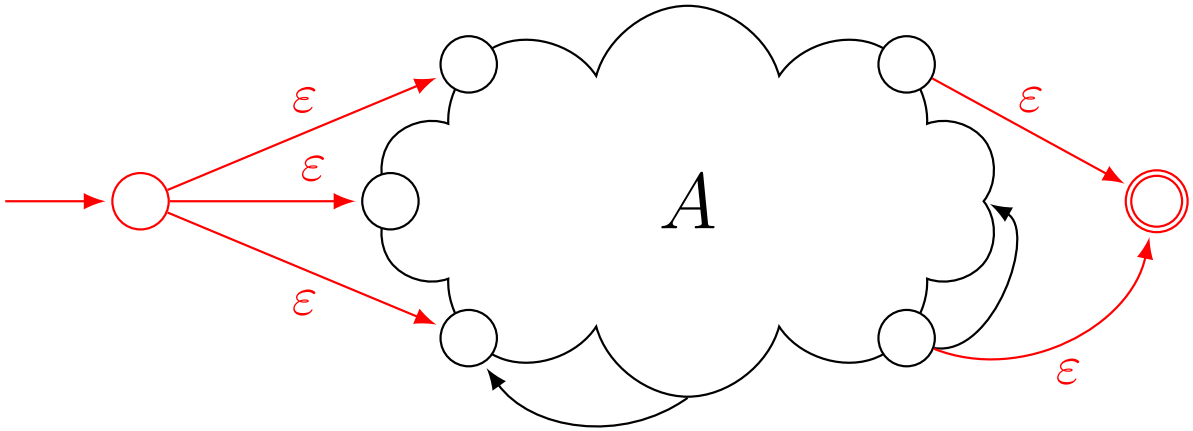
Proposition

Pour tout automate fini A, il existe un automate fini B avec :

- *un unique état initial sans transition entrante,*
- *un unique état acceptant sans transition sortante,*

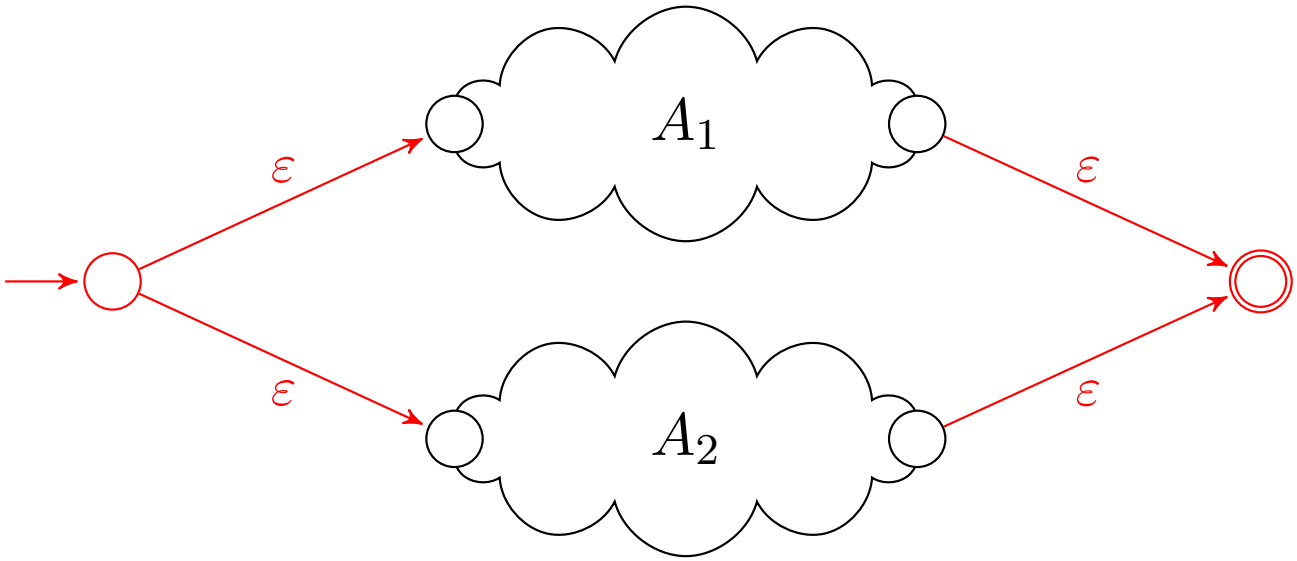
équivalent à A.

Construction :



Union

Etant donnés deux automates A_1 et A_2, construire un automate reconnaissant $\mathcal{L}(A_1) \cup \mathcal{L}(A_2)$.



Formellement :
Si $A_1 = (Q_1, V, \delta_1, I_1, F_1)$ et $A_2 = (Q_2, V, \delta_2, I_2, F_2)$, on a :

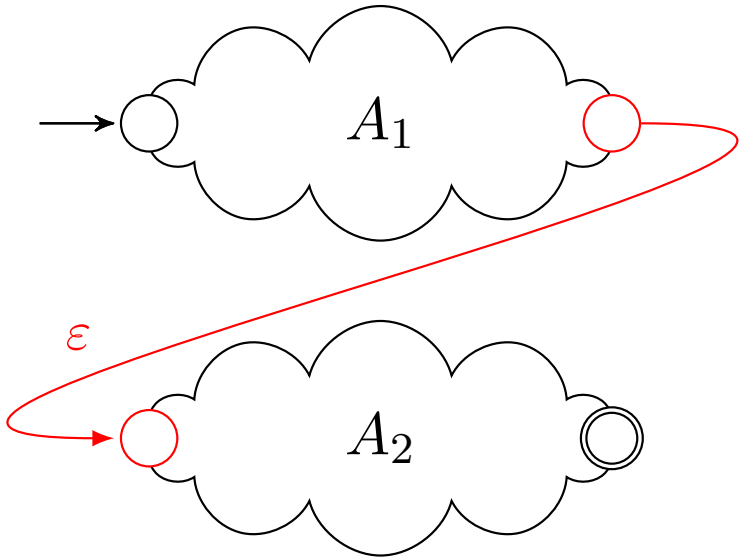
$$A_1 \cup A_2 \stackrel{\text{def}}{=} (Q_1 \uplus Q_2 \uplus \{i, f\}, V, \delta, \{i\}, \{f\})$$

avec

$$\delta \stackrel{\text{def}}{=} \delta_1 \cup \delta_2 \cup \{(i, \varepsilon, q) \mid q \in I_1 \cup I_2\} \cup \{(q, \varepsilon, f) \mid q \in F_1 \cup F_2\}$$

Concaténation

Etant donnés deux automates A_1 et A_2, construire un automate reconnaissant $\mathcal{L}(A_1).\mathcal{L}(A_2)$.



Exercice : écrire formellement cette transformation

Si $A_1 = (Q_1, V, \delta_1, I_1, F_1)$ et $A_2 = (Q_2, V, \delta_2, I_2, F_2)$, on a :

$$A_1.A_2 \stackrel{\text{def}}{=} (Q_1 \uplus Q_2, V, \delta, I_1, F_2)$$

avec $$\delta \stackrel{\text{def}}{=} \delta_1 \cup \delta_2 \cup \{(f, \varepsilon, i) \mid f \in F_1, i \in I_2\}$$

Comment montrer que cette définition est correcte ?

On pose $A_1.A_2 \stackrel{\text{def}}{=} (Q_1 \uplus Q_2, V, \delta, I_1, F_2)$

$\delta \stackrel{\text{def}}{=} \delta_1 \cup \delta_2 \cup \{(f, \varepsilon, i) \mid f \in F_1, i \in I_2\}$

Montrer que $\mathcal{L}(A_1.A_2) = \mathcal{L}(A_1).\mathcal{L}(A_2)$.

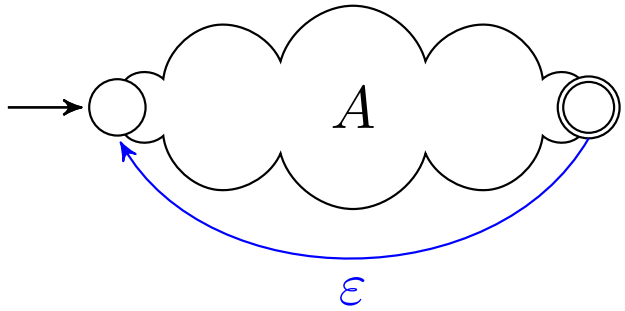
Par double inclusion :

- $\mathcal{L}(A_1).\mathcal{L}(A_2) \subseteq \mathcal{L}(A_1.A_2)$: Soient
  $w_1 \in \mathcal{L}(A_1) \iff \exists \chi_1$ chemin de $i_1 \in I_1$ à $f_1 \in F_1$ de trace w_1
  $w_2 \in \mathcal{L}(A_2) \iff \exists \chi_2$ chemin de $i_2 \in I_2$ à $f_2 \in F_2$ de trace w_2
  Alors $\chi_1 \cdot (f_1, \varepsilon, i_2) \cdot \chi_2$ est un chemin de i_1 à f_2 dans $A_1.A_2$ de trace $w_1 w_2$. Ainsi, $w_1 w_2 \in \mathcal{L}(A_1.A_2)$.
- $\mathcal{L}(A_1.A_2) \subseteq \mathcal{L}(A_1).\mathcal{L}(A_2)$: Soit $w \in \mathcal{L}(A_1.A_2)$, donc il existe un chemin dans $A_1.A_2$ de $i \in I_1$ à $f \in F_2$ de trace w. Par contruction de $A_1.A_2$, ce chemin passe par une ε-transition entre un état $f_1 \in F_1$ et un état $i_2 \in I_2$. Ainsi, on a $\chi = \chi_1 \cdot (f_1, \varepsilon, i_2) \cdot \chi_2$. χ_1 et χ_2 sont acceptants dans A_1 et A_2 d'où $\text{tr}(\chi_1) \in \mathcal{L}(A_1)$ et $\text{tr}(\chi_2) \in \mathcal{L}(A_2)$.

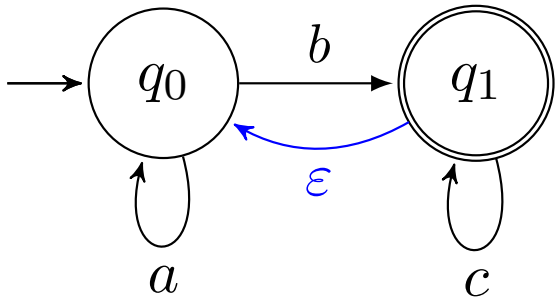
Exercice : Faire de même pour l'union $A_1 \cup A_2$.

Concaténation itérée

Etant donné un automate A, construire un automate reconnaissant $\mathcal{L}(A)^*$.



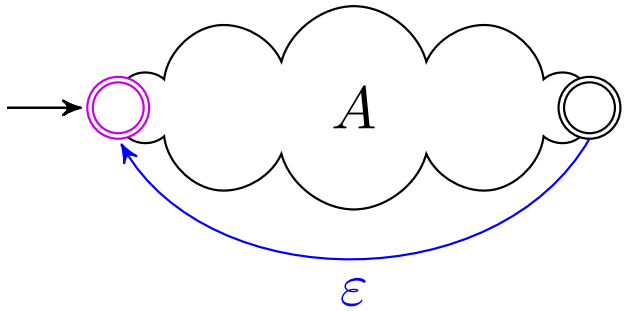
Exemple : $\mathcal{L}(A) = \{a\}^* \{b\} \{c\}^*$



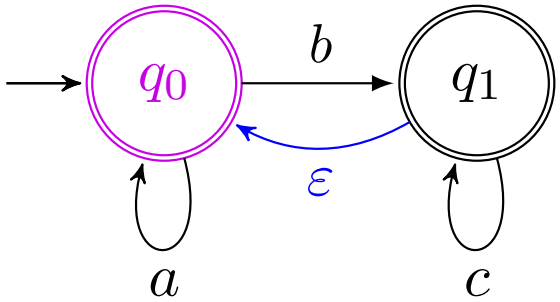
On ne reconnaît pas ε.

Concaténation itérée

Etant donné un automate A, construire un automate reconnaissant $\mathcal{L}(A)^*$.



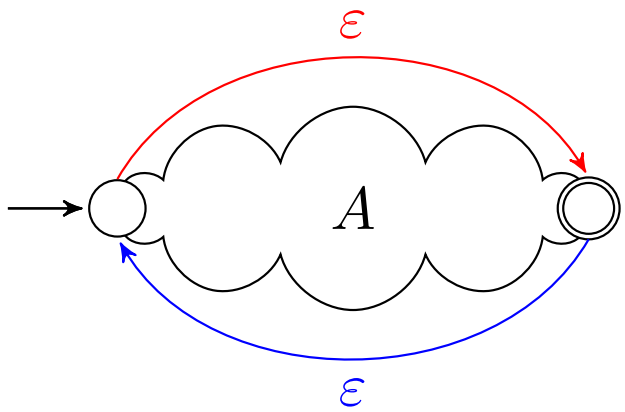
Exemple : $\mathcal{L}(A) = \{a\}^* \{b\} \{c\}^*$



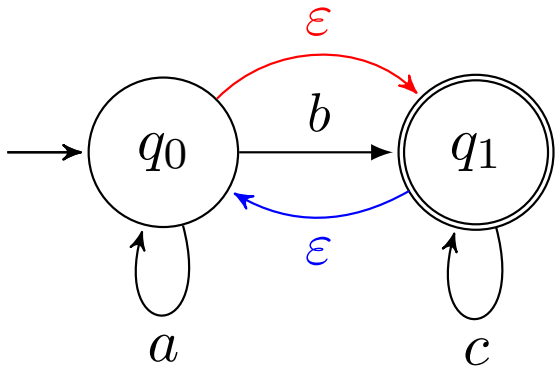
On reconnaît $\{a\}^*$ en trop.

