# COMPILATION

## ANALYSE LEXICALE

EMSI - 3$^{ÈME}$ IIR 2016/2017

Prof. M. D. RAHMANI

# Conception d'un analyseur lexical



1. Rôle d'un analyseur lexical,
2. Terminologie,
3. Spécification des unités lexicales,
4. Reconnaissance des unités lexicales,
5. Automates à états finis déterministes,

# 1- **Le rôle d'un analyseur lexical**:



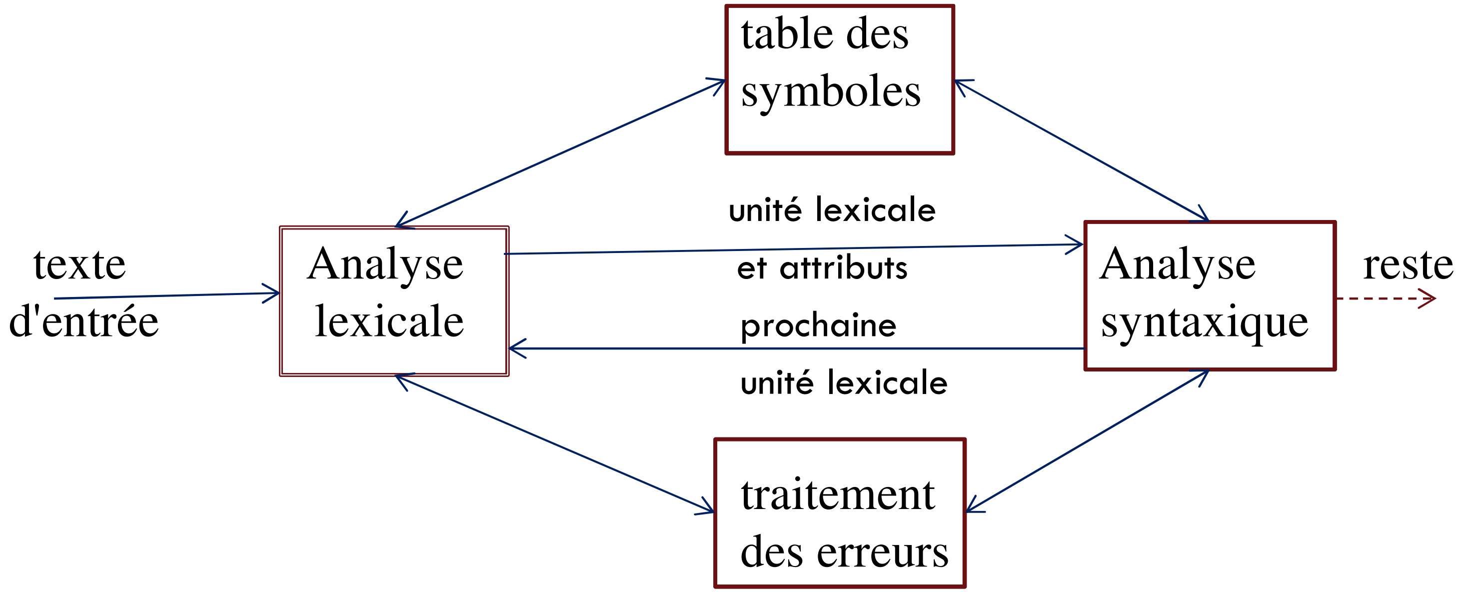
*L'analyseur lexical est chargé de lire le texte d'entrée, caractère par caractère, de la gauche vers la droite et isoler les mots et leur classe.*

De plus, il doit:

- éliminer les blancs (espaces, tabulations, fin de lignes) et les commentaires.
- détecter les erreurs et associer des messages d'erreurs.

# 1- **Le rôle d'un analyseur lexical:**






Interaction entre analyseur lexical et analyseur syntaxique.

## 2- Terminologie



- ✓ **Unité lexicale:** est un symbole terminal de la grammaire du langage.

- ✓ **Modèle:** est une règle qui décrit un ensemble de chaînes associées à la même unité lexicale.

- ✓ **Lexème:** est une suite de caractères du texte d'entrée qui concorde avec le modèle.

Exemple: **35** est un **lexème** (un mot) qui appartient à l'**unité lexicale** (la classe) **nombre**.

# 2- **Terminologie**



Remarques:

Dans de nombreux langages, les classes suivantes couvrent la plupart des unités lexicales:

1- Une unité lexicale pour chaque mot clé.

2- Des unités lexicales pour les opérateurs, soit individuellement, soit par classes.

3- Une unité lexicale pour les identificateurs (noms de variables, fonctions, tableaux, structures...).

4- Une ou plusieurs unités lexicales pour les nombres et les chaînes.

5- Une unité lexicale pour chacun des signes de ponctuation, tels que les parenthèses gauche et droite, la virgule , le point-virgule...

# 3- Spécification des unités lexicales



**3.1- Chaînes et langages:**

Définitions générales:

- *Un alphabet Σ ou une classe de caractères définit un ensemble fini de symboles.*

Exemples: {0,1} : l'alphabet binaire

ASCII : l'alphabet informatique

- *Une chaîne ou un mot sur un alphabet Σ est une séquence finie de symboles extraits de cet ensemble.*

# 3- **Spécification des unités lexicales**



Soit une chaîne ***s*** :

- **préfixe de *s*** *est une chaîne obtenue en supprimant un nombre quelconque (même nul) de symboles à la fin de* ***s****.*
- **suffixe de *s*** *est une chaîne obtenue en supprimant un nombre quelconque (même nul) de symboles au début de* ***s****.*
- **sous-chaîne de *s*** *est une chaîne obtenue en supprimant un préfixe et un suffixe.*
- **sous-suite de s** *est une chaîne obtenue en supprimant un nombre quelconque (même nul) de symboles <u>non nécessairement consécutifs</u>.*
- **Un langage** *est un ensemble quelconque de chaînes construites sur un alphabet fixé.*

# 3- Spécification des unités lexicales



**3.2- <u>Opérations sur les langages</u>:**

Soit ***L*** et ***M*** deux langages:

- Union de ***L*** et ***M*** : $L \cup M = \{ \forall s \mid s \in L \text{ ou } s \in M \}$

- Concaténation de ***L*** et ***M*** : $LM = \{ st \mid s \in L \text{ et } t \in M \}$

- Fermeture de Kleene de ***L*** : $L^* = \bigcup_{i=0}^{\infty} L^i$

  ***L**** dénote un nombre quelconque (même nul) de concaténation de ***L***.

  On note $L^0 = \{\varepsilon\}$

- Fermeture positive de ***L*** : $L^+ = \bigcup_{i=1}^{\infty} L^i$

# 3- **Spécification des unités lexicales**



Exemples:

Soit $L = \{A,B,\ldots,Z\} \cup \{a,b,\ldots,z\}$ *et* $C = \{0,1,\ldots9\}$

A partir de *L* et *C*, nous pouvons produire d'autres langages.

