COMPILATION

ANALYSE LEXICALE (1ÈRE PARTIE)

EMSI - 4ÈME IIR

2017/2018

Prof. M. D. RAHMANI

Conception d'un analyseur lexical

1. Rôle d'un analyseur lexical,
2. Expressions et définitions régulières,
3. AFD et diagramme de transition,
4. Implantation manuelle avec les langages C,C++,
5. Implantation automatique avec le langage Flex,
6. Application à un mini langage.

L'analyse lexicale (1ère partie)

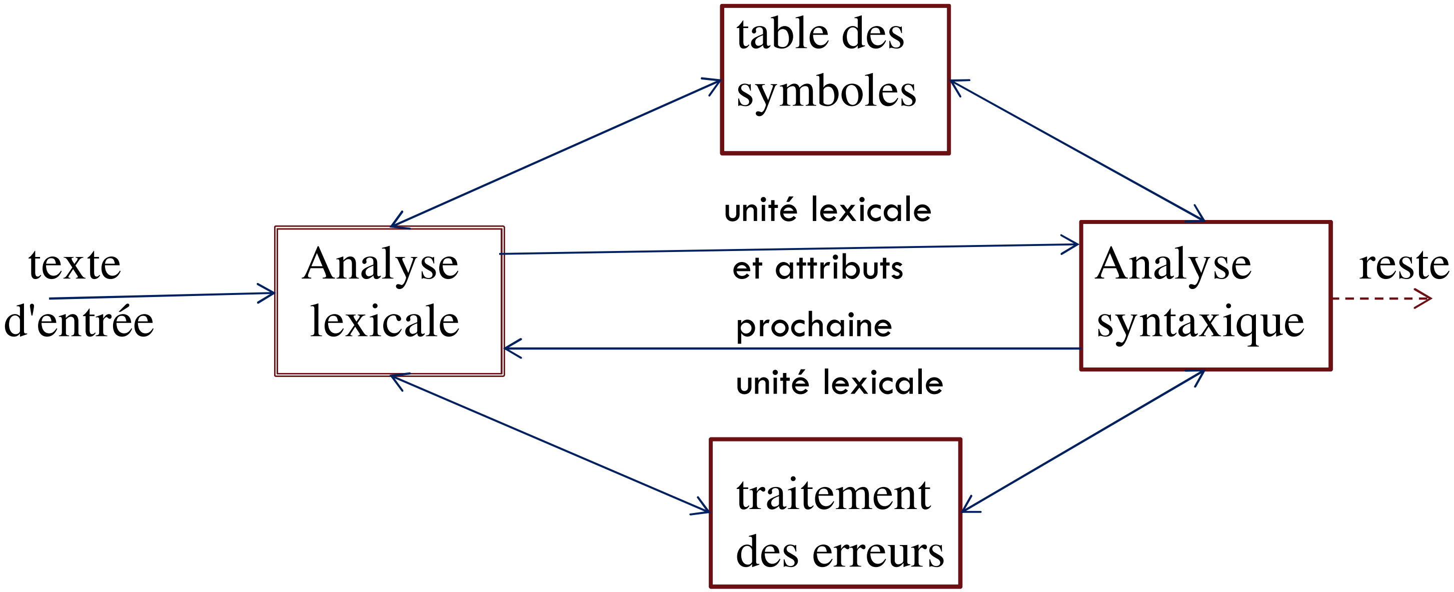
1- Le rôle d'un analyseur lexical (1)

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 (2)



Interaction entre analyseur lexical et analyseur syntaxique.

2- Terminologie (1)

- ✓ **Unité lexicale: classe des mots**

  est un symbole terminal de la grammaire du langage.

- ✓ **Modèle: expression** ou **définition régulière**

  est une règle qui décrit un ensemble de chaînes associées à la même unité lexicale.

- ✓ **Lexème: mot**

  est une suite de caractères du texte d'entrée qui concorde avec le modèle.

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

2- **Terminologie (2)**

<u>Remarques</u>:

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 groupes.

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 (1)

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 (2)**

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)

3.2- Opérations sur les langages:

Soit L et M deux langages:

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

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

- 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 (4)

<u>Exemples:</u>

Soit $L = \{A,B,\ldots,Z\}\ U\ \{a,b,\ldots,z\}$ *et* $C = \{0,1,\ldots 9\}$

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

- $\boldsymbol{L\ U\ C}$: ensemble des lettres et chiffres,
- $\boldsymbol{LC}$: ensemble des chaînes constituées d'une lettre suivie d'un chiffre,
- $\boldsymbol{L^4}$: ensemble des chaînes constituées de 4 lettres,
- $\boldsymbol{C^+}$: ensemble des entiers naturels,
- $\boldsymbol{L(LUC)^*}$: 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 (5)**

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 (6)

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 (7)**

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 (8)

Exemples:

`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}`

Définition:

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

Exemple: **`(a|b)= (b|a)`**

3- Spécification des unités lexicales (9)

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 (10)

Notations:

***** 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.

[a-z] = a|b|c...|z

3- Spécification des unités lexicales (11)

Conventions:

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 *****.