Concaténation itérée

Etant donné un automate A, construire un automate reconnaissant $\mathcal{L}(A)^*$.



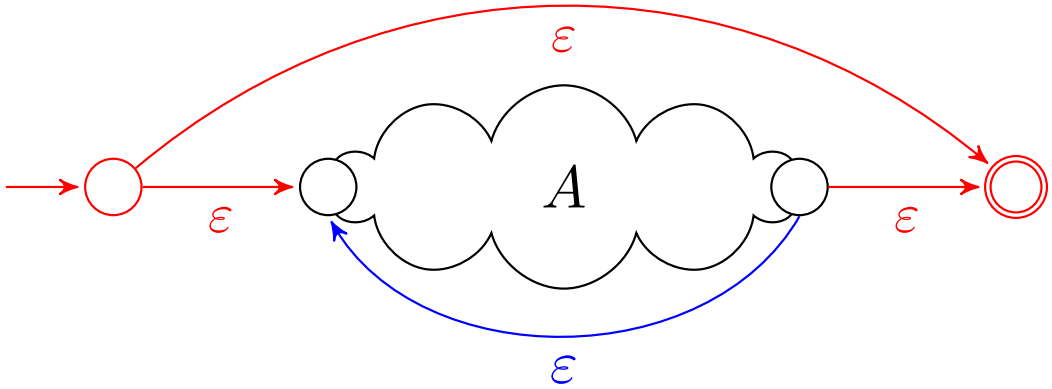
Exemple : $\mathcal{L}(A) = \{a\}^* \{b\} \{c\}^*$



On reconnaît $\{c\}^*$ en trop.

Concaténation itérée

Etant donné un automate A, construire un automate reconnaissant $\mathcal{L}(A)^*$.



Propriétés des langages réguliers

Théorème

La classe des langages réguliers est fermée :

- *par union*
- *par concaténation*
- *par concaténation itérée*

Nous en verrons d'autres au cours 5.

Quel est le lien entre langage régulier et expressions régulières ?

Rappel : définition inductive des ER

Base	$\emptyset$,	ϵ,	x	$(x \in V)$
Induction	$E_1 + E_2$,	$E_1.E_2$,	E_1^*	

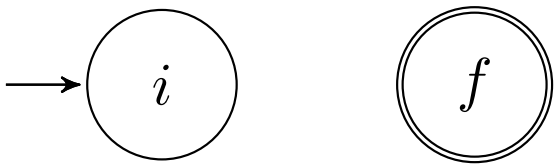
Expressions régulières et langage régulier

Lemme

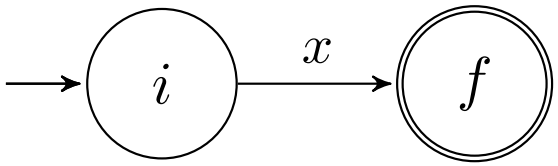
Une ER représente un langage régulier.
$\forall E(ER), \exists A(AF)$ *tel que* $\mathcal{L}(E) = \mathcal{L}(A)$
De plus, A *a un unique état initial et un unique état acceptant.*

Preuve : par induction structurelle

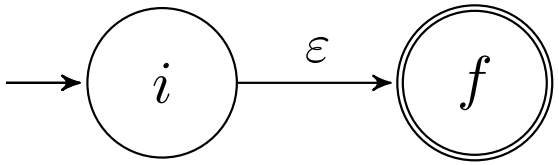
- $E = \emptyset$



- $E = x \in V$



- $E = \epsilon$



- $E = E_1 + E_2,\ E_1.E_2,\ E_1^*$

  Hyp. Ind.
  + propriétés précédentes

Expressions régulières équivalentes

Définition

Deux expressions régulières E et E' sont équivalentes si $\mathcal{L}(E) = \mathcal{L}(E')$.

Exemple

Les expressions régulières $(a+b)^*$ et $(a^*b^*)^*$ sont équivalentes.
Exercice : démontrer ce résultat

Théorème (Kleene)

Les langages représentés par des expressions régulières sont les langages réguliers.

Rappel : L régulier $\stackrel{\text{def}}{=} \exists A(\text{AF}) : \mathcal{L}(A) = L$

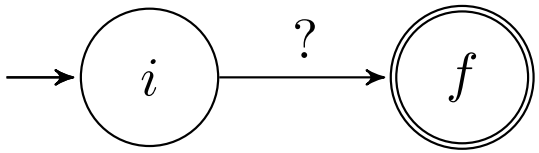
À démontrer :

1. $\forall E(\text{ER}), \exists A(\text{AF}) : \mathcal{L}(E) = \mathcal{L}(A)$
2. $\forall A(\text{AF}), \exists E(\text{ER}) : \mathcal{L}(A) = \mathcal{L}(E)$

Un langage régulier est représentable par une ER

2 visions possibles :

- Version graphique :
  - Idée : se ramener à un automate de la forme



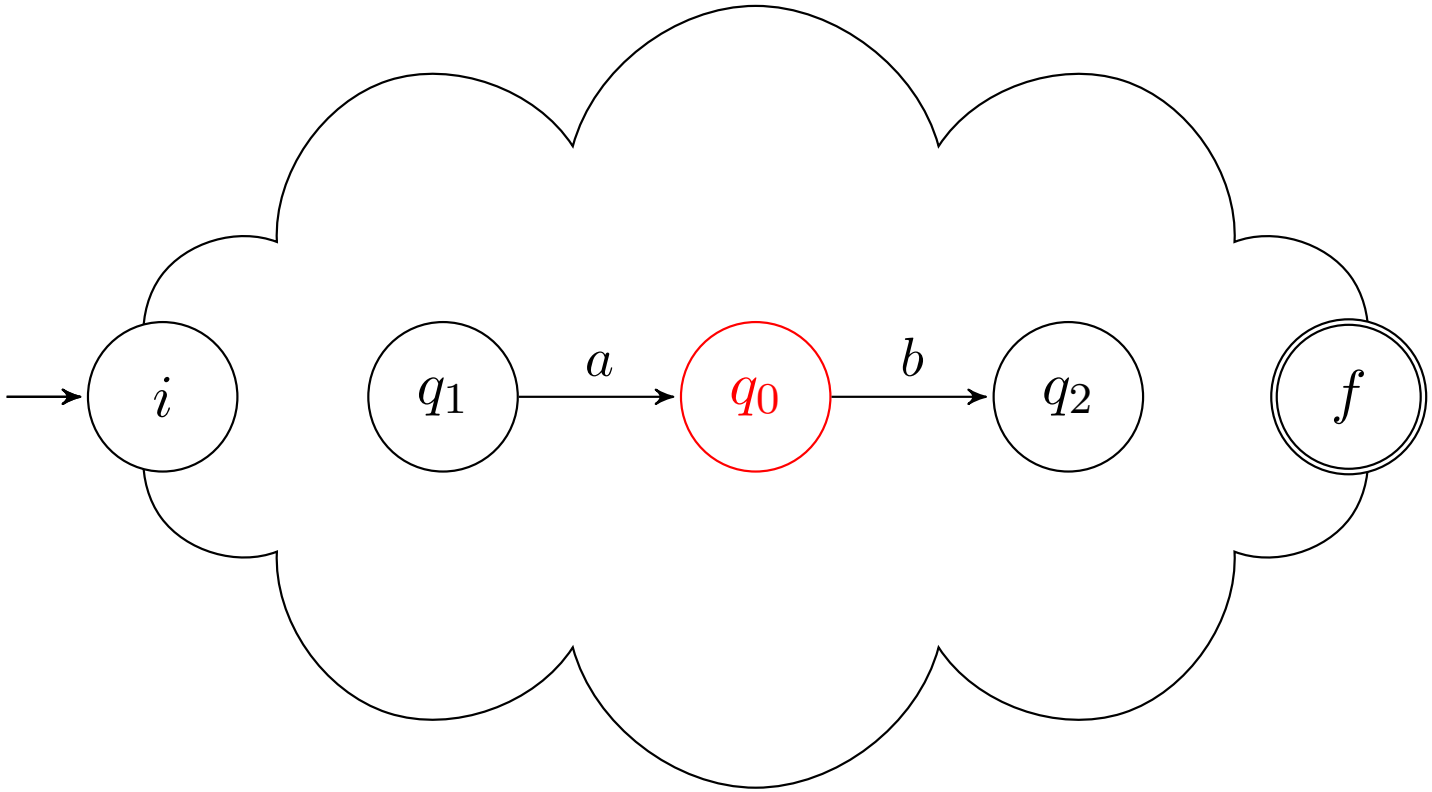
où ? est une ER

  - Méthode :
    - ★ S'autoriser des ER sur les transitions
    - ★ Partir d'un automate avec un unique état initial i sans transition entrante et un unique état acceptant f sans transition sortante (voir cours 3)
    - ★ Supprimer successivement les états (sauf i et f) en préservant le langage reconnu

      ⚠ Ne pas supprimer un état initial ou acceptant !
- Version par équations : résolution d'un système d'équations d'ER
  - Pas besoin de transformer l'automate
  - Représentation algébrique de la version graphique

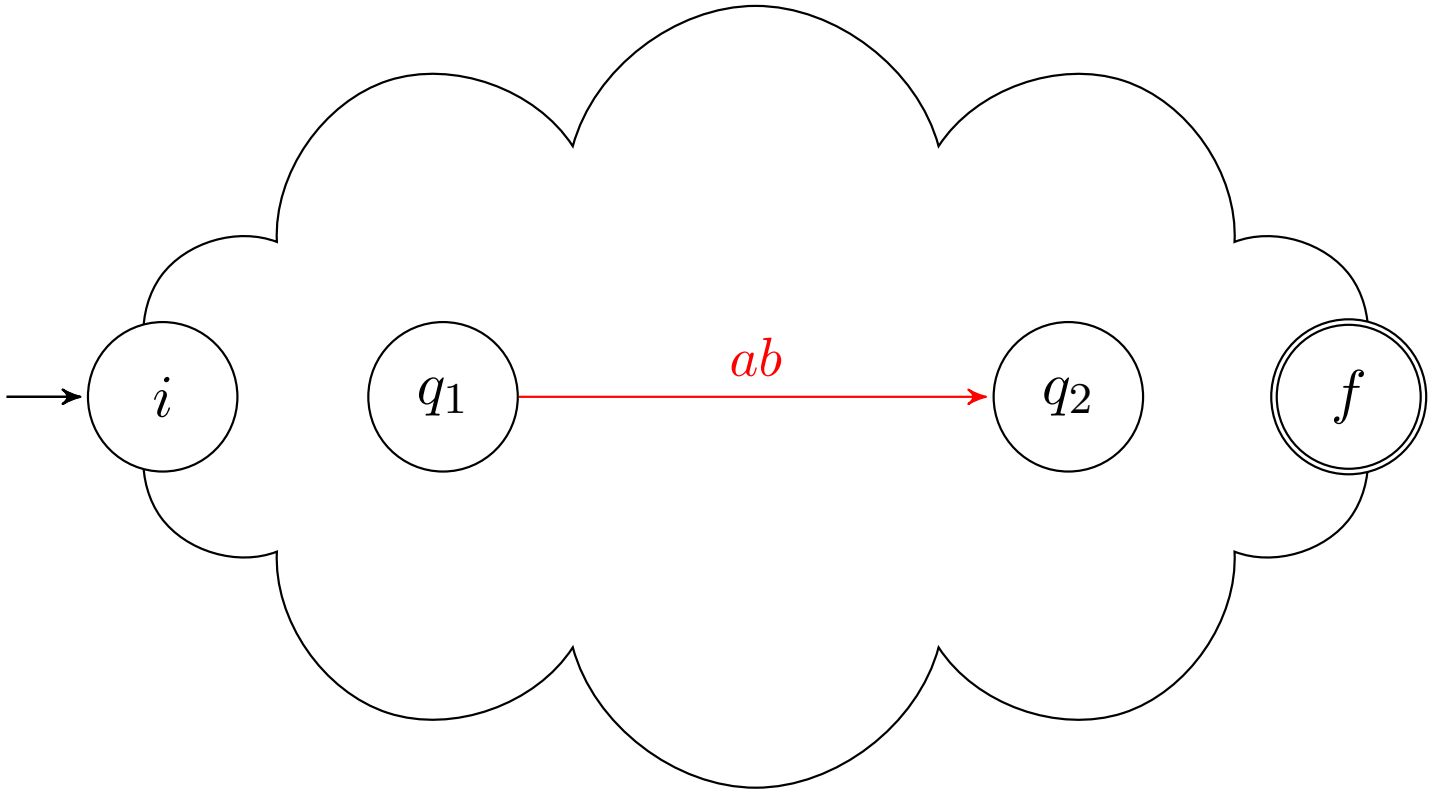
Version graphique

Comment supprimer un état d'un automate sans changer son langage ?