- ***L U C*** *:* ensemble des lettres et chiffres,
- ***LC*** *:* ensemble des chaînes constituées d'une lettre suivie d'un chiffre,
- $L^4$ *:* ensemble des chaînes constituées de 4 lettres,
- $C^+$ *:* ensemble des entiers naturels,
- $L(LUC)^*$ *:* ensemble des chaînes constituées d'une lettre suivie d'une chaîne de lettres et de chiffres ou d'une chaîne vide.

# 3- **Spécification des unités lexicales**



## 3.3- <u>Expressions régulières:</u>

*Une expression régulière est une notation qui permet de décrire un ensemble (une classe) de chaînes de caractères.*

<u>Exemple:</u> Un nombre entier non signé est une chaîne constituée d'une suite de chiffres, au moins un.

L'expression régulière associée est: **`(chiffre)+`**

**+** *est un opérateur unaire post-fixe qui veut dire un ou plusieurs fois.*

## 3- **Spécification des unités lexicales**



Les règles qui définissent les expressions régulières sur un **alphabet Σ** sont:

- **ε** est une expression régulière qui dénote **{ε}** c-à-d l'ensemble dont le seul élément est la **chaîne vide ε**.
- si **a** est un symbole de l'alphabet Σ, alors **a** est une expression régulière qui dénote **{a}**, c-à-d l'ensemble constitué de la chaîne **a**.

# 3- **Spécification des unités lexicales**



soit ***r*** et ***s*** deux expressions régulières qui dénotent les langages L(r) et L(s), alors:

- **(r) | (s)** est une expression régulière qui dénote **(L(r)) U (L(s))**.
- **(r)(s)** est une expression régulière qui dénote **(L(r))(L(s))**.
- **(r)*** est une expression régulière qui dénote **(L(r))***.

- *Les langages dénotés par les expressions régulières sont appelés **langages réguliers**.*

# 3- Spécification des unités lexicales



**<u>Exemples:</u>**

`a|b*c` : les chaînes constituées, soit d'un **`a`**, ou d'un nombre quelconque, éventuellement nul, de la lettre **`b`** suivie de la lettre **`c`**.

`a | b = {a,b}`

`(a|b)(a|b) = {aa,ab,ba,bb}`

**<u>Définition:</u>**

Si deux expressions **`r`** et **`s`** dénotent le même langage, on dit qu'elles sont *<u>équivalentes</u>* et on écrit: **`r=s`**

**<u>Exemple</u>:** `(a|b)= (b|a)`

# 3- Spécification des unités lexicales



**Propriétés algébriques sur les expressions régulières:**

Soit **r, s** et **t** des expressions régulières.

**r|s = s|r** : *l'opérateur* | *(ou) est commutatif.*

**r|(s|t) = (r|s)|t** : *l'opérateur* | *est associatif.*

**(rs)t = r(st)** : *la concaténation est associative.*

**r(s|t)=rs|rt** : *la concaténation est distributive par rapport au* |

**εr = r ε = r** : *ε est l'élément neutre de la concaténation.*

**r* =(r|ε)+** : *ε est inclus dans une fermeture.*

**r** = r*** : ** est idempotent*

Remarque: la chaîne vide $\varepsilon = s^0$

# 3- **Spécification des unités lexicales**



**<u>Notations:</u>**

**`*`** est un opérateur unaire poste-fixe qui veut dire zéro, un ou plusieurs fois.

**`+`** est un opérateur unaire poste-fixe qui veut dire un ou plusieurs fois.

$$r+ = r\ r* = r*r$$

$$r* = r+|\varepsilon$$

**`?`** est un opérateur unaire poste-fixe qui veut dire zéro ou une fois.

$$r? = r|\varepsilon$$

**`[a-z]`** désigne un élément (une lettre) de cette classe.

**exemple:** [a-z] = a|b|c...|z

# 3- **Spécification des unités lexicales**



**<u>Conventions:</u>**

**1-** L' opérateur unaire poste-fixe ***** a la plus haute priorité et est associatif à gauche.

**2-** Les opérateurs **+** et **?** ont la même priorité et la même associativité que *****.

**2-** La concaténation a la deuxième priorité et est associative à gauche.

**3-** Le **|** a la plus faible priorité et est associatif à gauche.

Selon ces conventions, **(a)|((b)*(c))** est équivalente à **a|b*c**

# 3- Spécification des unités lexicales



**Exemples d'expressions régulières:**

- Un identificateur: `lettre(lettre|chiffre)*`

  `= [a-zA-Z][a-zA-Z0-9]*`

- Un entier signé ou non: `(+|-)? (chiffre)+`

  `= [+-]?[0-9]+`

- Un nombre décimal: `(+|-)? (chiffre)+ (.(chiffre)+)?`

- Un réel:

  `(+|-)?(chiffre)+(.(chiffre)+)?((e|E)(+|-)?(chiffre)+)?`

  `= [+-]?[0-9]+(.[0-9]+)?((e|E)(+|-)?[0-9]+)?`

# 3- Spécification des unités lexicales



## 3.4- Définitions régulières:

Une définition régulière est une suite de définitions de la forme:

$$\begin{array}{lcl} \mathbf{d_1} & \longrightarrow & \mathbf{r_1} \\ \mathbf{d_2} & \longrightarrow & \mathbf{r_2} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \mathbf{d_n} & \longrightarrow & \mathbf{r_n} \end{array}$$

Chaque $\mathbf{d_i}$ est un nom distinct et chaque $\mathbf{r_i}$ est une expression régulière sur les symboles :

$\Sigma$ U $\{\mathbf{d_1},\mathbf{d_2},\ldots,\mathbf{d_{i-1}}\}$

# 3- Spécification des unités lexicales



**Exemples:**

1- Définition régulière d'un identificateur:

```
lettre      ⟶   A|B|...|Z|a|b|...|z
chiffre     ⟶   0|1|2...|9
id          ⟶   lettre(lettre|chiffre)*
```

2- Définition régulière des entiers signés et non signés:

```
chiffre     ⟶   0|1|2...|9
entier      ⟶   [+|-]?(chiffre)+
```

# 3- Spécification des unités lexicales



3- Définition régulière d'un réel:

L'alphabet $\Sigma$=`{0,1,...,9,.,e,E,+,-}`

Une définition régulière sera:

```
chiffre       →  0|1|2...|9
chiffres      →  (chiffre)+
p_entiere     →  (+|-)? chiffres
p_decimale    →  (.chiffres)?
p-puissance   →  ((e|E)(+|-)? chiffres)?
reel          →  p_entiere p_decimale p_puissance
```

# 4- Reconnaissance des unités lexicales



Soit le fragment de grammaire des instructions conditionnelles:

```
inst     →    si (exp) alors inst
              |si (exp) alors inst sinon inst
              |autre_inst
exp      →    terme operel terme
              |terme
terme    →    id
              |nb
```

Les terminaux de cette grammaire sont:

`si`, `alors`, `sinon`, `(`, `)`, `operel`, `id` et `nb`.

Pour les reconnaître, nous allons d'abord donner les définitions régulières associées.