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

4- 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 (12)

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 (13)

3.4- Définitions régulières:

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

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

Chaque $\mathbf{d_i}$ est un nom distinct et chaque $\mathbf{r_i}$ est une expression régulière sur les symboles : $\mathbf{\Sigma\ U\ \{d_1,d_2,\ldots,d_{i-1}\}}$

3- Spécification des unités lexicales (14)

<u>Exemples</u>:

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 (15)

3- Définition régulière d'un réel:

L'alphabet Σ={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- Automates à états finis (AEF) (1)

4.1 Définition :

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

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

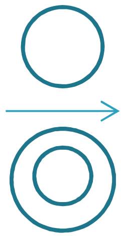
- Les automates à états finis non déterministes (**AFN**) n'ont aucune restriction sur les étiquettes de leurs arcs.

  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'accepte pas d'ε-transition

4- Automates à états finis (2)

4.2- <u>Structure</u> :

Il est constitué d'états et de transition 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.

<u>Remarque</u>:

En pratique, une transition correspond à la consommation d'<u>un caractère et un seul</u>.

4- Automates à états finis (3)

4.3- Automates à états finis non déterministes (AFN):

Un AFN se compose de:

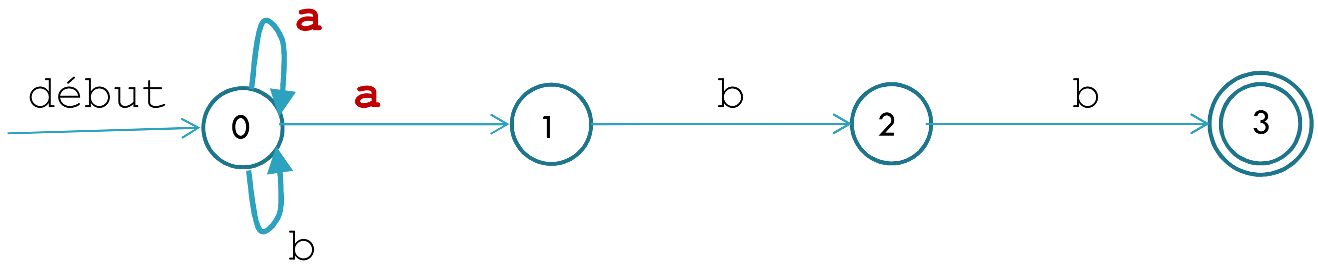
- Un ensemble fini **S** d'états.
- Un ensemble **Σ** de symboles d'entrée, l'alphabet du langage.

  *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*.

4- Automates à états finis (4)

4.4- 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****.*

4- Automates à états finis (5)

4.5- 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	-	-	-

4- Automates à états finis (6)

4.6- <u>Automates à états finis déterministes (AFD)</u>:

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.

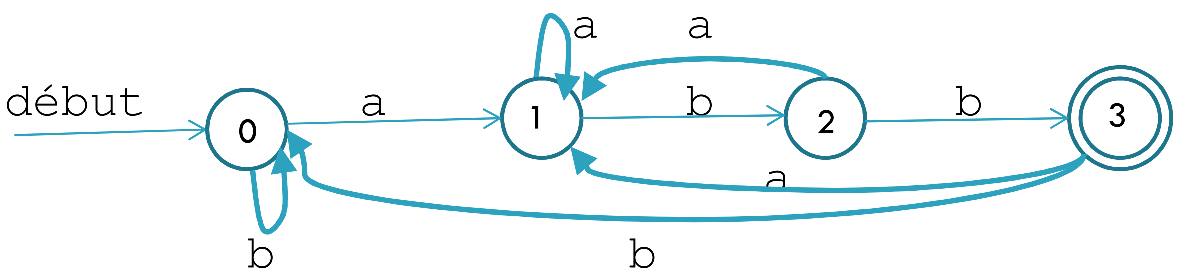
Un AFD est un algorithme ***concret*** de reconnaissance de chaînes.

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

4- Automates à états finis (7)

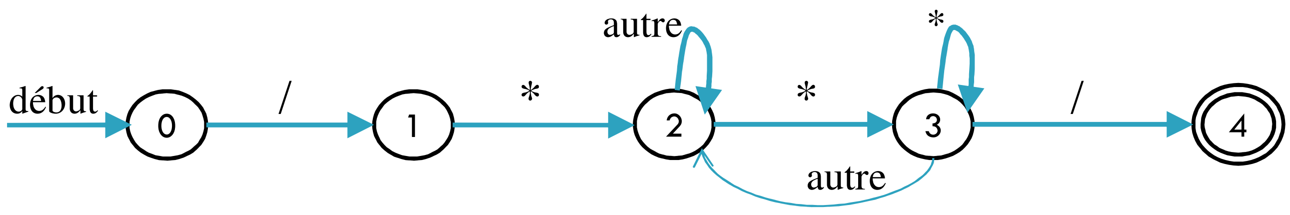
Exemple 1: L'AFD qui reconnait le langage défini par l'expression régulière :