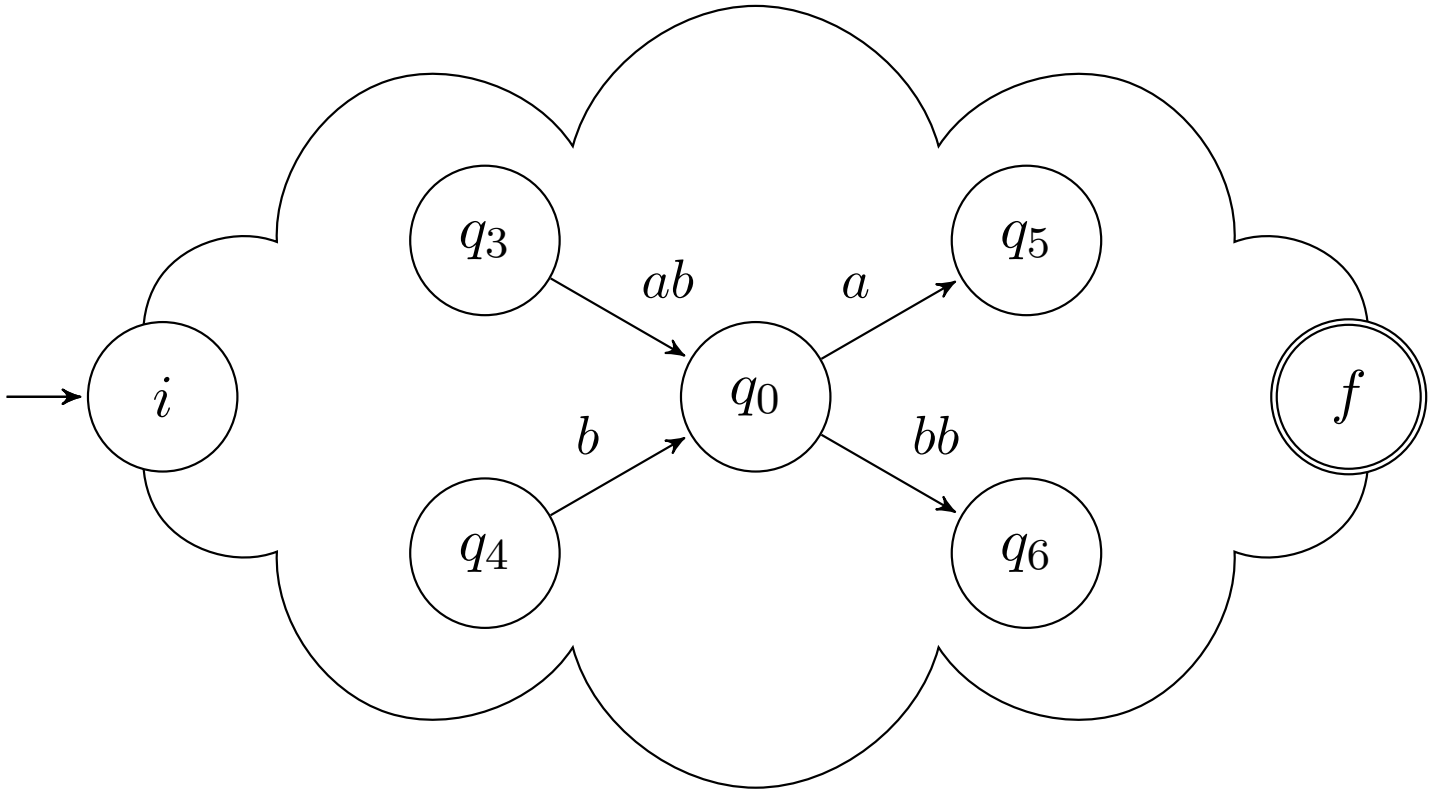
Version graphique

Comment supprimer un état d'un automate sans changer son langage ?



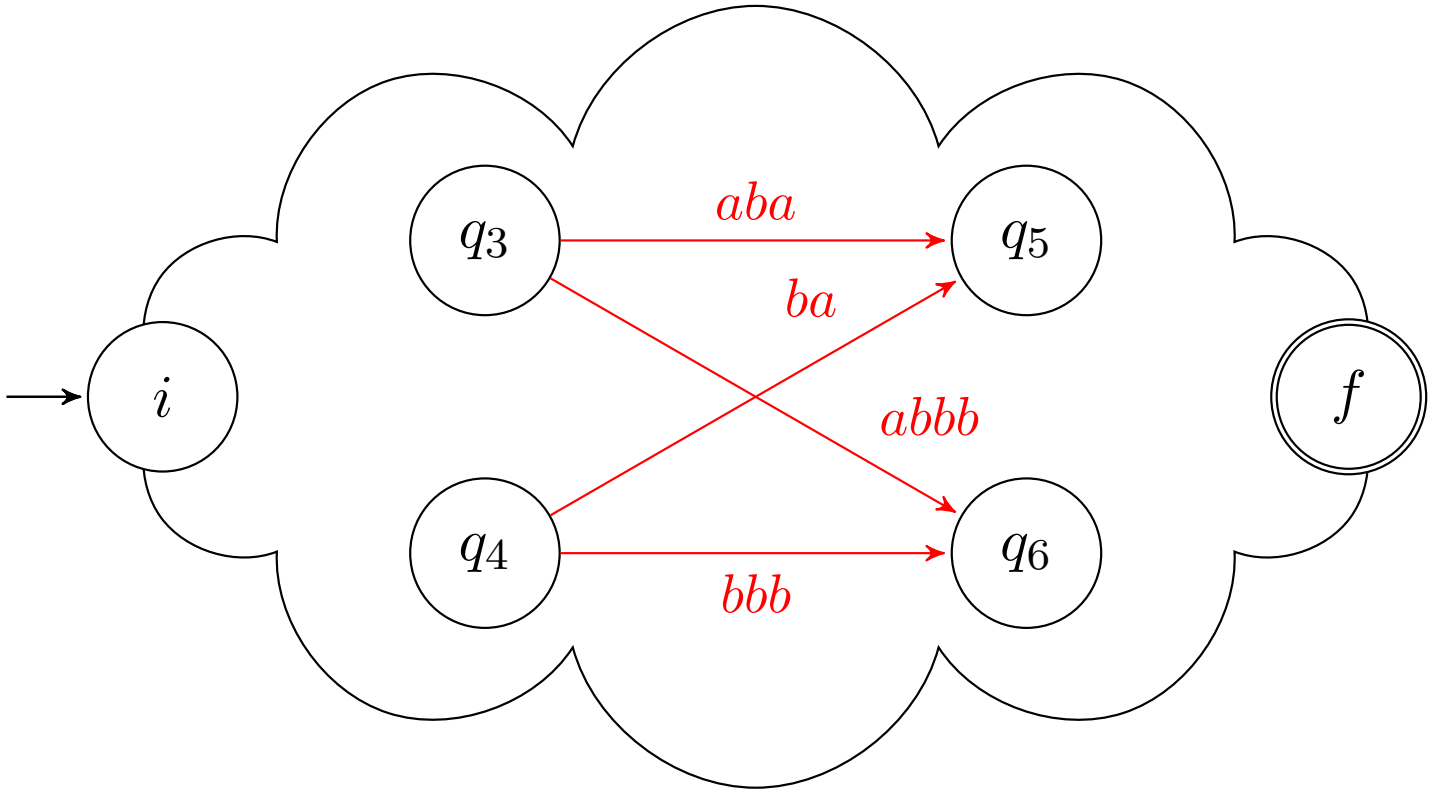
Version graphique

Comment supprimer un état d'un automate sans changer son langage ?



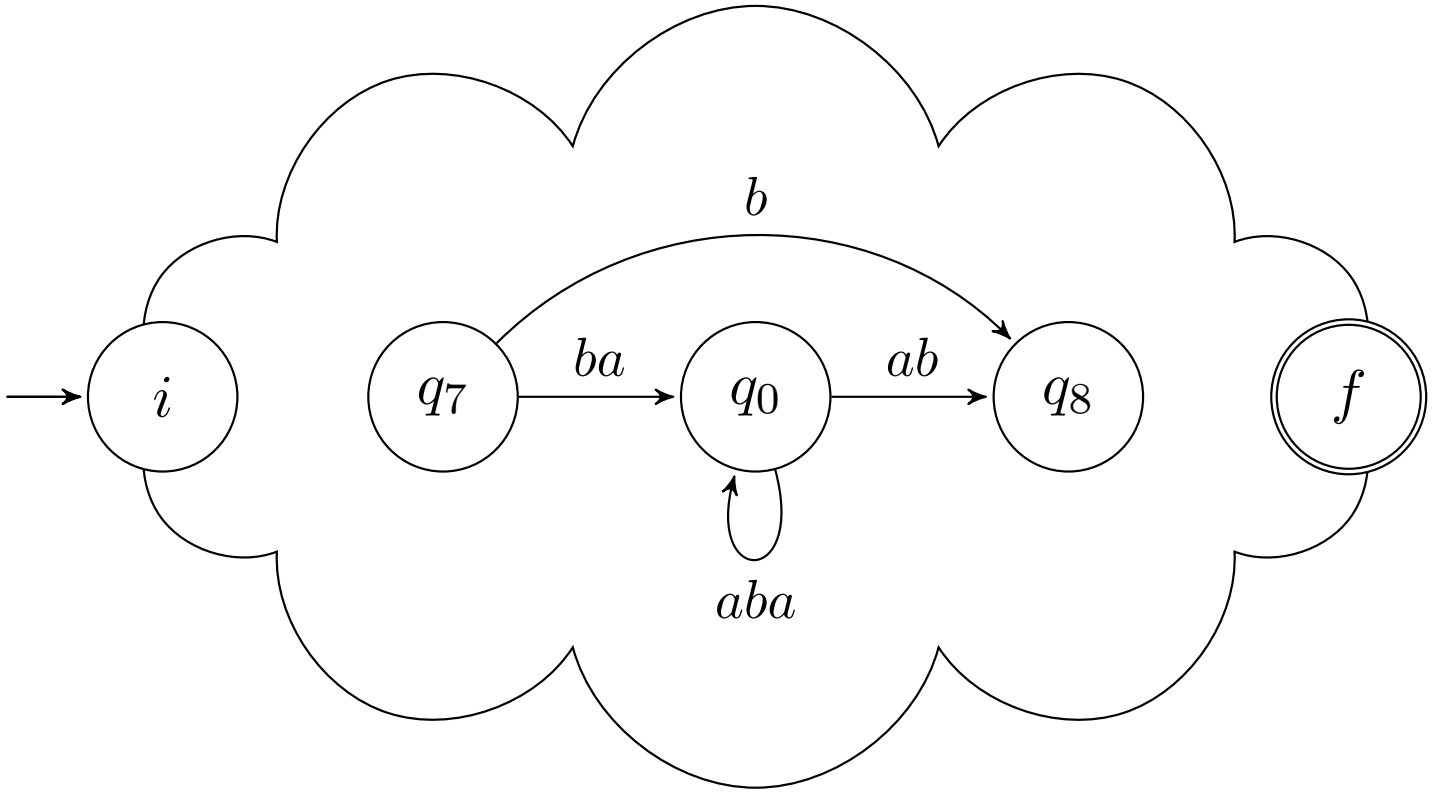
Version graphique

Comment supprimer un état d'un automate sans changer son langage ?