# 4- **Reconnaissance des unités lexicales**



### 4.1- Définitions régulières des terminaux de la grammaire:

*A noter qu'il faut reconnaitre les blancs aussi pour les ignorer.*

```
delim    →  espace|tabulation|fin_de_ligne
blanc    →  (delim)+
IF       →  si
THEN     →  alors
ELSE     →  sinon
operel   →  <|<=|==|<>|>=|>
id       →  [A-Za-z][A-Za-z0-9]*
nb       →  (+|-)?[0-9]+(.[0-9])?((+|-)?(e|E)[0-9]+)?
```

Remarque: *Les commentaires et les blancs sont traités comme des modèles qui ne retournent aucune unité lexicale.*

# 4- **Reconnaissance des unités lexicales**

**4.2- <u>Diagramme de transition</u>:**

*Un diagramme de transition est un organigramme orienté qui décrit les actions à réaliser par l'analyseur lexical.*

Il est constitué d'états et de transitions entre états, définis par les notations suivantes:

un état

un arc ou une transition

état d'acceptation, c-à-d que le lexème est reconnu.

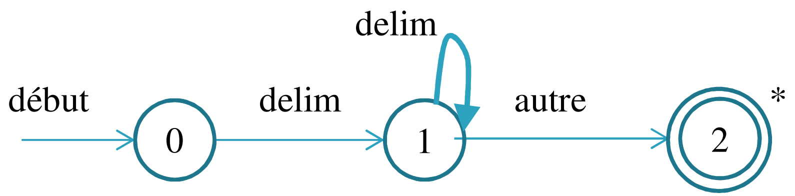
* état d'acceptation avec recul

<u>Remarque</u>: *en pratique, une transition correspond à la consommation d'**<u>un caractère et un seul</u>**.*

# 4- Reconnaissance des unités lexicales



1- Diagramme de transition des `blancs`:




(ne rien retourner)

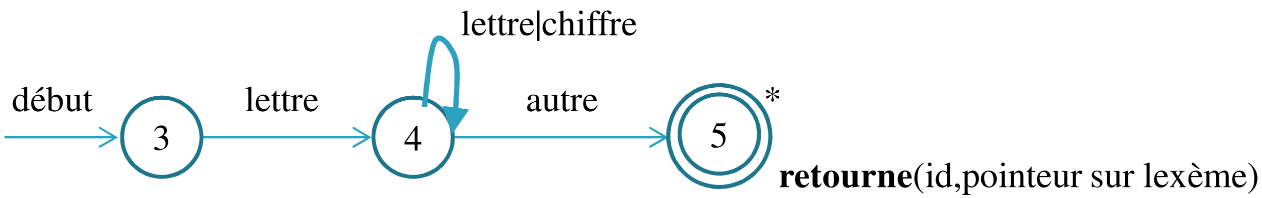
Remarque: *`autre`* veut dire, autre que les autres arcs sortants du même état.

Dans ce cas, *`autre`* veut dire autre qu'un délimiteur.

# 4- Reconnaissance des unités lexicales



2- Diagramme de transition des `identificateurs`:

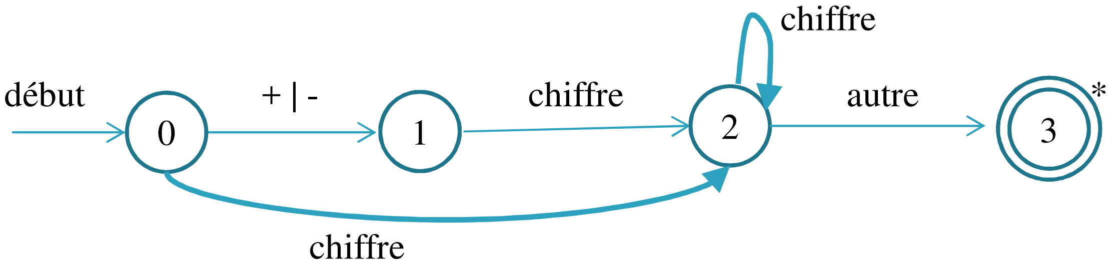



Remarque: `autre` veut dire, autre que ***lettre*** et ***chiffre***.

# 4- **Reconnaissance des unités lexicales**



Diagramme de transition des `entiers signés ou non`:

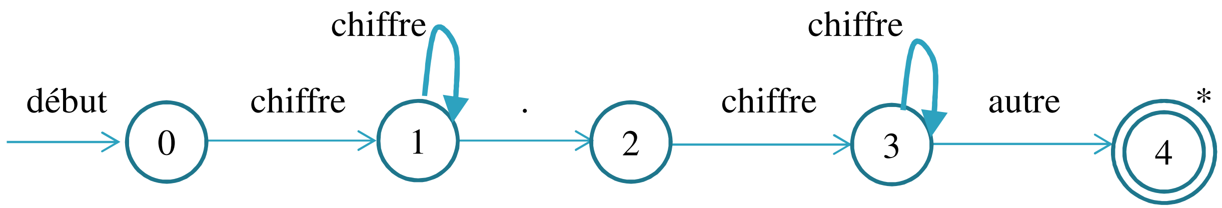



Remarque: *`autre`* veut dire, **autre que les `chiffres`.**

# 4- **Reconnaissance des unités lexicales**



Diagramme de transition des nombres `décimaux non signés`:



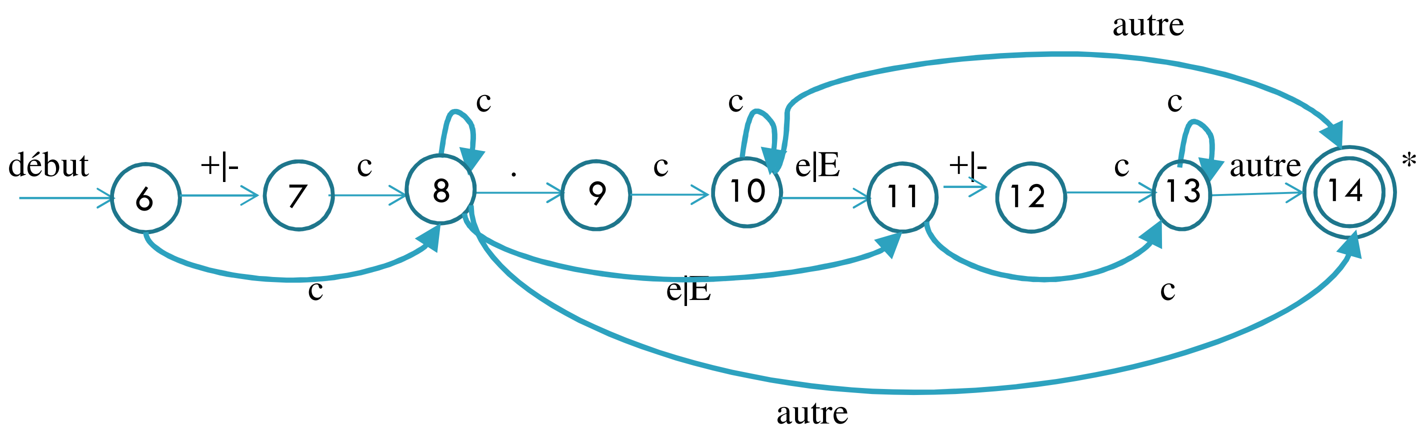

Expression régulière: `(chiffre)+.(chiffre)+`

Remarque: *Nous exigeons par ce diagramme au moins un chiffre après le point.*

# 4- **Reconnaissance des unités lexicales**



3- Diagramme de transition des `nombres réels`:



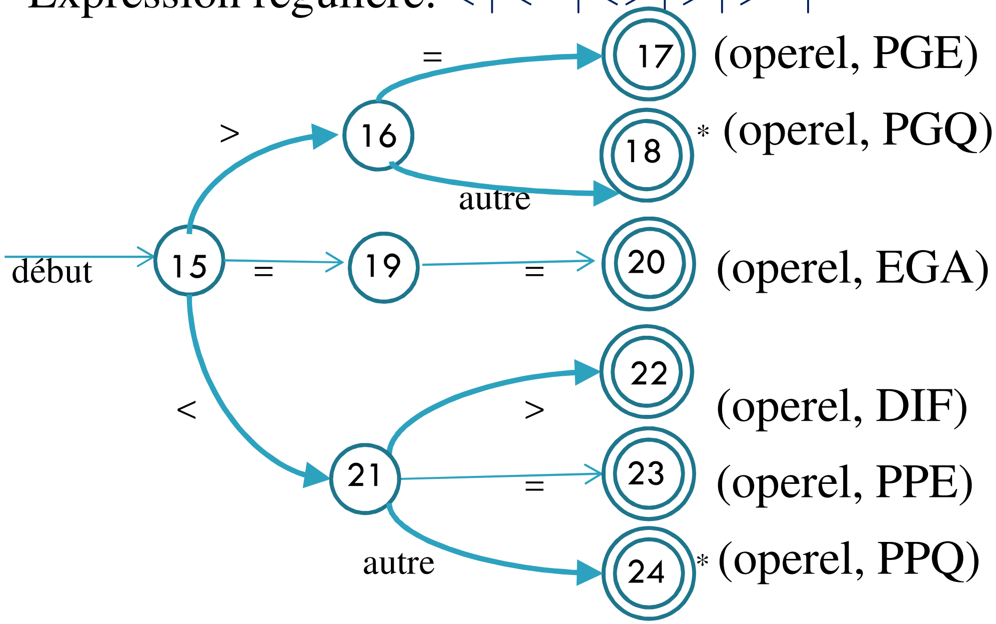

*A l'état* ***`14`*** *d'acceptation avec recul, nous retournons l'unité lexicale* ***`nb`*** *et un pointeur sur le lexème reconnu*

# 4- Reconnaissance des unités lexicales



4- <u>Diagramme de transition des `opérateurs de relation`</u>

Expression régulière: < | <= | <> | > | >= | ==

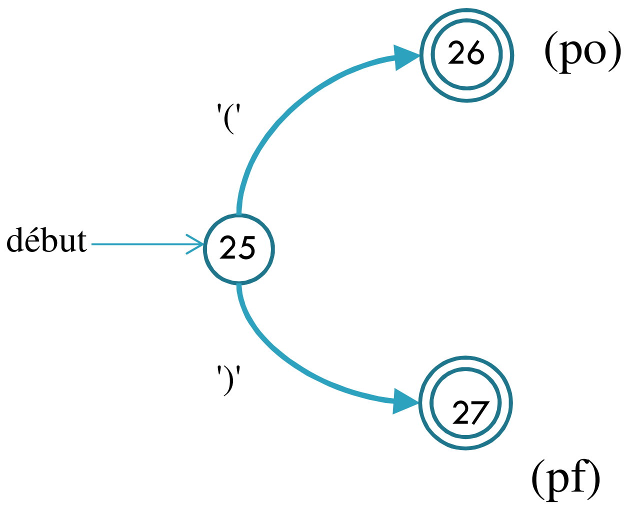


# 4- **Reconnaissance des unités lexicales**



5- <u>Diagramme de transition des `parenthèses`</u>

Expression régulière: `'('|')'`



# 5- Automates à états finis (AEF)



Les automates à états finis sont des graphes orientés à l'image des diagrammes de transition, avec certaines différences:

1- Les automates à états finis sont des ***reconnaisseurs***; ils disent simplement "***oui***" ou "***non***" à propos de chaque chaîne d'entrée.

2- Il y'a 2 types d'automates à états finis:

- Les automates à états finis non déterministes (**AFN**) n'ont aucune restriction sur les étiquettes de leurs arcs.

  Un symbole peut étiqueter plusieurs arcs partant d'un même état, et la chaîne vide ε est une étiquette possible.

- Les automates à états finis déterministes (**AFD**), pour lesquels, ne peuvent pas partir plusieurs transitions du même état avec le même caractère et n'acceptent pas d'ε-transition

# 5- Automates à états finis (AEF)



## 5.1- Automates à états finis non déterministes (AFN):

Un AFN se compose de:

- Un ensemble fini S d'états.
- Un ensemble Σ de symboles d'entrée, l'alphabet du langage.

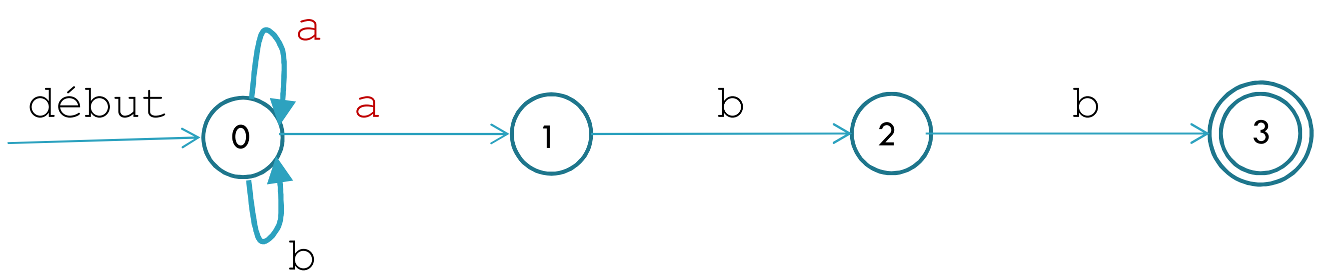
  *On considère que la chaîne vide ε, n'est jamais un membre de Σ.*
- Une fonction de transition qui donne pour chaque état et pour chaque symbole de Σ U {ε}, l'ensemble des états suivants.
- Un état $s_0$ appartenant à S, qui est l'état de départ.
- Un ensemble d'états F, sous-ensemble de S, l'ensemble des états d'acceptation ou états finaux.

# 5- Automates à états finis (AEF)



## 5.1- Automates à états finis non déterministes (AFN):

Exemple: L'AFN qui reconnait le langage défini par l'expression régulière :
**`(a|b)* abb`**




Remarque: *Le non déterminisme ici est associé à deux arcs sortants de l'état 0 avec le même symbole a.*

# 5- Automates à états finis (AEF)



## 5.2- Tables de transition:

Nous pouvons représenter un AFN par une table de transition, dont les lignes correspondent aux états et les colonnes aux symboles d'entrée et à ε.

L'entrée pour un état donné et une entrée donnée est la valeur de la fonction de transition appliquée à ces arguments.