`(a|b)* abb`

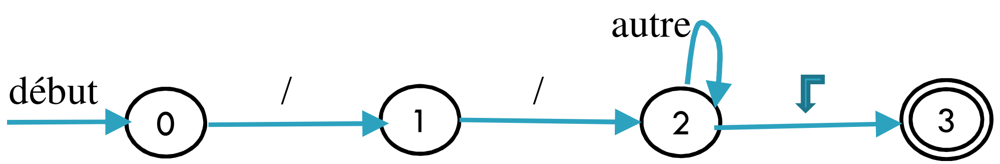


4- Automates à états finis (8)

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



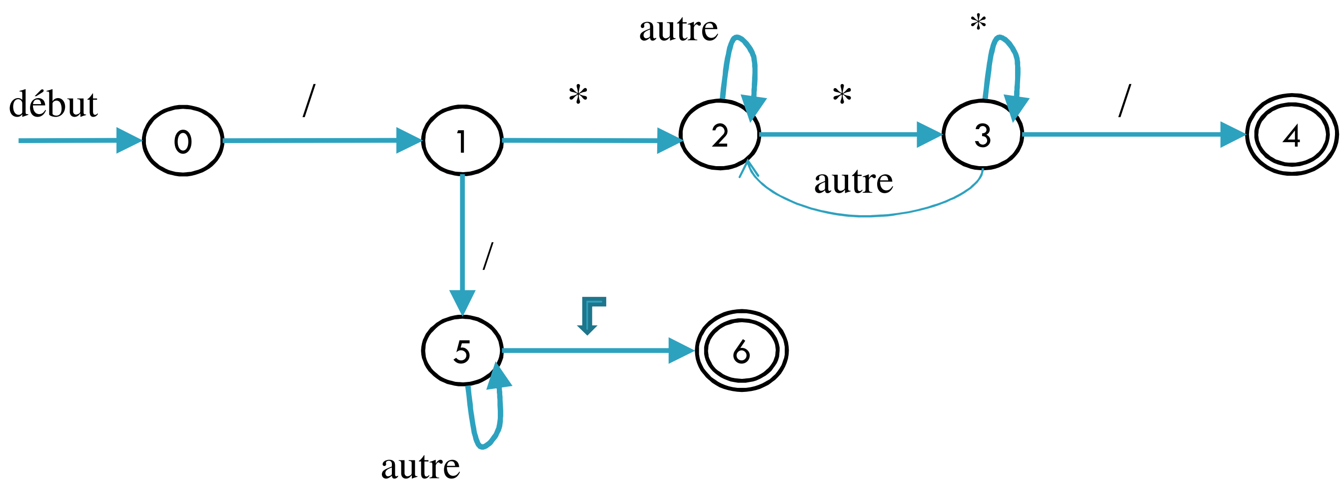
<u>Exemple 3</u>: L'automate à états finis déterministe d'un commentaire à la C++



↵ :désigne le retour à la ligne

4- Automates à états finis (9)

<u>Exemple 4</u>: L'automate à états finis déterministe des 2 commentaires groupés.



4- Automates à états finis (10)

4.7- <u>Algorithme d'application d'un AFD à une chaîne</u> :

<u>Donnée</u>:

Une chaîne d'entrée ***x*** terminée par un caractère de fin ***eof***.

Un AFD ***D*** dont l'état initial est e_0, les états finaux sont ***F*** et la fonction de transition est ***trans***.

<u>Résultat</u>:

"oui" si ***D*** accepte ***x***

"non" dans le cas contraire.

5- Grammaires régulières (1)

5.1- **Définition**:

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

$$A \longrightarrow a\,B$$

ou

$$A \longrightarrow a$$

avec **A** et **B** des non-terminaux et **a** un terminal ou ε.

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

5- Grammaires régulières (2)

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

$$A \longrightarrow B\ a$$

ou

$$A \longrightarrow a$$

<u>Remarque</u>:

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

Elles permettent de décrire les langages réguliers.

5- Grammaires régulières (3)

5.2- Correspondance entre une grammaire régulière et un automate:

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 final correspond à la production de la chaine vide ε.

6- Des expressions régulières aux automates (1)

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

Algorithme de Mc Naughton-Yamada-Thomson:

Données: *Une expression régulière* **r** *sur un alphabet* Σ.

Résultat: *Un AFN* N *acceptant* **L(r)** .