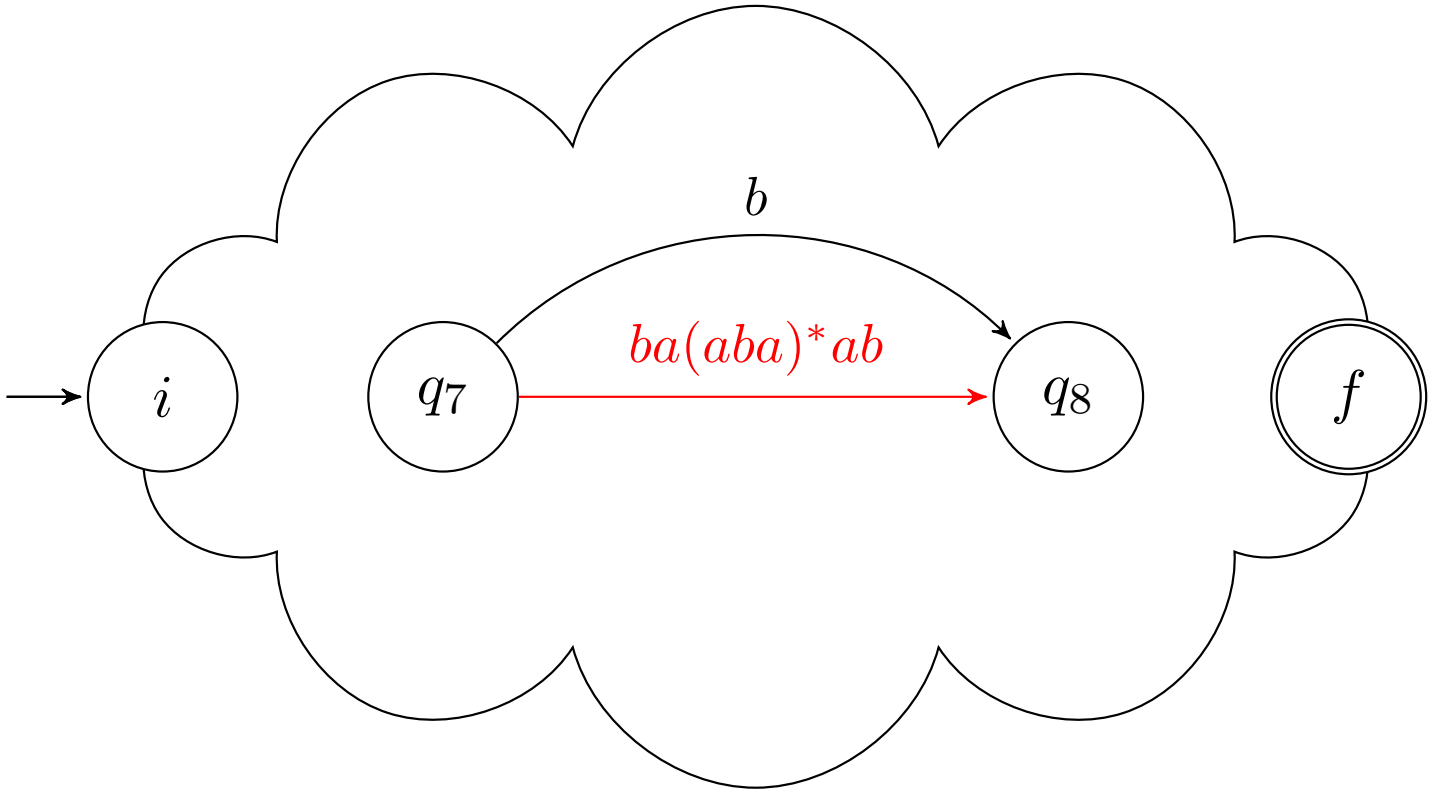
Version graphique

Comment supprimer un état d'un automate sans changer son langage ?



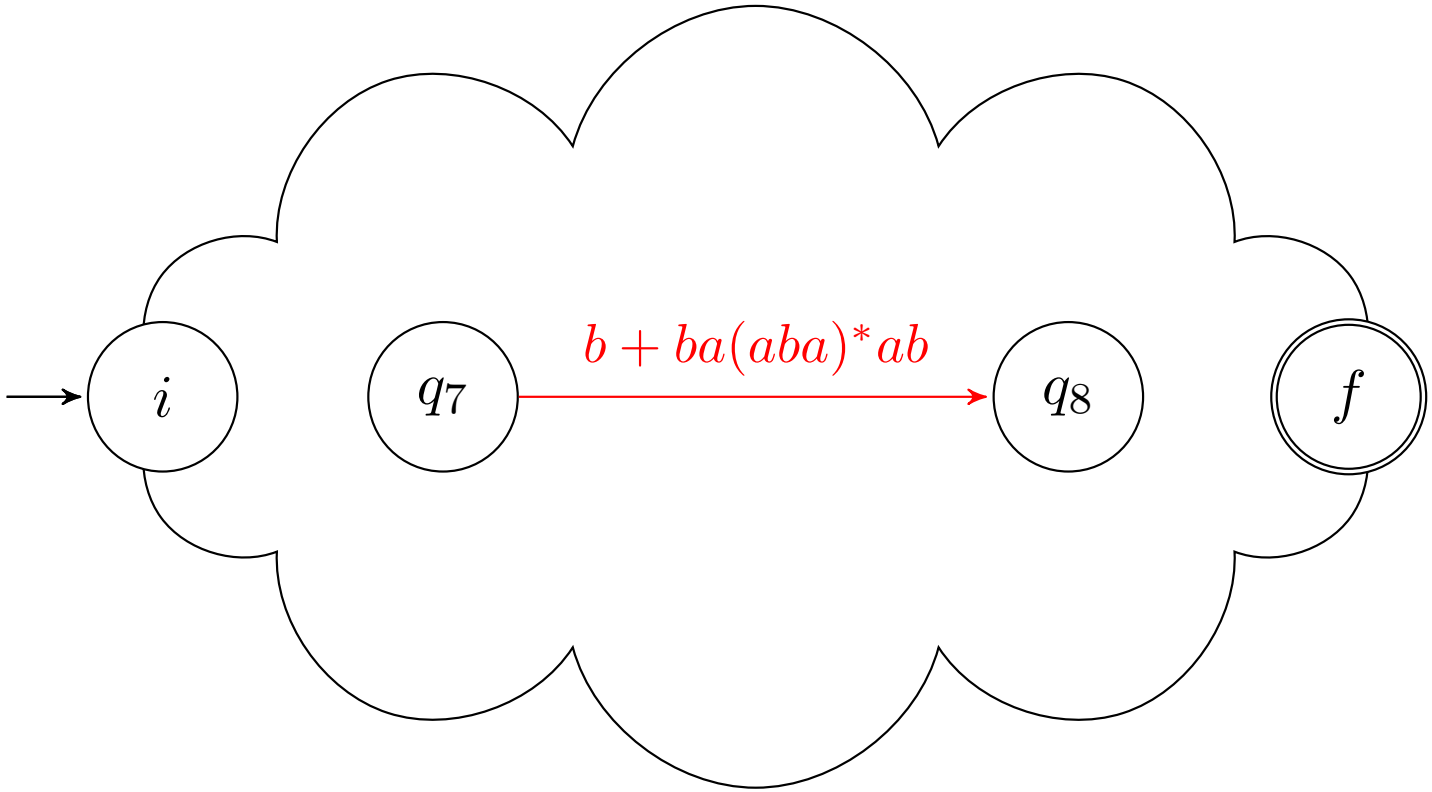
Version graphique

Comment supprimer un état d'un automate sans changer son langage ?



Version graphique

Comment supprimer un état d'un automate sans changer son langage ?



Version graphique

Comment supprimer un état d'un automate sans changer son langage ?

Formellement, pour supprimer un état q avec

- des transitions entrantes $(p, x, q) \in \delta$ (avec $p \neq q$)
- possiblement une boucle $(q, y, q) \in \delta$
- des transitions sortantes $(q, z, r) \in \delta$ (avec $r \neq q$)

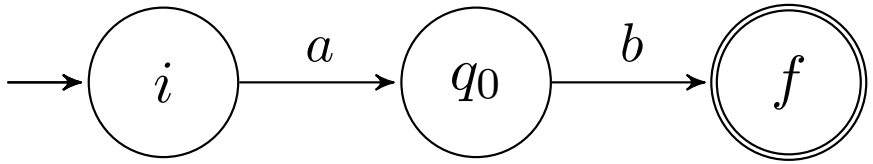
on doit

- supprimer l'état q et les transitions ci-dessus,
- ajouter à δ les transitions (p, xy^*z, r) (pour chaque couple (p, r)),
- possiblement fusionner des transitions parallèles.

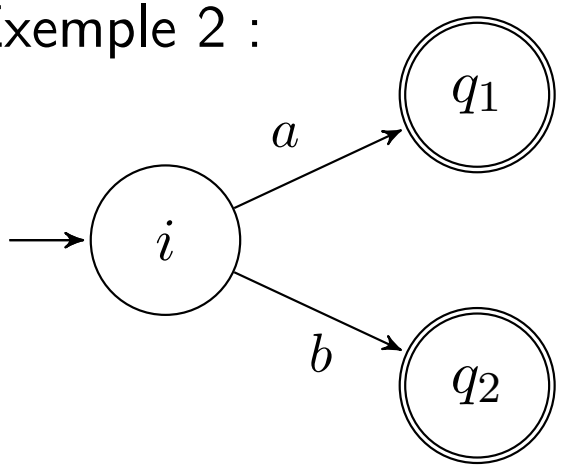
C'est la méthode de Brzozowski et Mc Cuskey.
On peut programmer cette méthode : l'algorithme de Kleene (prog. dyn.).

Version graphique : exemples

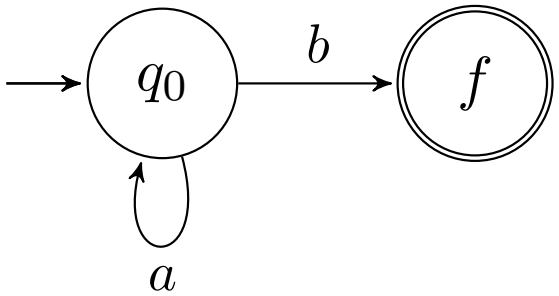
- Exemple 1 :



- Exemple 2 :

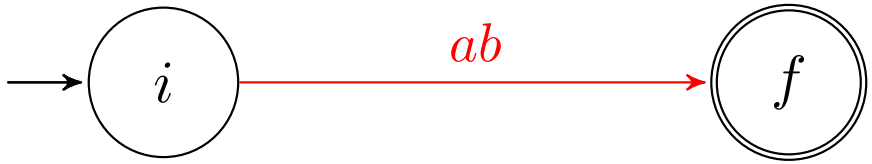


- Exemple 3 :

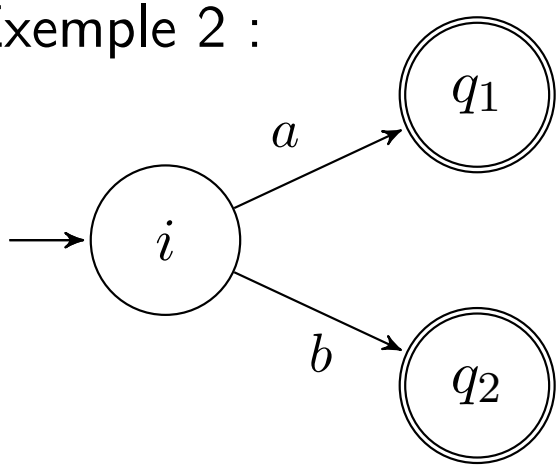


Version graphique : exemples

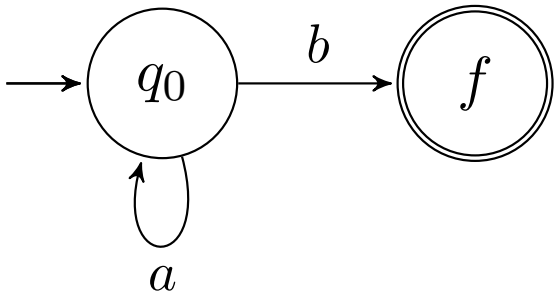
- Exemple 1 :



- Exemple 2 :

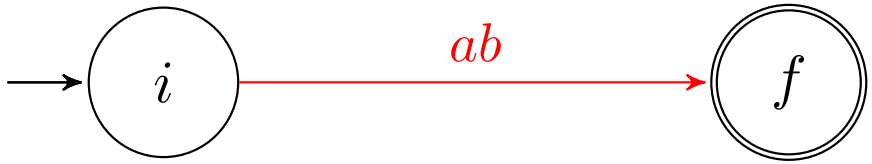


- Exemple 3 :

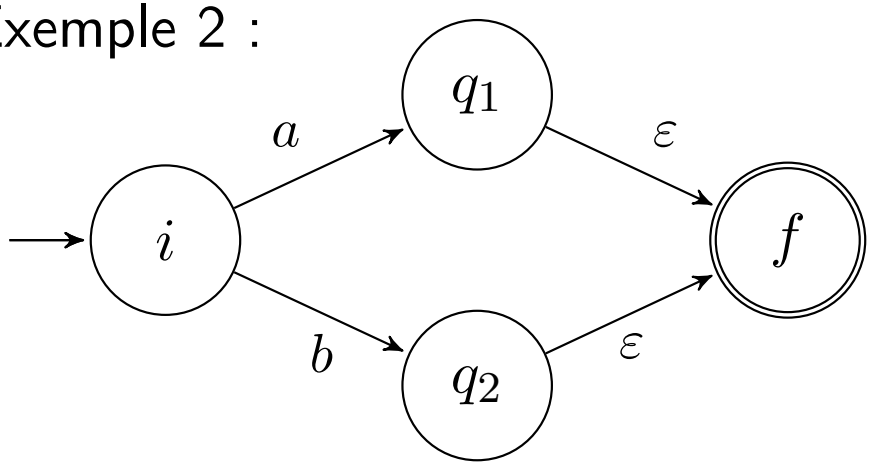


Version graphique : exemples

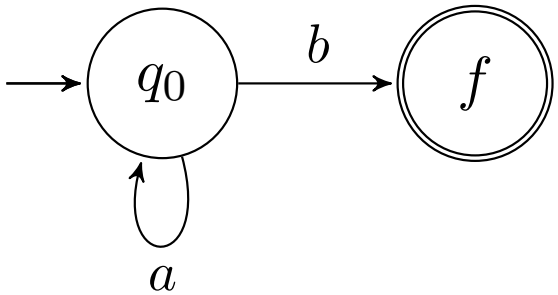
- Exemple 1 :