Exemple: soit l'AFN précédant

| Symbole / Etat | a | b | ε |
|---|---|---|---|
| 0 | {0,1} | {0} | - |
| 1 | - | {2} | - |
| 2 | - | {3} | - |
| 3 | - | - | - |

# 5- Automates à états finis (AEF)



## 5.3- Automates à états finis déterministes (AFD):

Un AFD est un cas particulier d'un AFN où:

- il n'y a aucun arc étiqueté par ε,
- pas plus d'un arc avec le même symbole sortant d'un état.

Un AFN est une représentation abstraite d'un algorithme de reconnaissance des chaînes d'un langage.

Un AFD est un algorithme concret de reconnaissance de chaînes.

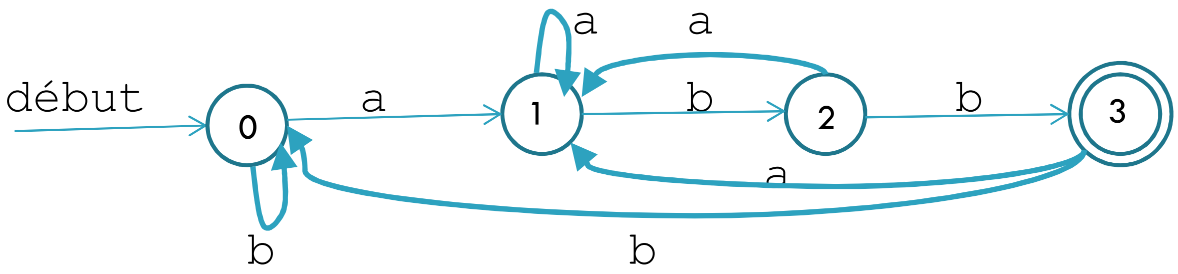
Remarque: *Toute expression régulière et tout AFN peuvent être convertis en un AFD.*

# 5- Automates à états finis (AEF)



<u>Exemple</u>: L'AFD qui reconnait le langage défini par l'expression régulière :

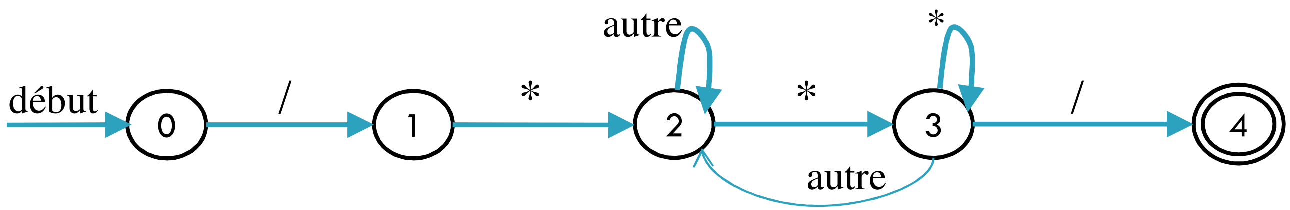
**`(a|b)* abb`**



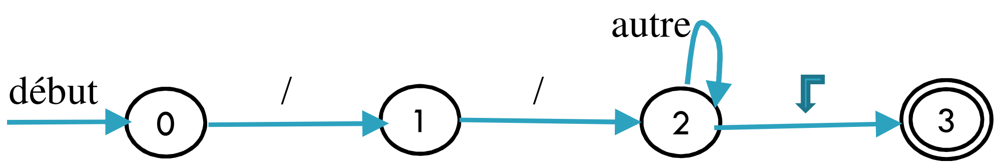
# 5- Automates à états finis (AEF)



<u>Exemple 2</u>: L'automate à états finis déterministe d'un commentaire à la C




<u>Exemple 3</u>: L'automate à états finis déterministe d'un commentaire à la C++

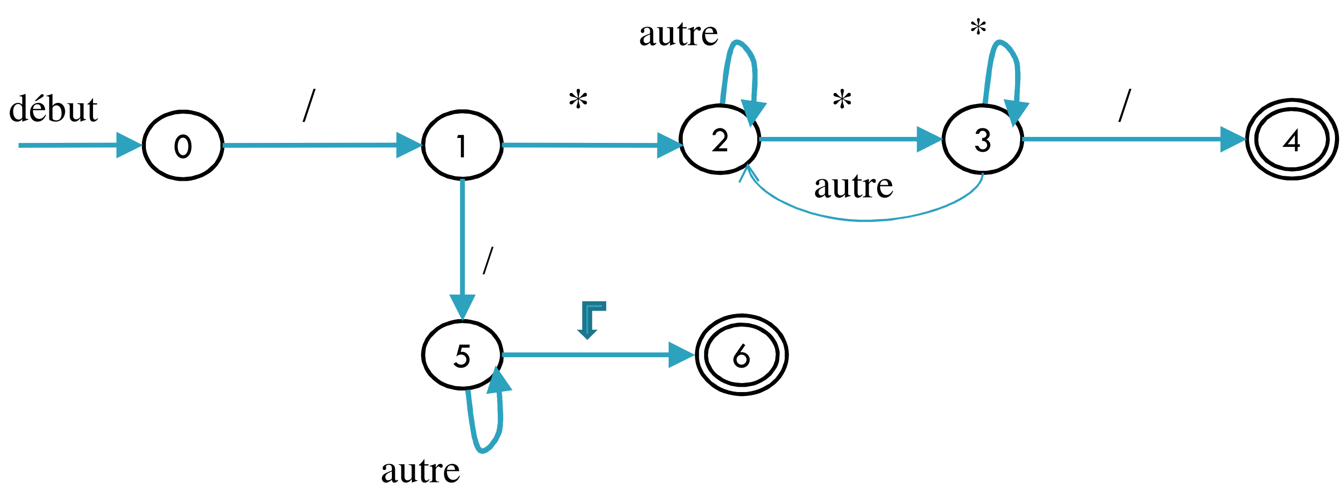



:désigne le retour à la ligne

# 5- Automates à états finis (AEF)



Exemple 4: L'automate à états finis déterministe des 2 commentaires groupés.



# 6- Grammaires régulières



## 6.1- **Définition**:

Une grammaire est régulière si toutes ses productions vérifient une des 2 formes:

$\underline{A} \longrightarrow a\,\underline{\mathbf{B}}$

ou $\underline{A} \longrightarrow a$

avec:

- ✓ **A** et **B** des non-terminaux
- ✓ **a** un terminal ou une chaîne vide ε.

- Ces grammaires régulières sont appelées des grammaires ***linéaires droites***.

# 6- Grammaires régulières



Par analogie, il est possible de définir des grammaires ***linéaires gauches***:

$$\underline{\mathbf{A}} \longrightarrow \underline{\mathbf{B}}\ a$$

ou $$\underline{A} \longrightarrow a$$

Remarque:

Les grammaires régulières sont une sous-classe des grammaires hors contextes.

Elles permettent de décrire les ***langages réguliers***.

# 6- Grammaires régulières



**6.2- <u>Correspondance entre une grammaire régulière et un automate</u>:**

Nous pouvons faire la correspondance entre un ***automate*** et une ***grammaire régulière*** de la manière suivante:

- ✓ Chaque état de l'automate correspond à un non terminal de la grammaire.
- ✓ Chaque transition correspond à une production de la grammaire.
- ✓ L'état initial de l'automate correspond à l'axiome de la grammaire.
- ✓ Un état d'acceptation final correspond à la production de la chaine vide ε.