Méthode:

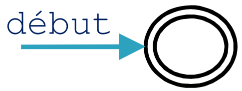
- *Décomposer* **r** *en sous expressions constitutives.*
- *Les règles de construction d'un AFN contiennent des règles de base pour traiter les sous-expressions .*

6- Des expressions régulières aux automates (2)

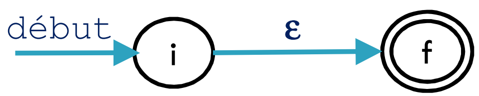
Soit **r** une expression régulière,

<u>Cas de base</u>:

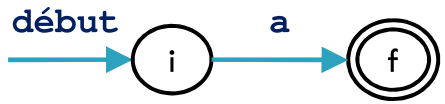
si **r = ε** , l'automate est :



ou



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

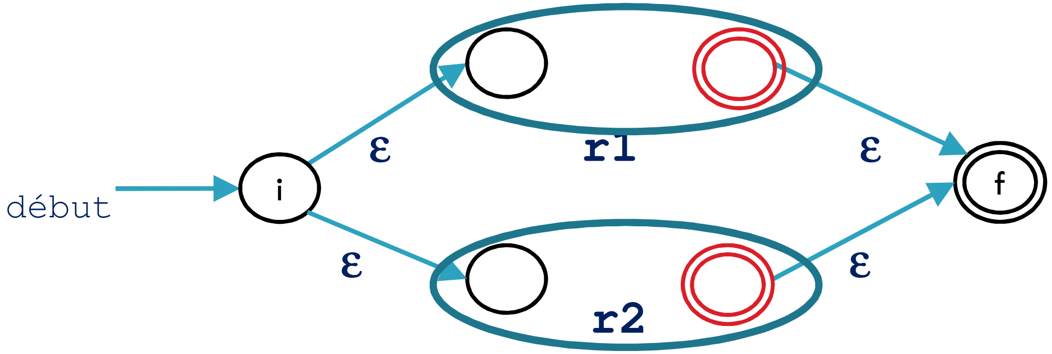


i est l'état initial et f est l'état final de l'AFN.

6- Des expressions régulières aux automates (3)

Cas composés:

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

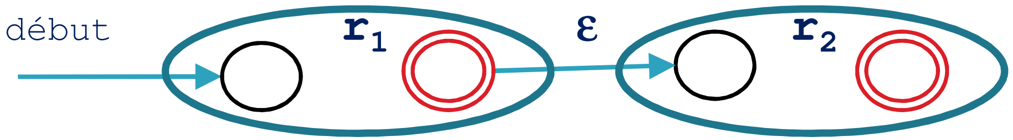


L'état initial associé à $\mathbf{r}$ comporte des ε-transitions vers les états initiaux des automates associés à r1 et r2.

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

6- Des expressions régulières aux automates (4)

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

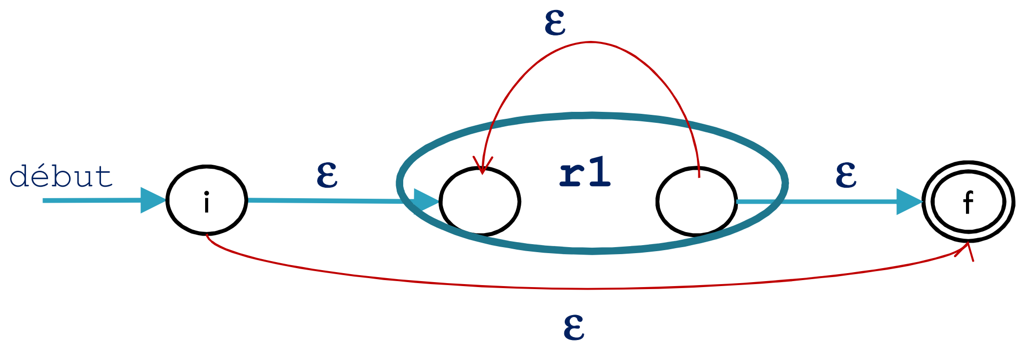


L'état initial associé à $\mathbf{r}_1$ devient un état initial de $\mathbf{r}$ et l'état final de $\mathbf{r}_2$ devient état final de $\mathbf{r}$.

.

6- Des expressions régulières aux automates (5)

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



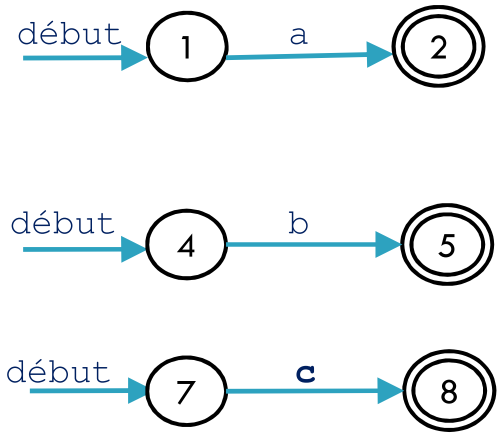
La répétition non nulle (+) consiste à relier l'état final de l'automate de **r1** à son état initial.

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** .