- Exemple 2 :

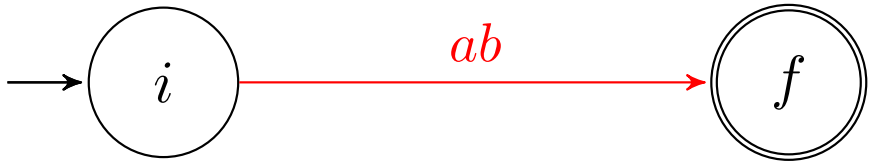


- Exemple 3 :

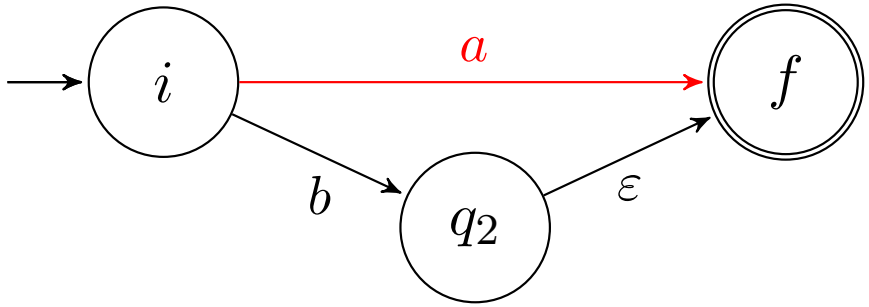


Version graphique : exemples

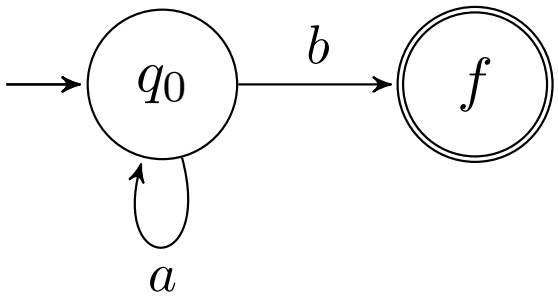
- Exemple 1 :



- Exemple 2 :

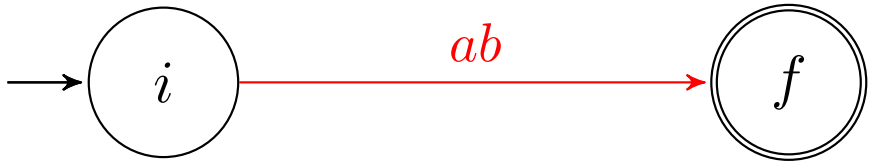


- Exemple 3 :

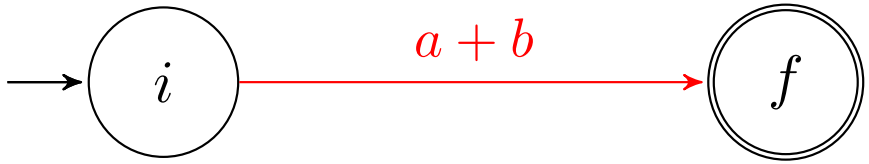


Version graphique : exemples

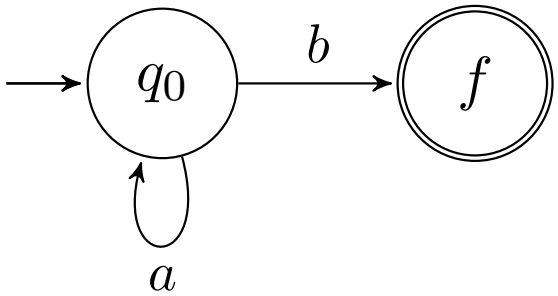
- Exemple 1 :



- Exemple 2 :

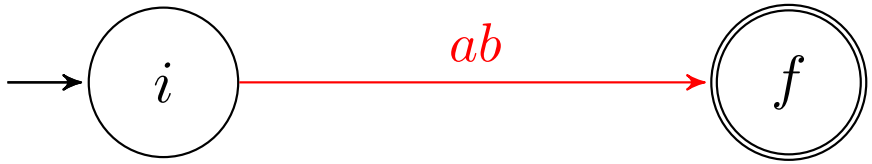


- Exemple 3 :

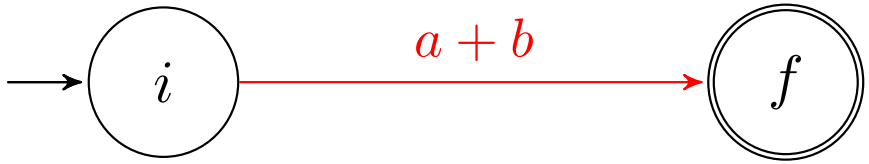


Version graphique : exemples

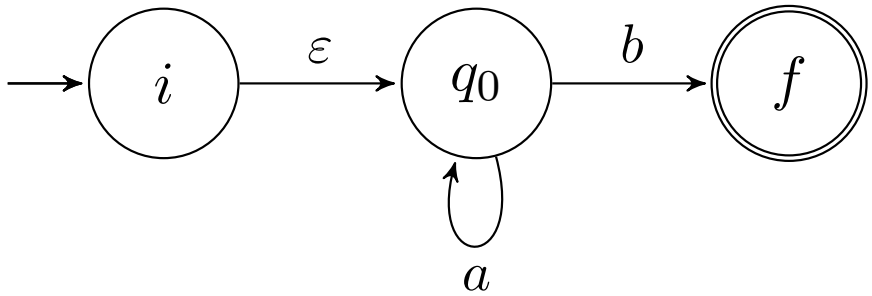
- Exemple 1 :



- Exemple 2 :

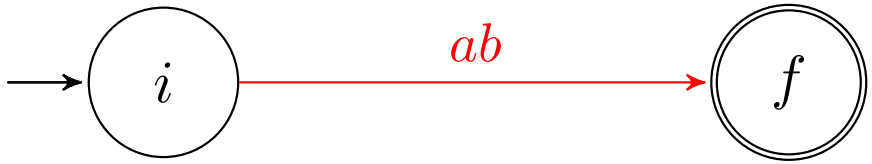


- Exemple 3 :

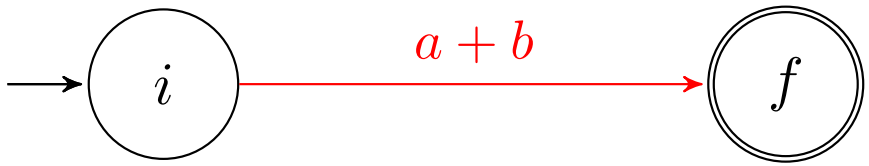


Version graphique : exemples

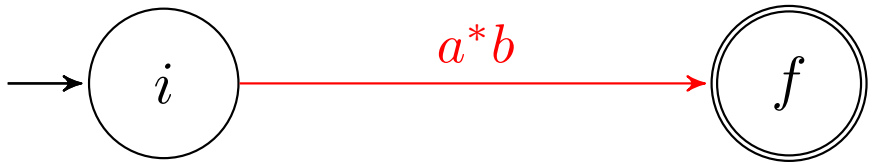
- Exemple 1 :



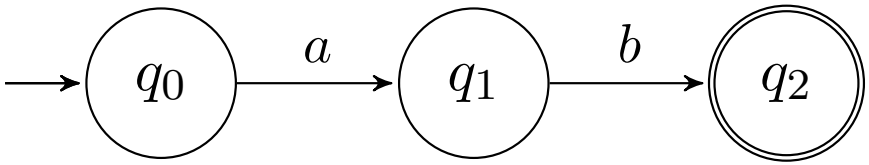
- Exemple 2 :



- Exemple 3 :



Version par équations : exemple 1



Considérons A_{q_i} la variante de A où l'unique état initial est q_i.

On a $$\mathcal{L}(A) = \mathcal{L}(A_{q_0})$$

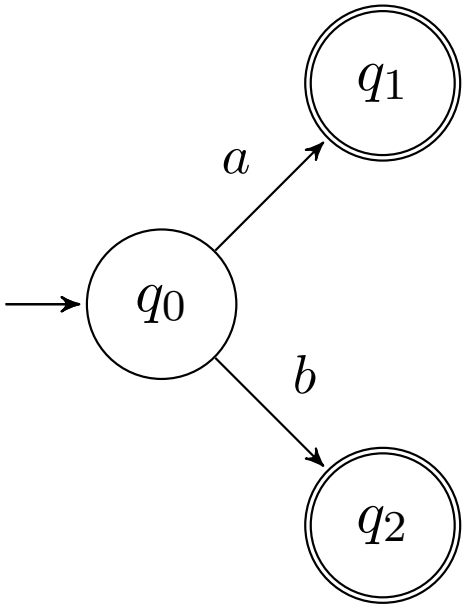
$$\begin{cases} \mathcal{L}(A_{q_0}) = \{a\}.\mathcal{L}(A_{q_1}) \\ \mathcal{L}(A_{q_1}) = \{b\}.\mathcal{L}(A_{q_2}) \\ \mathcal{L}(A_{q_2}) = \{\varepsilon\} \end{cases} \quad \Longrightarrow \quad \begin{cases} \mathcal{L}(A_{q_0}) = \{ab\} \\ \mathcal{L}(A_{q_1}) = \{b\} \\ \mathcal{L}(A_{q_2}) = \{\varepsilon\} \end{cases}$$

Système d'équations (sur des langages / des ER) :

$$\begin{cases} x_0 &= ax_1 \\ x_1 &= bx_2 \\ x_2 &= \epsilon \end{cases}$$

Solution : $x_0 = ab$

Version par équations : exemple 2



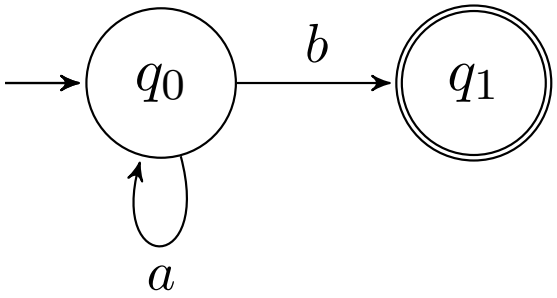
$$\begin{cases} \mathcal{L}(A_{q_0}) &= \{a\}.\mathcal{L}(A_{q_1}) \cup \{b\}.\mathcal{L}(A_{q_2}) \\ \mathcal{L}(A_{q_1}) &= \{\varepsilon\} \\ \mathcal{L}(A_{q_2}) &= \{\varepsilon\} \end{cases}$$

$$\begin{cases} \mathcal{L}(A_{q_0}) &= \{a, b\} \\ \mathcal{L}(A_{q_1}) &= \{\varepsilon\} \\ \mathcal{L}(A_{q_2}) &= \{\varepsilon\} \end{cases}$$

Système d'équations : $\begin{cases} x_0 &= ax_1 + bx_2 \\ x_1 &= \epsilon \\ x_2 &= \epsilon \end{cases}$

Solution : $x_0 = a + b$

Version par équations : exemple 3



$$\begin{cases} \mathcal{L}(A_{q_0}) = \{a\}.\mathcal{L}(A_{q_0}) \cup \{b\}.\mathcal{L}(A_{q_1}) \\ \mathcal{L}(A_{q_1}) = \{\varepsilon\} \end{cases} \Longrightarrow \begin{cases} \mathcal{L}(A_{q_0}) = \{a\}^* \{b\} \\ \mathcal{L}(A_{q_1}) = \{\varepsilon\} \end{cases}$$

Système d'équations : $\begin{cases} x_0 &= ax_0 + bx_1 \\ x_1 &= \epsilon \end{cases}$

Solution : $x_0 = a^*b$

Système d'équations associé à un automate A

Pour chaque état q_i :

- On considère une variable x_i, qui représentera le langage reconnu par l'automate dont q_i est l'unique état initial (A_{q_i})
- On considère les k transitions issues de q_i
  - $(q_i, a_{i_1}, q_{i_1}), (q_i, a_{i_2}, q_{i_2}), \ldots, (q_i, a_{i_k}, q_{i_k})$ où $a_{i_j} \in V \cup \{\varepsilon\}$
  - On crée l'équation :

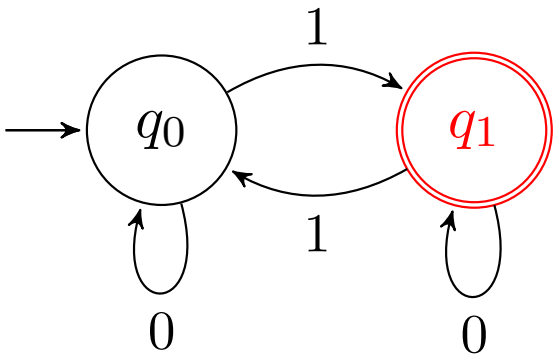
$$x_i = \sum_{j=1}^{k} a_{i_j} x_{i_j} \quad + \epsilon \text{ si } q_i \text{ est final}$$

Remarque : Si $k = 0$, la somme se réduit à $\emptyset$ et on a alors
$x_i = \emptyset$ si $q_i \notin F$ ou $x_i = \epsilon$ si $q_i \in F$

Théorème (Admis)

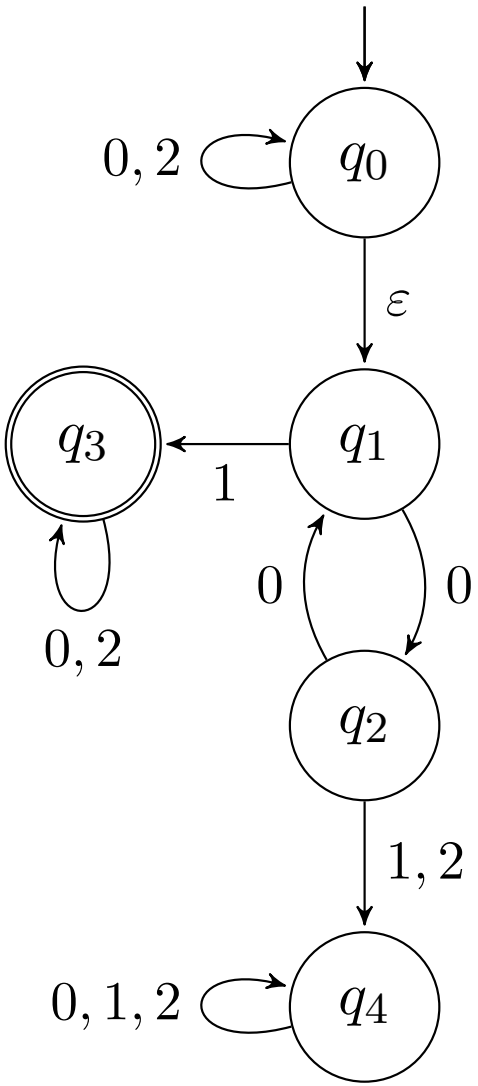
Pour tout i, la plus petite solution de l'équation associée à q_i représente le langage reconnu par l'automate dont q_i est l'unique état initial.