# 6- Grammaires régulières



Exemple:

1- Soit l'expression régulière : **(a|b)*a(a|b)***

- Donner l'automate à états finis déterministe qui accepte les mots de cette expression régulière.
- Donner une grammaire régulière équivalente.

# 6- Grammaires régulières



<u>Exercices</u>:

1- Soit l'expression régulière : **`(a|b)*ab(a|b)*`**

- Donner l'automate à états finis déterministe qui accepte les mots de cette expression régulière.
- Donner une grammaire régulière équivalent.

# 6- Grammaires régulières



<u>Exercices</u>:

2- Soit l'ensemble des entiers multiples de '**5**'

- Donner une expression régulière qui valide ces mots.
- Donner l'automate à états finis déterministe correspondant.
- Donner une grammaire régulière équivalente.

# 6- Grammaires régulières



Exercices:

3- Soit l'ensemble des entiers multiples de '**10**'

- Donner une expression régulière qui valide ces mots.
- Donner l'automate à états finis déterministe correspondant.
- Donner une grammaire régulière équivalente.

# 6- Grammaires régulières



<u>Exercices</u>:

4- Soit l'ensemble des mots sur l'alphabet $\sum = \{\mathbf{a}, \mathbf{b}\}$ qui contiennent un nombre paire de '**a**' et un nombre impaire de '**b**'.

- Donner l'automate à états finis déterministe correspondant.

- Donner une grammaire régulière équivalente.

# 6- Grammaires régulières



Exercices:

4- Soit l'ensemble des mots sur l'alphabet $\sum = \{\mathbf{a}, \mathbf{b}\}$ qui contiennent un nombre impaire de '**b**'.

- Donner l'automate à états finis déterministe correspondant.
- Donner une expression régulière.
- Donner une grammaire régulière équivalente.

# 6- Grammaires régulières



<u>Exercices</u>:

5- Soit l'ensemble des mots sur l'alphabet $\sum = \{$**`a`**, **`b`**$\}$ qui contiennent un nombre impaire de '**`a`**' et un nombre impaire de '**`b`**'.

- Donner l'automate à états finis déterministe correspondant.
- Donner une grammaire régulière équivalente.

# Exercices



1- Donner un AFD qui accepte les mots spécifiés par l'expression régulière : `(01|10)+`

2- Donner un AFD qui accepte les mots spécifiés par l'expression régulière : `(0|1)+ 10 (1|0)+`

3- Donner un AFD qui accepte les mots sur l'alphabet {`0`,`1`} qui ne contiennent pas la chaîne : `"011"`

4- Donner un AFD qui accepte les mots sur l'alphabet {`0`,`1`} qui contiennent la chaîne : `"011"`

# Exercices



5- Donner un AFD qui accepte les mots sur l'alphabet $\sum = \{$**a**,**b**$\}$ qui contiennent au moins deux '**a**'

# Exercices



6- Donner un AFD qui accepte les mots sur l'alphabet $\sum = \{\mathbf{a},\mathbf{b}\}$ qui contiennent exactement trois '**b**'

# Exercices



7- Donner un AFD qui accepte les mots sur l'alphabet $\sum = \{\mathbf{a},\mathbf{b}\}$ qui contiennent au plus trois '**b**'

# Exercices



8- Donner un AFD qui accepte une forme simplifiée d'une adresse électronique.

# Exercices



9- Donner un AFD qui accepte les entiers strictement inférieurs à **57**, en évitant les '0' inutiles au début.

# Exercices



10- Donner un AFD qui accepte les entiers strictement inférieurs à **139**, en évitant les '0' inutiles au début.

# Exercices



11- Donner un AFD qui accepte les mots sur l'alphabet ∑ = {**a**,**b**} qui contiennent un nombre impair de '**a**' et un nombre impair de '**b**'.

# 7- Des expressions régulières aux automates



## 7.1- Conversion d'un AFN en un AFD:

Il s'agit de remplacer la relation de transition par une fonction partielle qui à un état et un caractère associe au plus un nouvel état.

L'idée est, si l'automate initial est construit sur un ensemble S d'états, de construire un nouvel automate avec comme états des ensembles d'états de S.

# 7- Des expressions régulières aux automates



## 7.1- Conversion d'un AFN en un AFD:

Définition: On définit l'ensemble des ε-successeurs d'un état p et on note ε-Succ(p), l'ensemble des états q tels que:

$$p \xrightarrow{\varepsilon} q$$

On note ε-Succ(P) pour un ensemble d'états, l'union des ε-successeurs des éléments $p \in P$.

Remarque: *De tels successeurs peuvent être obtenus par plusieurs transitions.*

On définit l'ensemble des successeurs d'un état p pour un caractère a et on note Succ(p,a), l'ensemble des états q tels que:

$$p \xrightarrow{a} q$$

# 7- Des expressions régulières aux automates



**7.1- Conversion d'un AFN en un AFD:**

Algorithme: On se donne un automate (S, $e_0$, F, Trans). L'automate correspondant aura pour états des parties de S, c-à-d des ensembles d'états.

On notera de manière générale *P*(E) l'ensemble des parties E et plus spécifiquement $P_S$ l'ensemble des parties de S.

Un automate déterministe reconnaissant le même langage est:

- ensemble d'états : $P_Q$
- état initial: ε-Succ($e_0$)
- état d'acceptation: $\{q \subset S \mid q \cap F \neq \emptyset\}$
- transition: $\{(q,a,q') \mid q,q' \in P_Q, a \in A, \forall y \in S. y \in q' \Leftrightarrow$

$$\exists\, x \in q. x \xrightarrow{a} y$$

## 7.2- Construction d'un AFN à partir d'une expression régulière:

**Algorithme de Mc Naughton-Yamada-Thomson**:

Données: *Une expression régulière* $\mathbf{r}$ *sur un alphabet* $\Sigma$.

Résultat: *Un AFN* $N$ *acceptant* $\mathbf{L(r)}$ *.*

Méthode:

- *Décomposer* $\mathbf{r}$ *en sous expressions constitutives.*

- *Les règles de construction d'un AFN contiennent des règles de base pour traiter les sous-expressions .*

# 7- Des expressions régulières aux automates

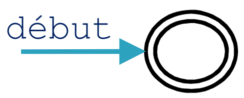


**Algorithme de Mc Naughton-Yamada-Thomson**:

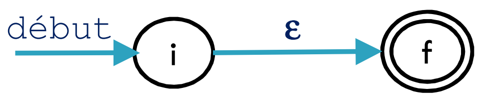
Soit **r** une expression régulière,

<u>Cas de base</u>:

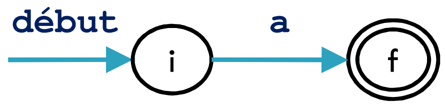
si **r = ε** , l'automate est :




ou




si **r = a** , l'automate est :