6- Des expressions régulières aux automates (6)

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

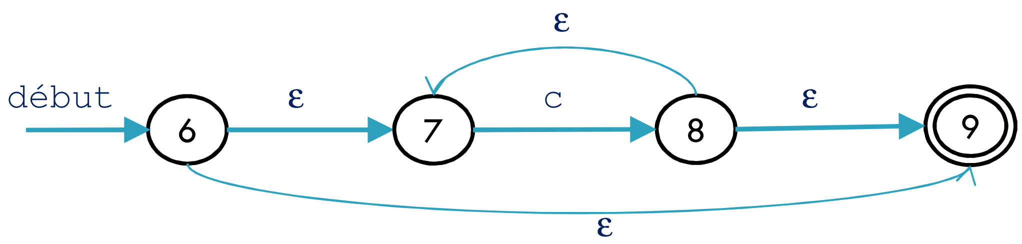
- Pour **`'a'`** , **`'b'`** et **`'c'`**, on a les automates:



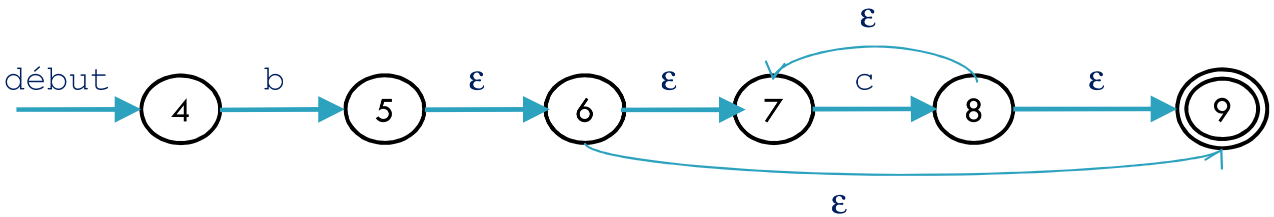
6- Des expressions régulières aux automates (8)

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

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



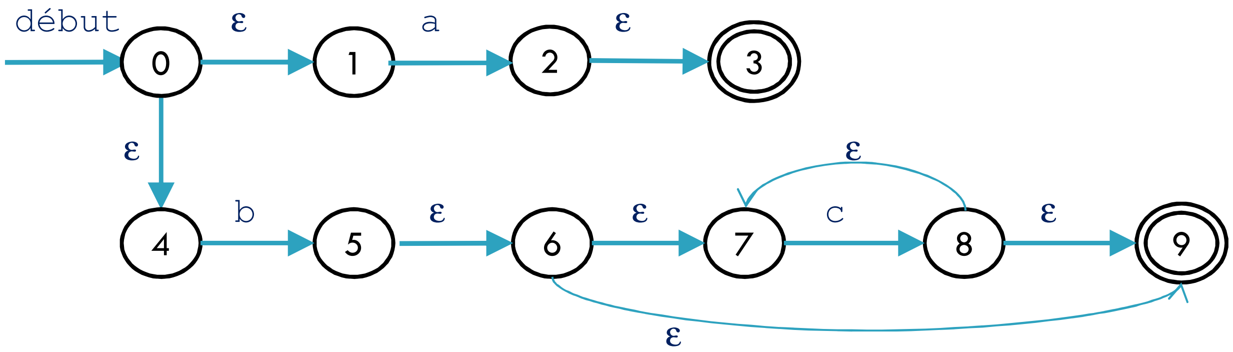
- Pour `b c*` , on a:



6- Des expressions régulières aux automates (9)

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*

6- Des expressions régulières aux automates (10)

Elimination des ε-transitions:

Elle se fait en 4 étapes:

1- Augmentation des transitions.

2- Propagation des états finaux.

3- Suppression des ε-transitions.

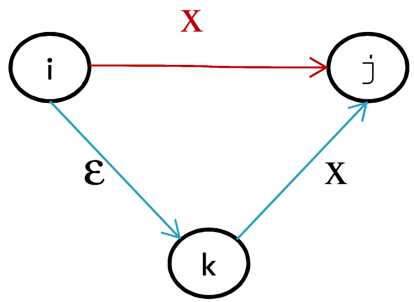
4- Elimination des états inaccessibles.

6- Des expressions régulières aux automates (11)

<u>Elimination des ε-transitions</u>:

1- <u>Augmentation des transitions</u>:

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`**.