Exemple



$$\begin{cases} x_0 &= 0x_0 + 1x_1 \\ x_1 &= 0x_1 + 1x_0 + \epsilon \end{cases}$$

Exercice



Donner le système associé à cet automate.

$$\begin{cases} x_0 = (0+2)x_0 + \varepsilon x_1 \\ x_1 = 0x_2 + 1x_3 \\ x_2 = 0x_1 + (1+2)x_4 \\ x_3 = (0+2)x_3 + \epsilon \\ x_4 = (0+1+2)x_4 \end{cases}$$

Résolution des systèmes d'équations

Lemme d'Arden

Soient A et B des langages, et considérons l'équation $X = AX + B$.
Alors :

- A^*B est la plus petite solution de cette équation
- Si $\varepsilon \notin A$, c'est l'unique solution

Preuve

- A^*B est une solution

$$\begin{aligned} A(A^*B) + B &= A^+B + B \\ &= A^+B + \epsilon B \\ &= (A^+ + \epsilon)B \\ &= A^*B \end{aligned}$$

Résolution des systèmes d'équations

Preuve (suite)

- A^*B est la plus petite solution

Soit C une solution : on a $C = AC + B$.
Donc $B \subseteq C$ et $AC \subseteq C$
Donc $AB \subseteq AC \subseteq C$
Donc $A^2B \subseteq AC \subseteq C$
$\vdots$ Preuve par récurrence sur k (exercice)
Donc $\forall k \geqslant 0,\ A^kB \subseteq C$
Donc $\bigcup_{k\geqslant 0} A^kB \subseteq C$
Or $\bigcup_{k\geqslant 0} A^kB = \left(\bigcup_{k\geqslant 0} A^k\right) B = A^*B$
Ainsi $A^*B \subseteq C$

Résolution des systèmes d'équations

Preuve (suite)

- Si $\varepsilon \notin A$ alors A^*B est l'unique solution.

  Soit C une solution, montrons que $C = A^*B$.
  Comme A^*B est la plus petite solution, on a déjà $A^*B \subseteq C$. Il suffit donc de montrer que $C \subseteq A^*B$.
  Par l'absurde : on suppose que $\exists w \in C$ tel que $w \notin A^*B$.
  Supposons w de longueur minimale :

  $$\text{si } w' \in C \text{ et } |w'| < |w| \text{ alors } w' \in A^*B.$$

  Par hypothèse, $C = AC + B$, donc $w \in B$ ou $w \in AC$.

  - Si $w \in B$ alors $w \in A^*B$, **absurde**
  - Sinon $w = w_1 w_2$, où $w_1 \in A$ et $w_2 \in C$
    Par hypothèse, $w_1 \neq \varepsilon$ (car $\varepsilon \notin A$), donc $|w_2| < |w|$, d'où $w_2 \in A^*B$.
    Mais dans ce cas, on a $w = w_1 w_2 \in AA^*B \subseteq A^*B$
    Contradiction

Autre démonstration par le lemme de commutation

Lemme (Lemme de commutation, rappel)

Pour $k \in \{1,2\}$, soit f_k applications continues de $\mathcal{P}(E_k) \to \mathcal{P}(E_k)$
Soit g application continue de $\mathcal{P}(E_1) \to \mathcal{P}(E_2)$ avec

$$g(\emptyset) = \emptyset \qquad \text{et} \qquad g \circ f_1 = f_2 \circ g$$

Alors, on a :

$$\mu(f_2) = \bigcup_{i\in\mathbb{N}} f_2^i(\emptyset) = g(\bigcup_{i\in\mathbb{N}} f_1^i(\emptyset)) = g(\mu(f_1))$$

On pose $\begin{cases} f_1(X) \stackrel{\text{def}}{=} A.X \cup \{\epsilon\} \\ f_2(Y) \stackrel{\text{def}}{=} A.Y \cup B \\ g(X) \stackrel{\text{def}}{=} X.B \end{cases}$ toutes de $\mathcal{P}(V^*)$ dans $\mathcal{P}(V^*)$

Elles sont continues, $g(\emptyset) = \emptyset$ et $g(f_1(X)) = A.X.B \cup B = f_2(g(X))$.
Par définition, $A^* = \bigcup_{i\in\mathbb{N}} f_1^i(\emptyset)$ donc d'après le lemme de commutation,

$$\mu(f_2) = \bigcup_{i\in\mathbb{N}} f_2^i(\emptyset) = g\Big(\bigcup_{i\in\mathbb{N}} f_1^i(\emptyset)\Big) = g(A^*) = A^*.B$$

Remarques importantes

Questions

- Quelle est la (plus petite) solution de l'équation $X = AX + \epsilon$?
  - $X = A^*$
- Quelle est la (plus petite) solution de l'équation $X = AX$?
  - $X = \emptyset$
- Si l'automate a deux états initiaux q_j et q_k ?
  - Renvoyer $Sol(x_j) + Sol(x_k)$.
- Si $\varepsilon \in A$, quelles autres solutions de $X = AX + B$ y a-t-il ?
  - Tout langage de la forme A^*C avec $B \subseteq C$ est solution.
- Qu'obtient-on pour l'équation $X = XA + B$?
  - La plus petite solution est BA^*.

Exercice

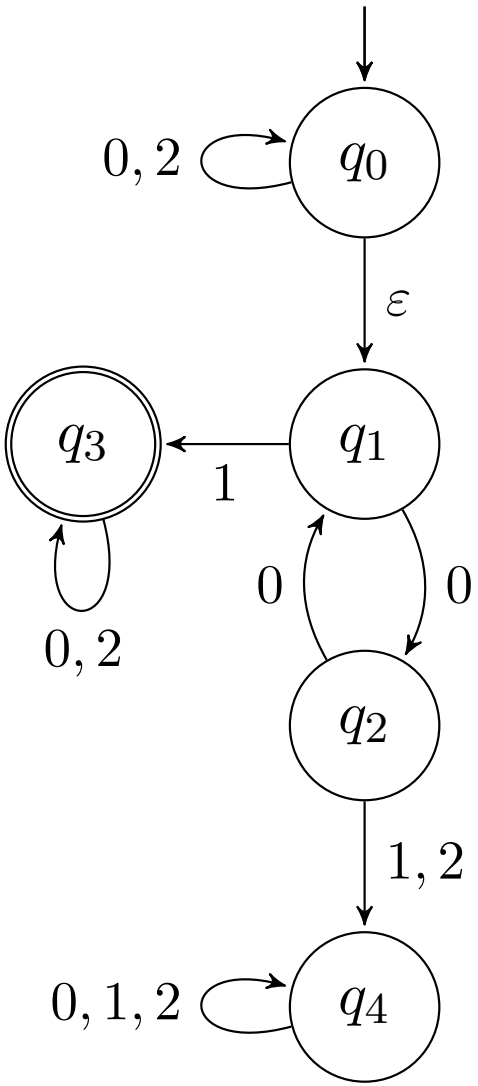
Résoudre le système d'équations suivant (q_0 état initial) :

$$\begin{cases} x_0 = (0+2)x_0 + x_1 \\ x_1 = 0x_2 + 1x_3 \\ x_2 = 0x_1 + (1+2)x_4 \\ x_3 = (0+2)x_3 + \epsilon \\ x_4 = (0+1+2)x_4 \end{cases} \qquad \begin{cases} x_0 = (0+2)x_0 + x_1 \\ x_1 = 00x_1 + 1(0+2)^* \\ x_2 = 0x_1 \\ x_3 = (0+2)^* \\ x_4 = \emptyset \end{cases}$$

$$\begin{cases} x_0 = (0+2)^*(00)^*1(0+2)^* \\ x_1 = (00)^*1(0+2)^* \\ x_2 = 0x_1 \\ x_3 = (0+2)^* \\ x_4 = \emptyset \end{cases}$$

donc $\mathcal{L}(A) = \mathcal{L}(A_{q_0}) = (0+2)^*(00)^*1(0+2)^*$

« Vérification » a posteriori



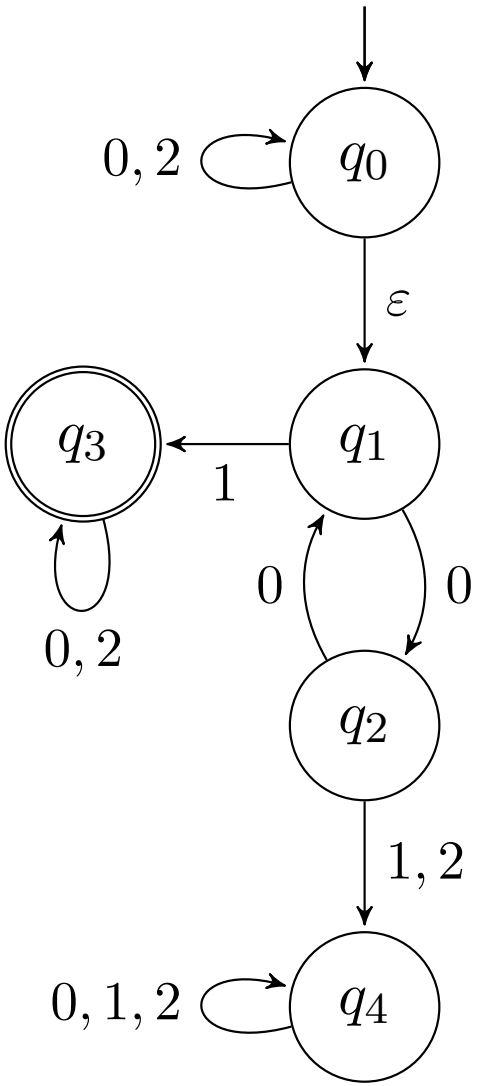
$$\mathcal{L}(A) \;=\; (0+2)^*(00)^*1(0+2)^*$$

On peut s'assurer que l'automate « reconnaît » l'expression régulière calculée.

Remarque : on peut trouver différentes expressions régulières en fonction de l'ordre dans lequel les équations sont résolues.

On peut faire correspondre la substitution d'une variable et la suppression de l'état correspondant dans la méthode graphique.

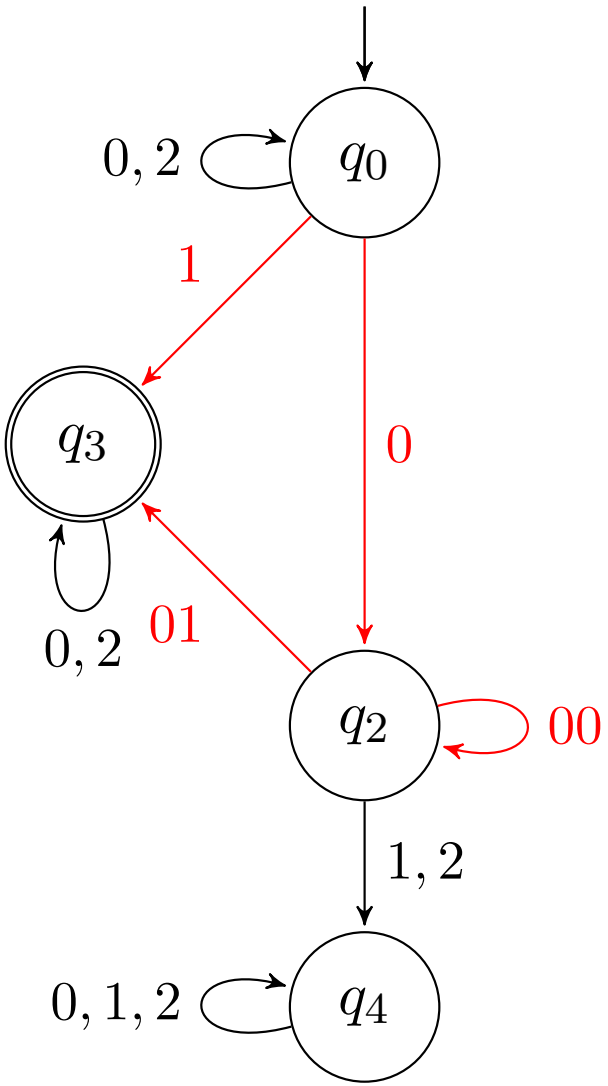
Correspondance entre versions graphique et par équation



Avant suppression :

$$\begin{cases} x_0 = (0+2)x_0 + x_1 \\ x_1 = 0x_2 + 1x_3 \\ x_2 = 0x_1 + (1+2)x_4 \\ x_3 = (0+2)x_3 + \epsilon \\ x_4 = (0+1+2)x_4 \end{cases}$$

Correspondance entre versions graphique et par équation



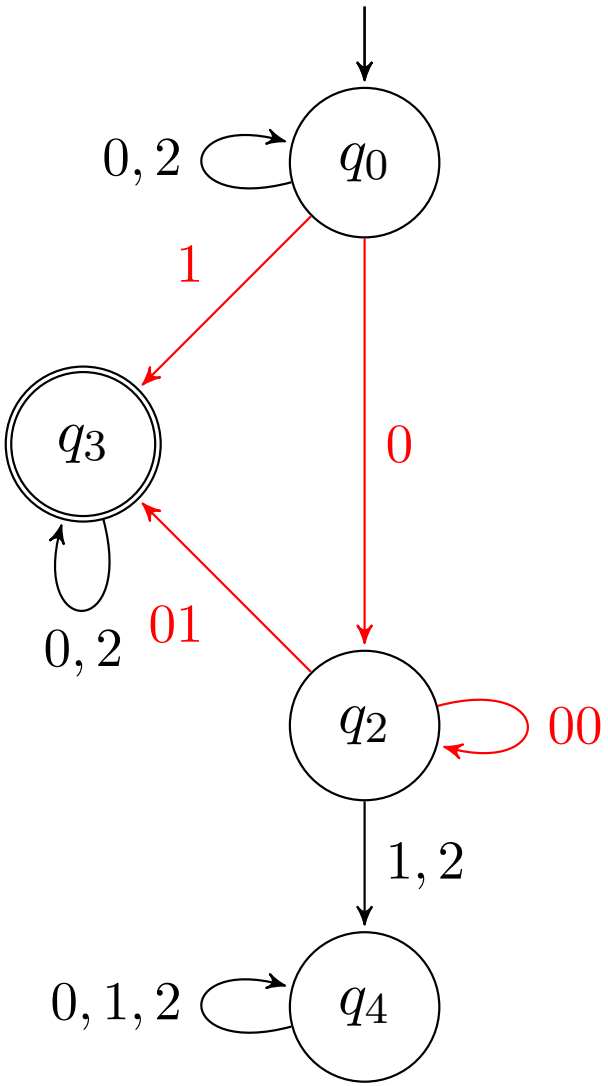
Avant suppression :

$$\begin{cases} x_0 = (0+2)x_0 + x_1 \\ x_1 = 0x_2 + 1x_3 \\ x_2 = 0x_1 + (1+2)x_4 \\ x_3 = (0+2)x_3 + \epsilon \\ x_4 = (0+1+2)x_4 \end{cases}$$

Après suppression de q_1 :

$$\begin{cases} x_0 = (0+2)x_0 + 0x_2 + 1x_3 \\ x_2 = 00x_2 + 01x_3 + (1+2)x_4 \\ x_3 = (0+2)x_3 + \epsilon \\ x_4 = (0+1+2)x_4 \end{cases}$$

Correspondance entre versions graphique et par équation



Avant suppression :

$$\begin{cases} x_0 = (0+2)x_0 + x_1 \\ x_1 = 0x_2 + 1x_3 \\ x_2 = 0x_1 + (1+2)x_4 \\ x_3 = (0+2)x_3 + \epsilon \\ x_4 = (0+1+2)x_4 \end{cases}$$

Après suppression de q_1 :

$$\begin{cases} x_0 = (0+2)x_0 + 0x_2 + 1x_3 \\ x_2 = 0(0x_2 + 1x_3) + (1+2)x_4 \\ x_3 = (0+2)x_3 + \epsilon \\ x_4 = (0+1+2)x_4 \end{cases}$$

Implémentation des automates

Pour les AFD complets

AFD = cas facile

- jamais de choix à faire (déterminisme)
- toujours défini (complétude)

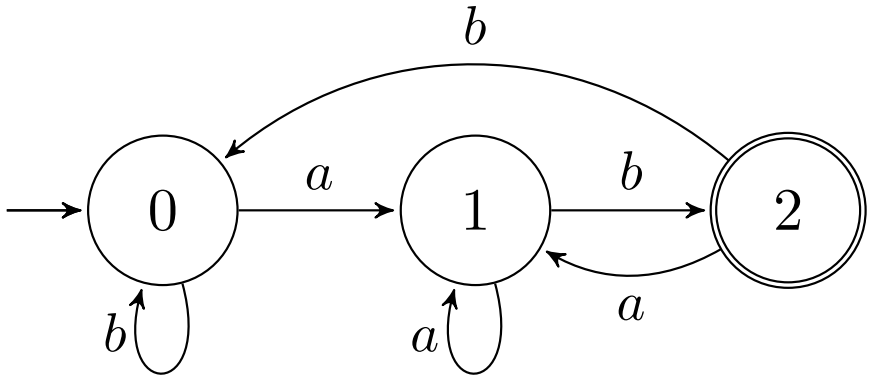
$\rightsquigarrow$ existence + unicité du chemin

$\rightsquigarrow$ important pour définir l'état d'un système

3 méthodes :

- par table
- par tests
- par fonctions

Exemple (filé) : $L = V^*.\{ab\}$



Interface

Comment est représenté le mot ?

- par une chaîne de caractères (tout est connu d'un coup)
- par une source de caractères (arrivée au compte-goutte)

```
# initialisation de la source
init_input(...)

# acces au caractere suivant
next_char()
```

Exemples

```
import sys
in_stream = sys.stderr

def init_input():
    global in_stream = sys.stdin

def next_char():
    global in_stream
    return in_stream.read(1)
```

```
word = ""
index = 0

def init_input(w):
    global word = w
    global index = -1

def next_char():
    global index+=1
    return global word[index]
```

Automates et fin de mot

```
def exec():
    state = i
    init_input()
    ch = next_char()
    while ch != '$':
        state = step(state, ch)
        ch = next_char()
    return state ∈ accepting
```

avec :

- i l'état initial
- step la fonction de transition
- accepting les états acceptants
- $ le caractère de fin

Comment reconnaître la fin du mot ?

- Connaître la taille du mot (exemple : chaînes Python, OCaml)
  $\rightsquigarrow$ possible uniquement si toute l'entrée est disponible à la fois
- Ajouter un caractère spécial $\notin V$ de fin de mot ('\0', EOL/EOF)
  $\rightsquigarrow$ marche dans tous les cas (transmission sur réseau)

Ici, on choisit d'ajouter un caractère spécial $(ou END)

Implémentation par table ou par tests

Par table

Idée : la fonction de transition est une matrice

```
def A = {0: {'a': 1, 'b': 0},
         1: {'a': 1, 'b': 2},
         2: {'a': 1, 'b': 0}}

def step(state, ch):
    return A.transitions[state][ch]
```

$\rightsquigarrow$ l'automate est une donnée

Coût = lecture mémoire

Par tests

Idée : la fonction de transition est un bout de code

```
def step(state, ch):
    if state == 0:
        if ch == 'a':
            return 1
        elif ch == 'b':
            return 0
    elif state == 1:
        ⋮
```

$\rightsquigarrow$ l'automate est un programme

Coût = tests + sauts

Implémentation par fonctions

Idée : chaque état est une fonction

$\rightsquigarrow$ pas de boucle **while** ni de fonction step

```
def state0():
    ch = next_char()
    if ch == 'a':
        return state1()
    elif ch == 'b':
        return state0()
    elif ch == '$':
        return False
```

```
def state1():
    ch = next_char()
    if ch == 'a':
        return state1()
    elif ch == 'b':
        return state2()
    elif ch == '$':
        return False
```

```
def state2():
    ch = next_char()
    if ch == 'a':
        return state1()
    elif ch == 'b':
        return state0()
    elif ch == '$':
        return True
```

- plus modulaire que la boucle while
- permet de faire du calcul
- Coût = appel de fonction

réduit par des sauts (appel terminal)

automate = ensemble de fonctions
lecture = appeler l'état initial
état courant = la fonction qui s'exécute

```
def exec_v2():
    init_input()
    return state0()
```

Quelle méthode pour les AFND ?

- Par fonction

  une fonction est spécifique à un état...

  Exponentiel avec plusieurs états ! (nb de chemins)

- Par matrice ou tests

  variable state = ensemble d'états

  + itération sur state pour calculer l'état suivant

  $\rightsquigarrow$ deux variables : state, new_state

  $\rightsquigarrow$ boucles sur state paralléliser les tests / lectures mémoire

```
def step(A, state, ch):
    new_state = set.empty()
    for q in state:
        new_state.add(
            A.transitions[
                q, ch])
    return new_state
```

```
def exec(A):
    state = A.init
    init_input()
    ch = next_char()
    while (ch != '$'):
        state = step(A, state, ch)
        ch = next_char()
    return A.accepting.inter(state)
```

Comparaison AFD/AFND

AFD

- exécution très rapide $O(|w|)$ (jamais de choix à faire)
- plus gros que AFND (exponentiellement !)

Cas d'utilisation

- Vitesse exigée
- Beaucoup d'utilisation
- Construction de l'automate à l'avance

$\rightsquigarrow$ ex : compilateur

AFND

- exécution plus lente $O(|w| \cdot |Q|)$ (ensembles d'états)
- plus compacts que AFD

Cas d'utilisation

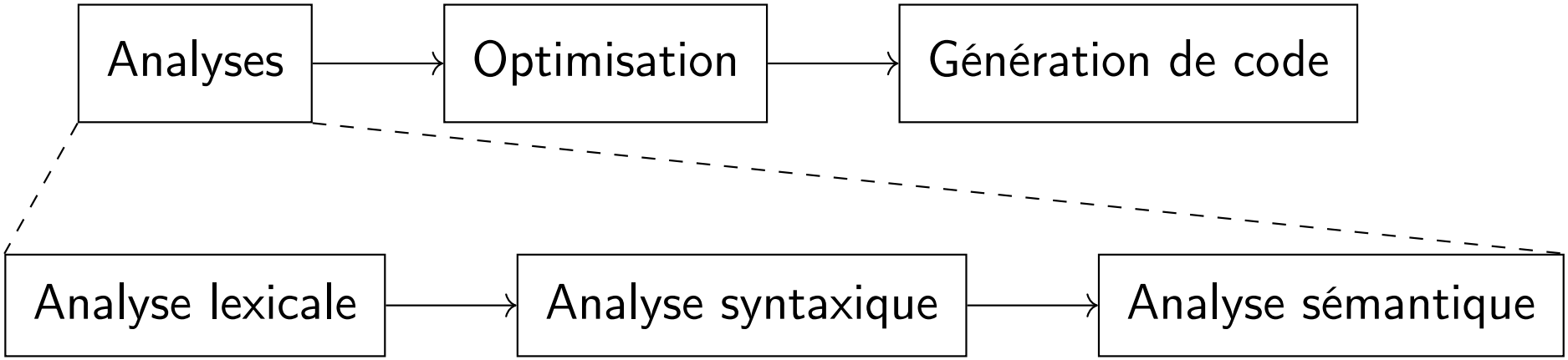
- Contraintes d'espace
- Utilisation unique (ou faible)
- Construction de l'automate à l'utilisation
- Facile dans les circuits

$\rightsquigarrow$ ex : expressions régulières

Choix AFD/AFND = compromis espace/temps

Exemple d'utilisation : analyse lexicale

Première partie d'un compilateur : reconnaître les programmes corrects



Outils :

1. Analyse lexicale : expressions régulières
2. Analyse syntaxique (grammaticale) : grammaires
3. Analyse sémantique : plus varié