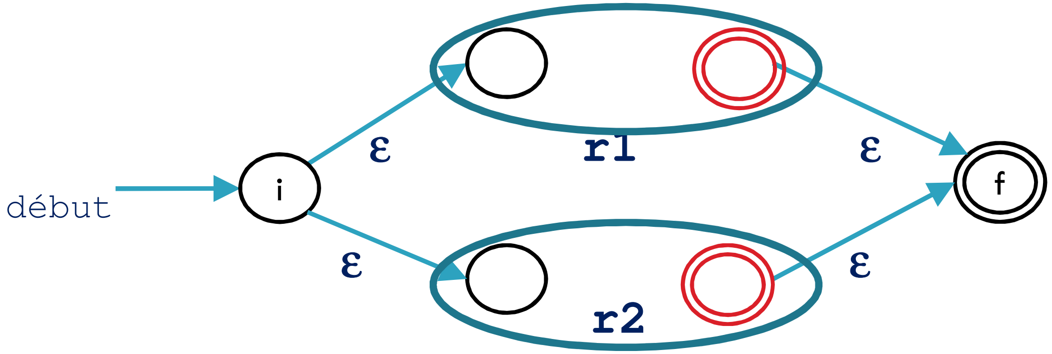

i est l'état initial et f est l'état final de l'AFN.

# 7- Des expressions régulières aux automates



Cas composés:

1- si $\mathbf{r} = \mathbf{r_1} \mid \mathbf{r_2}$ , l'automate associé à $\mathbf{r}$ est :



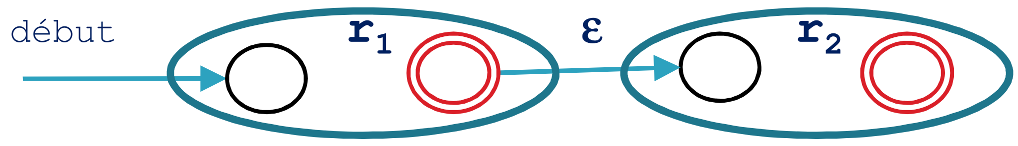

L'état initial associé à $\mathbf{r}$ comporte des ε-transitions vers les états initiaux des automates associés à r1 et r2.

Les anciens états initiaux deviennent des états ordinaires, de même pour les états finaux

# 7- Des expressions régulières aux automates



2- si $\mathbf{r} = \mathbf{r}_1\ \mathbf{r}_2$ , l'automate associé à $\mathbf{r}$ est :



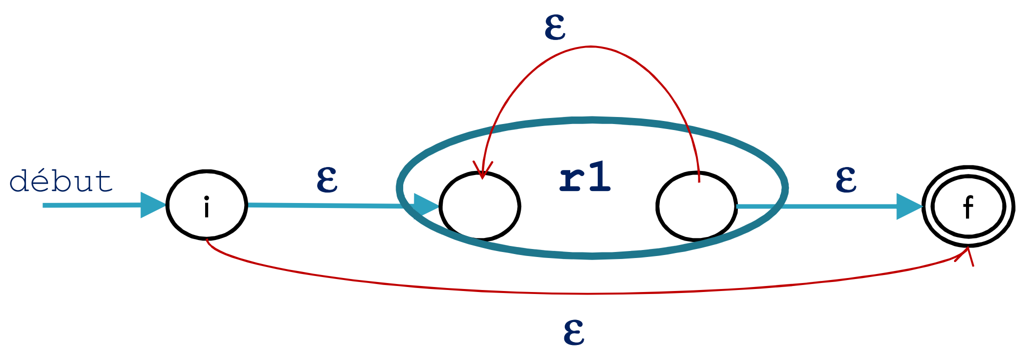

L'état initial associé à $\mathbf{r}_1$ devient un état initial de $\mathbf{r}$ et l'état final de $\mathbf{r}_2$ devient état final de $\mathbf{r}$.

.

# 7- Des expressions régulières aux automates



3- si $r = r_1{*} = \varepsilon \mid r_1{+}$ , l'automate associé à $r$ est :




La répétition non nulle (+) consiste à relier l'état final de l'automate de **r1** à son état initial.

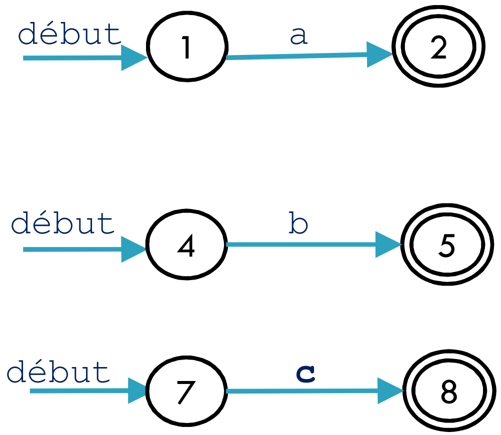
Pour ajouter ε au langage reconnu par l'automate, il suffit de créer un nouvel état initial et un état final et de les relier avec une transition ε .

# 7- Des expressions régulières aux automates



<u>Exemple</u>: Soit l'expression régulière **`a | b c*`**

- Pour '**`a`**' , '**`b`**' et '**`c`**', on a les automates:

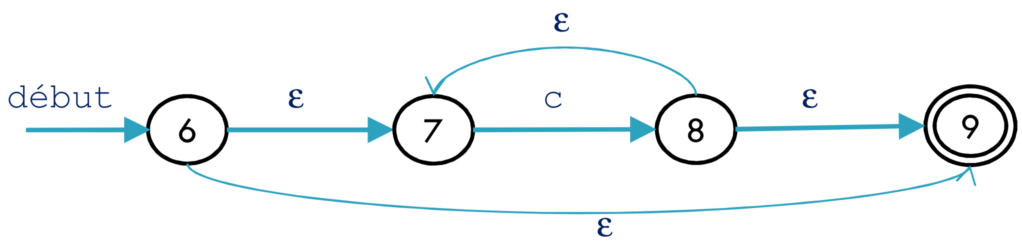


# 7- Des expressions régulières aux automates

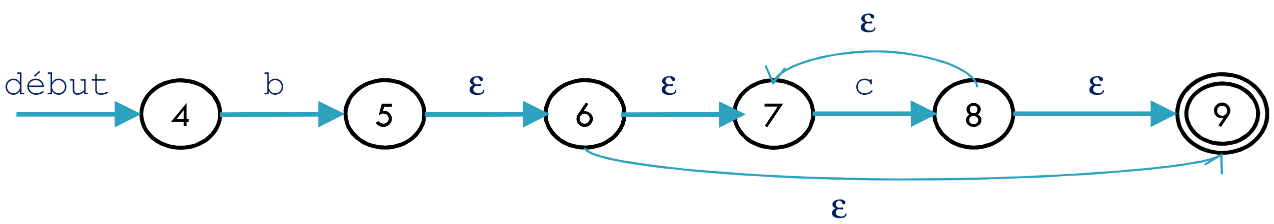


Exemple: Soit l'expression régulière **`a | b c*`**

- Pour **`c*`** , on a:




- Pour `b c*` , on a:

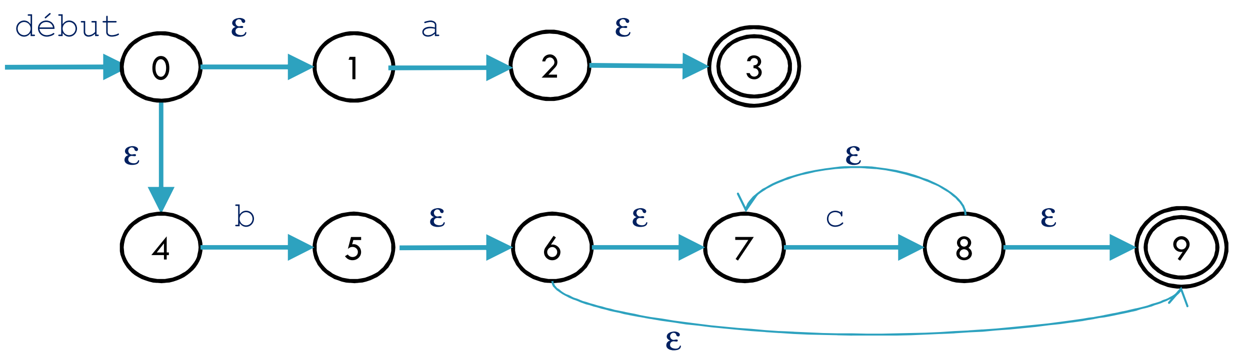


# 7- Des expressions régulières aux automates



Exemple: Soit l'expression régulière **a | b c***

- Pour **a|bc*** , on a:




L'expression régulière équivalente:

a|b|bcc* = a|b|bc+ = a|bc*

# 7- Des expressions régulières aux automates



**Elimination des ε-transitions:**

Elle se fait en 4 étapes:

1- Augmentation des transitions.

2- Propagation des états finaux.

3- Suppression des ε-transitions.

4- Elimination des états inaccessibles.

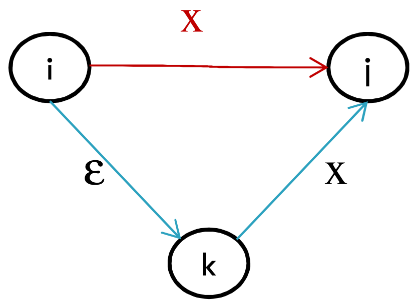
# 7- Des expressions régulières aux automates



**Elimination des ε-transitions:**

1- Augmentation des transitions:

On construit un nouvel automate où il existe une transition entre l'état **`i`** et l'état **`j`** étiqueté par **`x`**, s'il existe un état **`k`** tel qu'il existe une suite d' ε-transitions de **`i`** à **`k`** et qu'il existe une transition **`x`** de **`k`** à **`j`**.



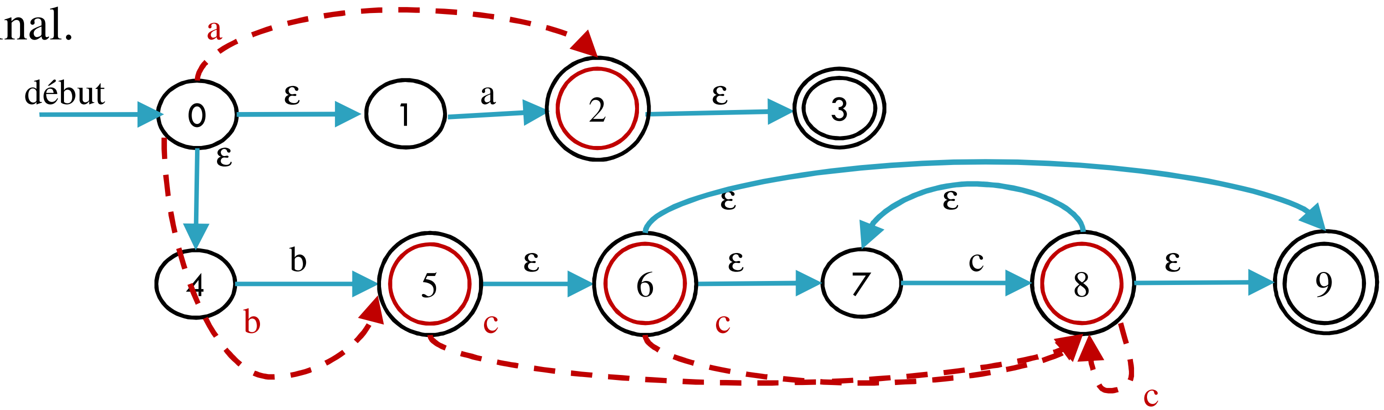
# 7- Des expressions régulières aux automates



**Elimination des ε-transitions:**

1- Augmentation des transitions:

2- Un état est final s'il existe est une suite d'ε-transitions qui mènent à un état final.



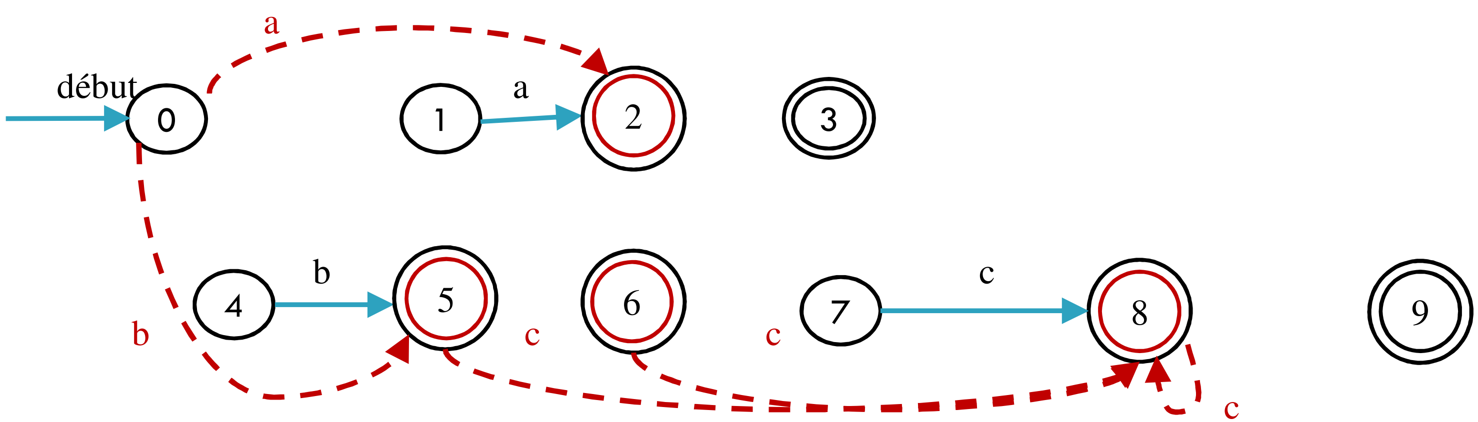
# 7- Des expressions régulières aux automates



**Elimination des ε-transitions:**

3- On supprime les ε-transitions:

4- On supprime les états inaccessibles à partir de l'état initial.



# 7- Des expressions régulières aux automates



**Elimination des ε-transitions:**

3- On supprime les ε-transitions:

4- On supprime les états inaccessibles à partir de l'état initial.