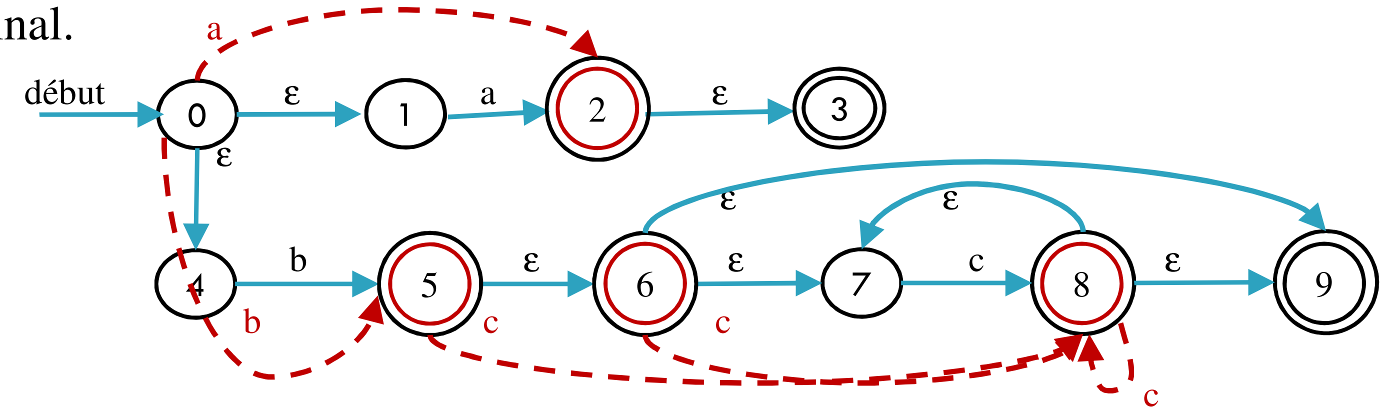


6- Des expressions régulières aux automates (12)

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.

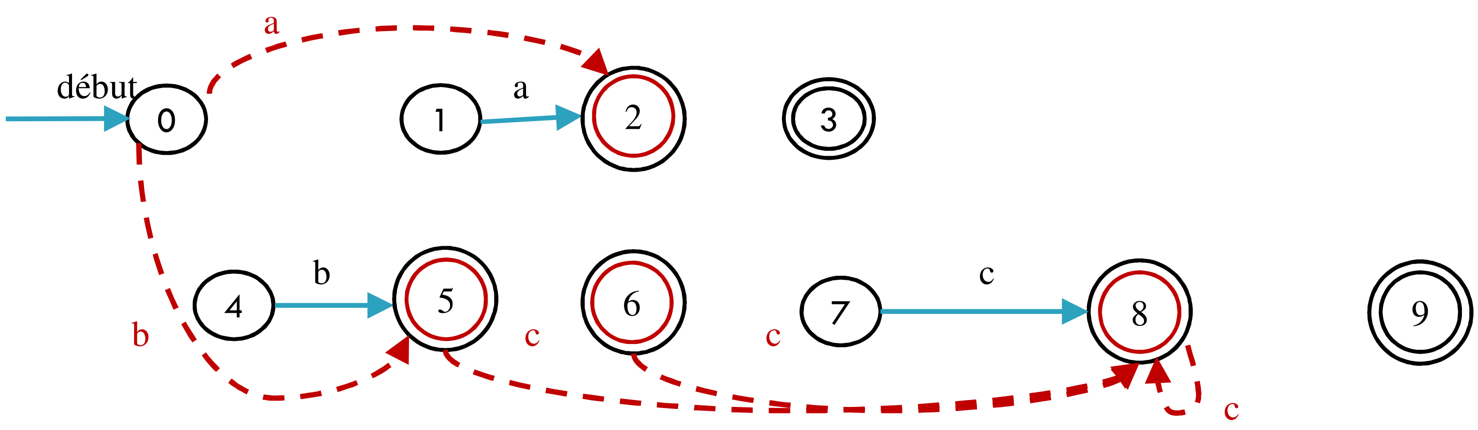


6- Des expressions régulières aux automates (13)

Elimination des ε-transitions:

3- On supprime les ε-transitions:

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

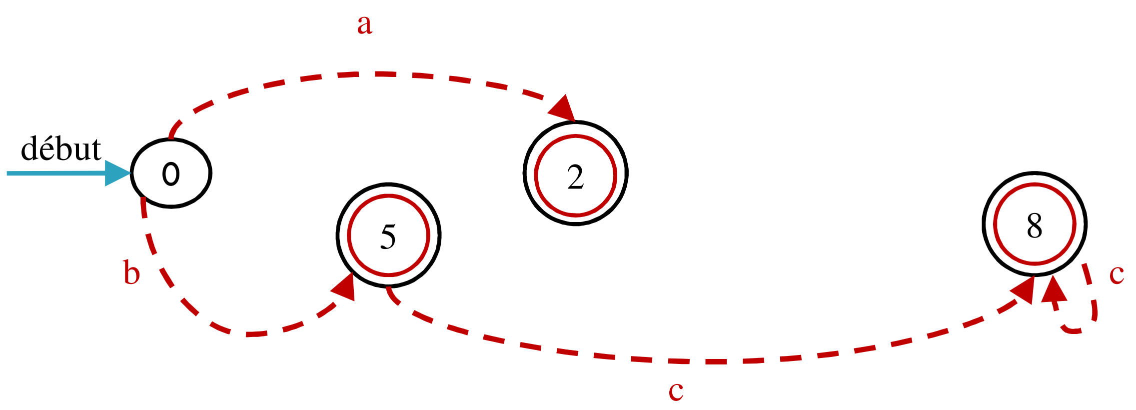


6- Des expressions régulières aux automates (14)

<u>Elimination des ε-transitions</u>:

3- On supprime les ε-transitions:

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



7- Reconnaissance des unités lexicales (1)

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.