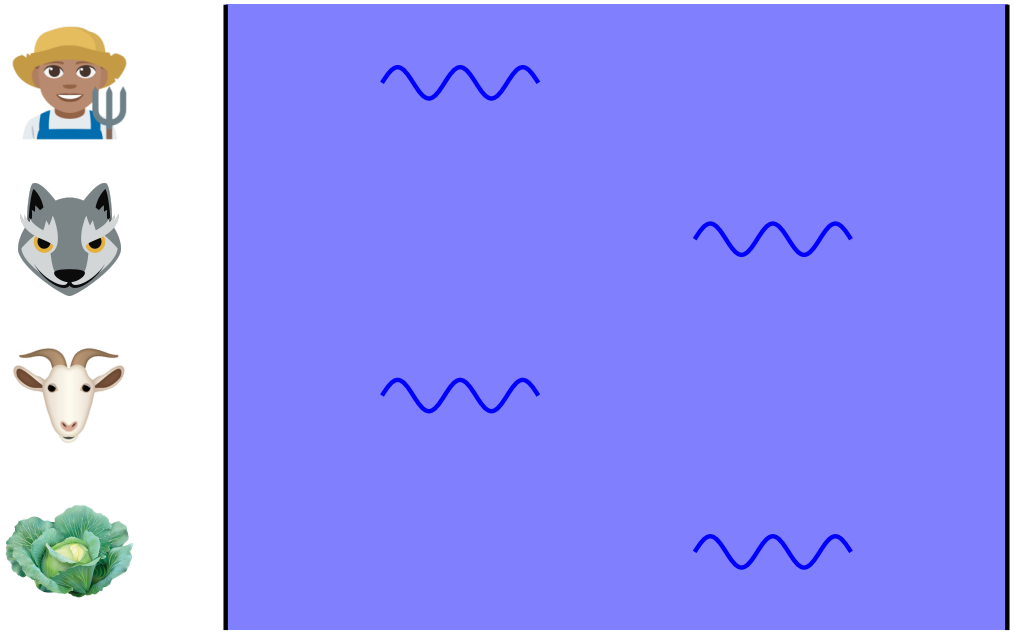
En projet : calculatrice sur des entiers

$\rightsquigarrow$ Analyses lexicale et syntaxique

Application des automates

Application 1 : statégie dans un jeu

Jeu du fermier (cf. exercice 28 du recueil de TD)



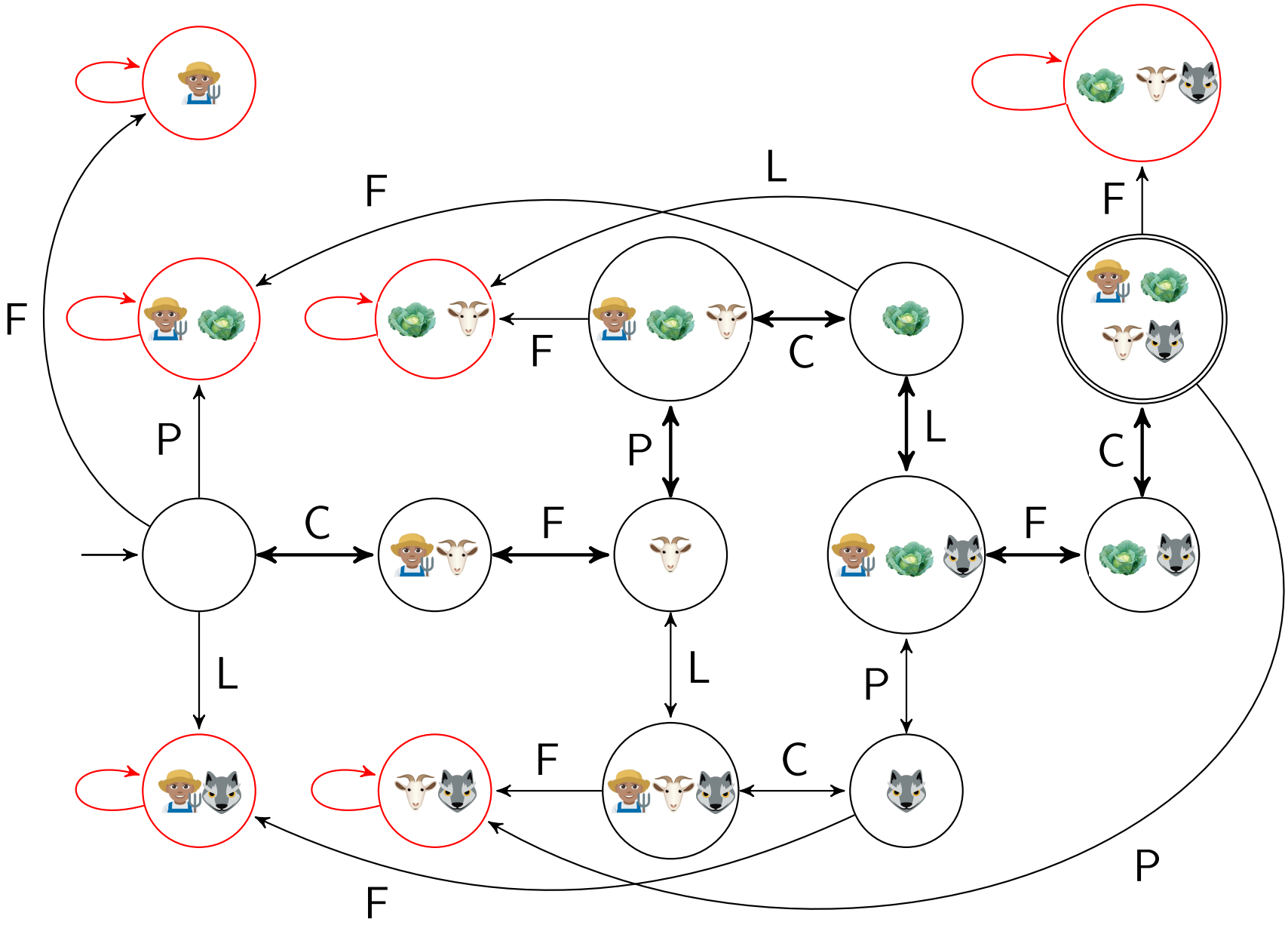
Exercice

1. Représenter le problème par un automate, en précisant le vocabulaire choisi.
2. Comment déterminer une stratégie à partir de l'automate ? Quelles sont les stratégies optimales ?

Solution

- États : positions (quelle rive) de  ,  ,  , 
  $\rightarrow$ 16 états
- Vocabulaire = mouvements possibles :
  - F : le fermier traverse seul
  - L : le fermier traverse avec le loup
  - C : le fermier traverse avec la chèvre
  - P : le fermier traverse avec le chou
- États acceptants = tous sont sur l'autre rive
- Sans passer par les états perdants
  $\rightsquigarrow$ États perdants = états puits
- Stratégie gagnante = mot accepté par cet automate
  Optimum = 7 coups

Automate



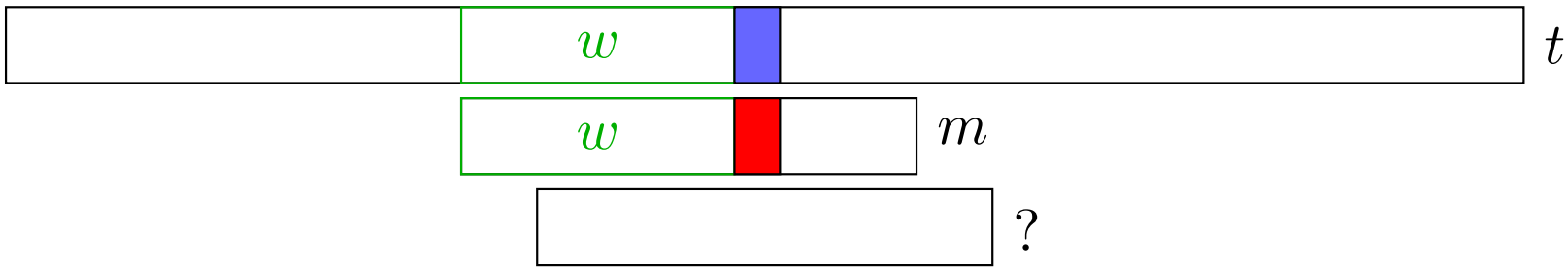
Application 2 : algo KMP

Contexte : recherche d'un motif m dans un texte t

Contraintes :

- construction de l'automate linéaire (en $|m|$)
- parcours du texte en ligne = caractère par caractère
  pas de retour en arrière possible
  Complexité linéaire en $|t|$

Idée : décaler le motif de « juste ce qu'il faut » en cas d'erreur
sans rater d'occurence de m



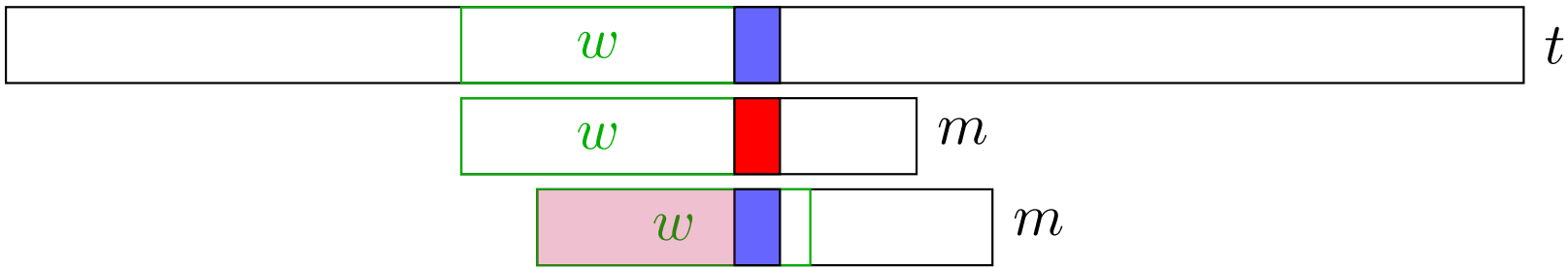
Application 2 : algo KMP

Contexte : recherche d'un motif m dans un texte t

Contraintes :

- construction de l'automate linéaire (en $|m|$)
- parcours du texte en ligne = caractère par caractère
  pas de retour en arrière possible
  Complexité linéaire en $|t|$

Idée : décaler le motif de « juste ce qu'il faut » en cas d'erreur
sans rater d'occurence de m



Que peut-on dire de par rapport à w ?
C'est un préfixe (strict) et un suffixe de w !

Calcul des bords d'un mot

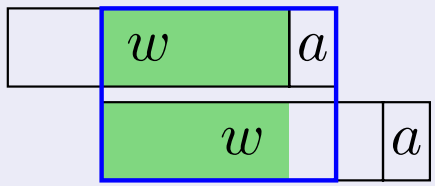
Définition (Bord)

Le bord d'un mot $w \neq \varepsilon$ est son plus grand préfixe strict qui en est également un suffixe. On le note $\varphi(w)$.

Pourquoi strict ?

Calcul de $\varphi(w)$

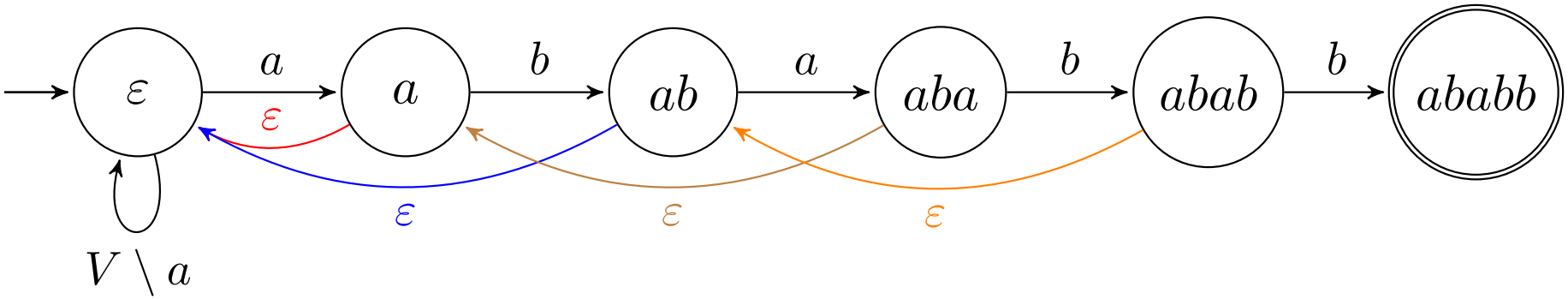
- pour $a \in V$, $\varphi(a) = \varepsilon$
- pour $w \in V^+, a \in V$, $\varphi(wa) = \begin{cases} \varphi(w)a & \text{si } \varphi(w)a \text{ préfixe de } w \\ \varphi(\varphi(w)a) & \text{sinon} \end{cases}$



Idée : si problème après w, réessayer avec $\varphi(w)$!

Exemple : recherche de $ababb$

Recherche de $m = ababb$ dans $t = abaababb$.



Calcul de $\varphi(w)$ pour w préfixe strict de m :

$\varphi(a) = \varepsilon$ $\qquad$ $\varphi(ab) = \varphi(\varphi(a)b) = \varphi(\varepsilon b) = \varepsilon$

$\varphi(aba) = \varphi(ab)a = a$ $\qquad$ $\varphi(abab) = \varphi(aba)b = ab$

Complexité de la construction : $O|m|$ en temps et en espace

Taille de l'AFD complet : $O(|m| \times |V|)$ $\qquad$ $|V|$ transitions par état

Complexité de la lecture : $O(|t|)$ $\qquad$ analyse amortie