7- Reconnaissance des unités lexicales (2)

7.1- 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.*

7- Reconnaissance des unités lexicales (3)

7.2- <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 transition entre états, définis par les notations suivantes:

un état

un arc ou une transition

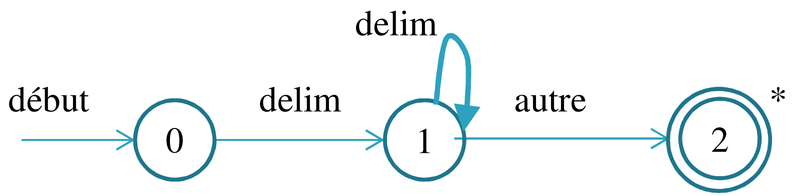
état d'acceptation, c-à-d que le lexème est reconnu.

état d'acceptation avec recul

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

7- Reconnaissance des unités lexicales (4)

1- Diagramme de transition des `blancs`:



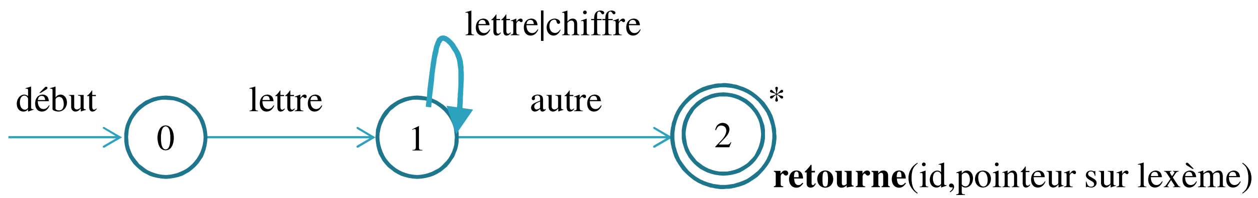
(supprimer)

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

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

7- Reconnaissance des unités lexicales (5)

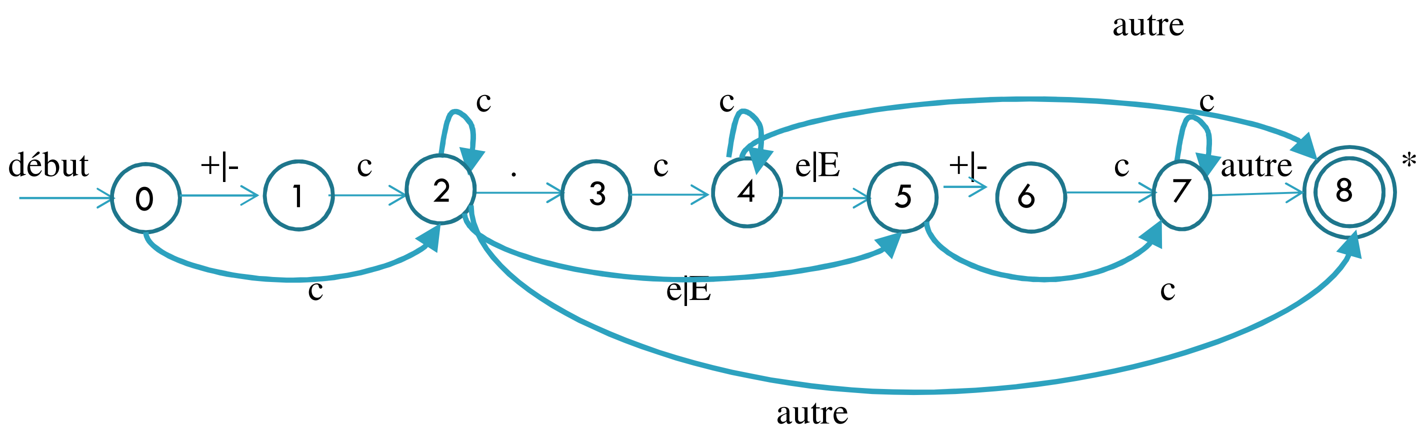
2- Diagramme de transition des `identificateurs`:



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

7- Reconnaissance des unités lexicales (6)

3- Diagramme de transition des `nombres réels`:

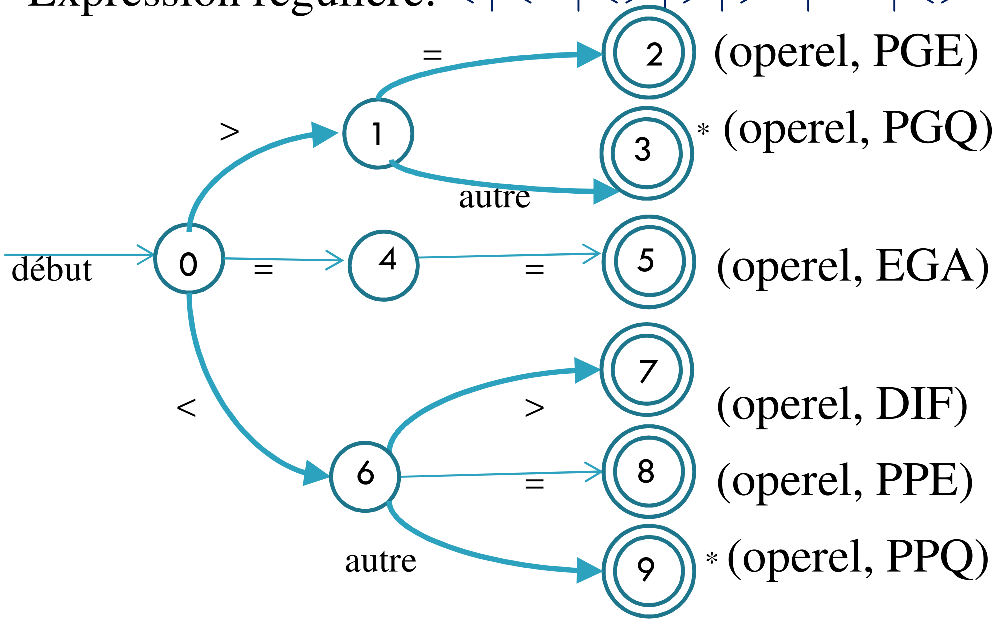


A l'état ***`14`*** *d'acceptation avec recul, nous retournons l'unité lexicale* ***`nb`*** *et un pointeur sur le lexème reconnu*

7- Reconnaissance des unités lexicales (7)

4- Diagramme de transition des `opérateurs de relation`

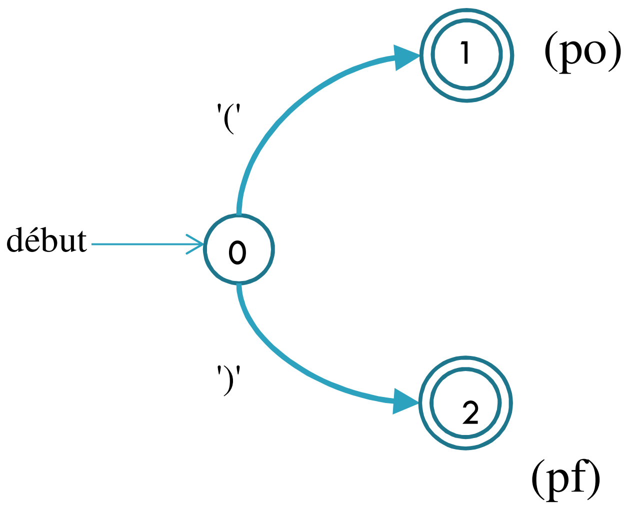
Expression régulière: < | <= | <> | > | >= | == | <>



7- Reconnaissance des unités lexicales (8)

5- Diagramme de transition des `parenthèses`

Expression régulière: `'('|')'`



7- Reconnaissance des unités lexicales (9)

7.2- Implantation des diagrammes de transition:

7- Reconnaissance des unités lexicales (10)

7.3- La table de symboles:

La table des symboles est une structure de données constituée des champs suivants:

- un pointeur (**`ptrlex`**) pointant sur l'adresse de la 1ère occurrence d'un lexème figurant dans le tampon (**`lexèmes`**);

- une chaîne de caractères (**`unilex`**) qui contient l'unité lexicale du lexème détecté;

- un attribut qui est un indice de la position du lexème dans la table des symboles.

7- Reconnaissance des unités lexicales (11)

Supposons que le texte d'entrée est: `"si gamma=10 alors aire >= 78 sinon g>1.3"`

La table des symboles aura la forme suivante:

ptrlex	unilex	indice
0	si	1
3	id	2
9	operel	3
11	nb	4
14	alors	5
20	id	6
25	operel	7
28	nb	8
31	sinon	9
37	id	10
39	operel	11
41	nb	12

7- Reconnaissance des unités lexicales (12)

La chaîne engendrée est:

```
"si$gamma$=$10$alors$aire$>=78$sinon$g$>$1.3"
```

Pour implanter la table des symboles, nous avons besoin des 2 fonctions suivantes:

- une fonction d'insertion;

- une fonction de recherche.

7- Reconnaissance des unités lexicales (13)

L'algorithme de la fonction d'insertion:

```
Fonction inserer
début
   indice ⟵ indice+1; création d'une nouvelle entrée de la TS
   TS[indice].ptrlex ⟵ l'adresse du début du lexème dans le
                        tampon lexèmes
   TS[indice].unilex ⟵ l'unité lexicale associée
   retourner(indice)
fin inserer
```

7- Reconnaissance des unités lexicales (14)

L'algorithme de la fonction de recherche:

```
Fonction chercher
début
   j ←———— 0;  pour parcourir la TS
   trouve=faux;  boolean pour arrêter la recherche
   tant que (j<taille de la TS et non trouvé) faire
        si TS[j].unilex=l'UL du lexème à chercher alors
                si TS[j].ptrlex pointe sur le lexème
                alors
                        trouve ←——— vrai;
                        retourne(j);
                sinon j ←——— j+1;
        sinon trouve retourne(-1)
Fin Fonction chercher
```

7- Reconnaissance des unités lexicales (15)

Un programme de la TS en langage C: