Analyse lexicale

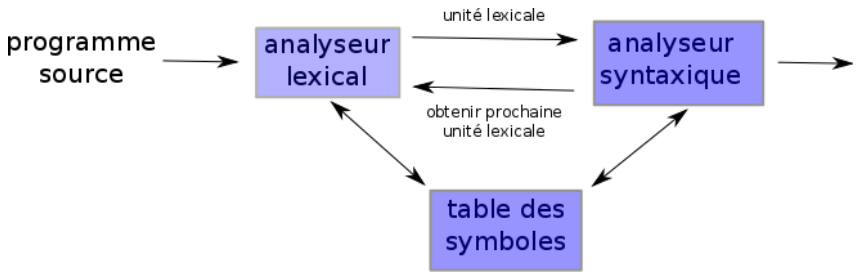
Objectifs :

- Transformation d'un ensemble de caractères en concepts (nombres, identificateurs, mots clés,...)
- On distingue les concepts suivants :

Unité lexicale : correspond à une entité (un concept) renvoyé par l'analyseur lexical. Par exemple $<, >, <=, >=$ sont tous des opérateurs relationels.

Un lexème est une instance d'unité lexicale. Par exemple le lexème 6.28 une instance de l'unité lexicale nombre.

Un modèle associe des lexèmes à leur unité lexicale.



Alphabet :

- Un alphabet Σ est un ensemble fini de symboles. L'alphabet binaire $\{0, 1\}$ et les codes ASCII sont des exemples d'alphabets, $\{A, G, C, T\}$ est l'alphabet des bases ADN.
- Une chaîne, est une séquence finie de symboles. Le terme mot est également utilisé.
- Un préfixe de s est obtenu en supprimant un nombre quelconque de caractères en fin de s.
- Un suffixe de s est obtenu en supprimant un nombre quelconque de caractères en début de s.
- Une sous-chaîne de s est obtenue en supprimant un préfixe et/ou un suffixe de s
- Un préfixe, suffixe, sous chaîne *propre de s est un suffixe, préfixe ou une sous chaîne de s non égal à s.*
- *Sous suite de s : toute chaîne obtenue en supprimant des caractères de s.*

Langages :

- Un langage est un ensemble a priori quelconques de chaînes construites sur un alphabet. L'ensemble des codes source C, des nombres binaires ou des codes ADN sont des langages.
- $\emptyset$ désigne le langage vide. On désigne également par $\{\epsilon\}$ le langage composé uniquement de la chaîne vide.
- Si x et y sont deux chaînes, la concaténation de x et y, xy représente la chaîne formée par la séquence des symboles de x suivie de celle de y. Ex : x=anti, y=constitutionnellement, xy=anticonstitutionnellement.
- Un alphabet Σ munit de la concaténation et de son élément neutre ϵ a une structure de monoïde (intuitivement un groupe sans l'inverse).
- Exponentiation : $s^0 = \epsilon$, $s^1 = s$, $s^n = s^{n-1}s$

Opérations sur les langages

Définitions :

- Union $(L \cup M)$:

$$L \cup M = \{s \mid s \in L \text{ ou } s \in M\}$$

- Concaténation (LM) :

$$LM = \{st \mid s \in L \text{ et } s \in M\}$$

Remarque : LL est dénoté L^2, $L^0 = \{\epsilon\}, L^1 = L$.

- Fermeture de Kleene (L^*) :

$$L^* = \bigcup_{i=0}^{+\infty} L^i$$

- Fermeture positive (L^+) :

$$L^+ = \bigcup_{i=1}^{+\infty} L^i$$

Opérations sur les langages

Exemples :

- Si $B = \{0, 1\}$,
  B^+ est l'ensemble des nombres binaires d'au moins un chiffre.
- Si $C = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$,
  C^4 est l'ensemle des nombres de 4 chiffres.
- Si $K = \{a, \ldots, z, A, \ldots, Z\}$,
  $K(K \cup C)^*$ est l'ensemble des mots commençant par une lettre puis composés d'un nombre quelconque de lettres et chiffres.

Expressions régulières

- Une expression régulière r est construite récursivement à partir d'union (notée $|$), de concaténation, et de fermeture. Elle définie un langage $L(r)$.

1. ϵ est une expression régulière qui dénote $L(\epsilon) = \{\epsilon\}$,
2. $a \in \Sigma$ est une expression régulière qui dénote l'ensemble $L(a) = \{a\}$,

2.1 (r) est une expression régulière dénotant $L(r)$,

2.2 $(r)|(s)$ est une expression régulière dénotant $L(r) \cup L(s)$,

2.3 $(r)(s)$ est une expression régulière dénotant $L(r)L(s)$,

2.4 $(r)^*$ est une expression régulière dénotant $L(r)^*$,

- Toute expression construite a partir de la construction ci-dessus est régulière.

Exemple

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

- $L(a|b) = \{a, b\}$,
- $L((a|b)(a|b)) = \{aa, ab, ba, bb\}$,
- $L(a^*) = \{\epsilon, a, a^2, \ldots\}$,
- $\{a, a^2, b, b^2, ab, ab^2, a^2b^2, \ldots\} \subset L((a|b)^*)$,
- On évite des paranthèse inutiles en utilisant les conventions suivantes :
- $*$ a la plus haute priorité,
- la concaténation à la deuxième plus haute priorité et est associative à gauche,
- $|$ a la plus faible priorité et est associatif à gauche.

$$a^*(a|b|c)^*de = ((a)^*\,(((a)|(b))|(c))^*)((d)(e))$$

Propriétés algébriques

Axiome	Descriptif
$r\|s = s\|r$	$\|$ est commutatif
$r\|(s\|t) = (r\|s)\|t$	$\|$ est associatif
$(rs)t = r(st)$	la concaténation est associative
$r(s\|t) = rs\|rt$ $(s\|t)r = sr\|tr$	la concaténation est distributive vis à vis de $\|$
$\epsilon r = r\epsilon = r$	ϵ est un élémement neutre pour la concaténation
$r^* = (r\|\epsilon)^*$	
$r^{**} = r^*$	$*$ est idempotent

Définitions régulières

Il peut être commode de donner des noms à des expressions régulières et de s'en reservir dans des expressions plus complexes. On utilise pour cela des définitions régulières.

- Soit Σ un alphabet, et un ensemble de couples

$$\begin{array}{lcl} d_1 & \rightarrow & r_1 \\ d_2 & \rightarrow & r_2 \\ & \vdots & \\ d_n & \rightarrow & r_n \end{array}$$

où r_i est une expression régulière et d_i son nom associé.

- Cette définition est appellée régulière si chaque r_i représente une expression régulière construite sur $\Sigma \cup \{d_1, \ldots, d_{i-1}\}$.

Exemple de définition régulière

- Identificateurs de variables

$$\begin{array}{rcl} \textbf{lettre} & \rightarrow & a|b|c\ldots|z|A|B\ldots|Z \\ \textbf{chiffre} & \rightarrow & 0|1|2|3|4|5|6|7|8|9 \\ \textbf{id} & \rightarrow & \textbf{lettre}(\textbf{lettre}|\textbf{chiffre})^* \end{array}$$

- Nombres :

$$\begin{array}{rcl} \textbf{pm} & \rightarrow & (+|-) \\ \textbf{chiffre} & \rightarrow & 0|1|2|3|4|5|6|7|8|9 \\ \textbf{frac} & \rightarrow & .\textbf{chiffre}^+ \\ \textbf{exp} & \rightarrow & E\ (\textbf{pm}|\epsilon)\ \textbf{chiffre}^+ \\ \textbf{nb} & \rightarrow & (\textbf{pm}|\epsilon)\left(\textbf{chiffre}^+(\textbf{frac}|\epsilon)|\textbf{frac}\right)(\textbf{exp}|\epsilon) \end{array}$$

NB : 14. ou -E5 ne sont pas reconnus par cette définition.

Notations abrégées et limites des expressions régulières

Notations abrégées :

- L'expression $(r|\epsilon)$ se note également $r?$. Ainsi **exp** $\rightarrow E$ $(\textbf{pm}|\epsilon)$ $\textbf{chiffre}^+$, se note également **exp** $\rightarrow E$ **pm**? $\textbf{chiffre}^+$,
- $(a|b|c)$ se note également $[abc]$. De même $(a|b|\ldots,|z)$ se note $[a-z]$ et $[a-zA-Z0-9]$ représente $(a|\ldots,|z|A|\ldots|Z|0\ldots|9)$. Ainsi l'ensemble des identificateurs se note : $[a-zA-Z][a-zA-Z0-9]^*$.

Limites des expressions régulières :
Ayez bien conscience que les expressions régulières ne permettent de coder qu'un nombre limité de languages. En particulier, les expressions régulières ne permettent pas de compter. Ainsi le langage :

$$L = \{wcw \mid w \in L(\{a,b\}^*)\}$$

n'est pas régulier.

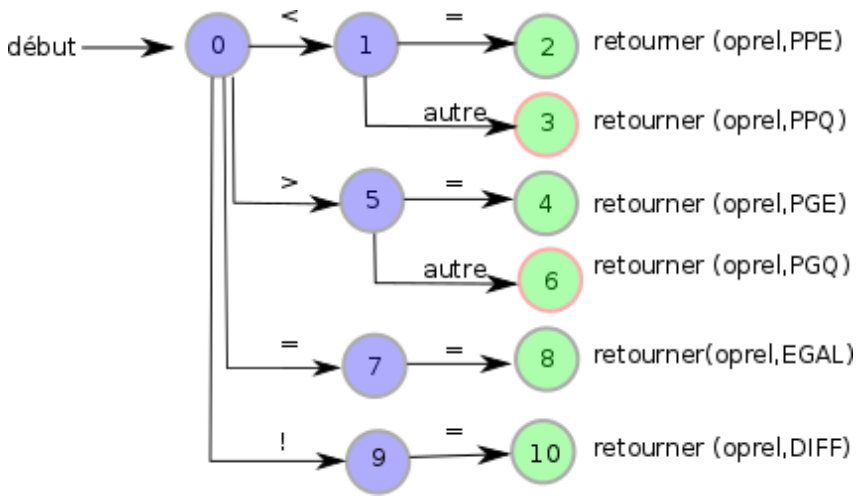
Unités lexicales et diagrammes de transition

Un diagramme de transition est une représentation graphique du processus de reconnaissance d'une unité lexicale. Il représente les actions réalisées par l'analyseur lexical sur le tampon d'entré lorsqu'il est appelé par l'analyseur syntaxique.

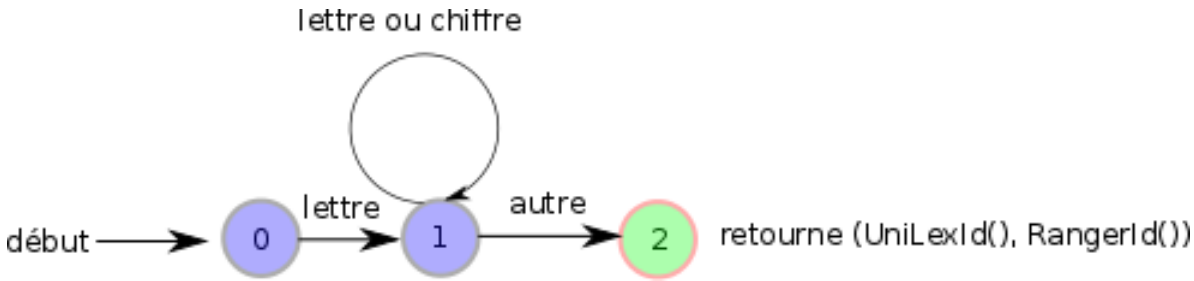
- Un diagramme de transition est constitué d'états :
- Internes
- Finaux :
- Finaux avec recul de tampon Ces états d'acceptation codent les cas où il est nécessaire de reculer le tampon d'entré.
- Les arcs codent les transitions entre états. Ils ont des étiquettes indiquant sur qu'elle entrée s'effectue chaque changement d'état. L'étiquette autre, code tout caractère autre que ceux indiqués par les changements d'états sur l'état courrant. On suppose les diagrammes déterministes (i.e. une même étiquette ne peut envoyer sur deux états différents)

Diagramme de transition : Exemple

- **oprel** $\rightarrow (< \mid > \mid <= \mid >= \mid == \mid !=)$



- **id** $\rightarrow$ **lettre**(**lettre**|**chiffre**)*.



Automates fini non déterministes (AFN)

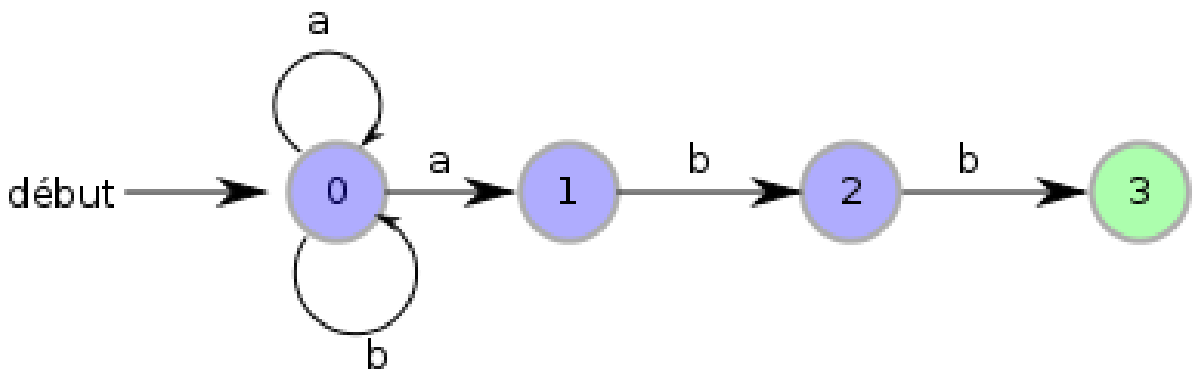
Un AFN est un modèle mathématique défini par :

- Un ensemble d'états E,
- Un ensemble de symboles d'entrées Σ
- Une fonction de transition Transiter, qui fait correspondre des couples (état,symbole) à des ensembles d'états
- Un état e_0 correspondant à l'état de départ
- Un ensemble d'états F représentant les états d'acceptation (ou états finaux).

Le langage défini par un AFN est l'ensemble des chaînes menant à un état d'acceptation.

AFN : Un premier exemple

- Soit l'unité lexicale $(a|b)^*abb$. Celle ci est reconnue par l'AFN suivant :

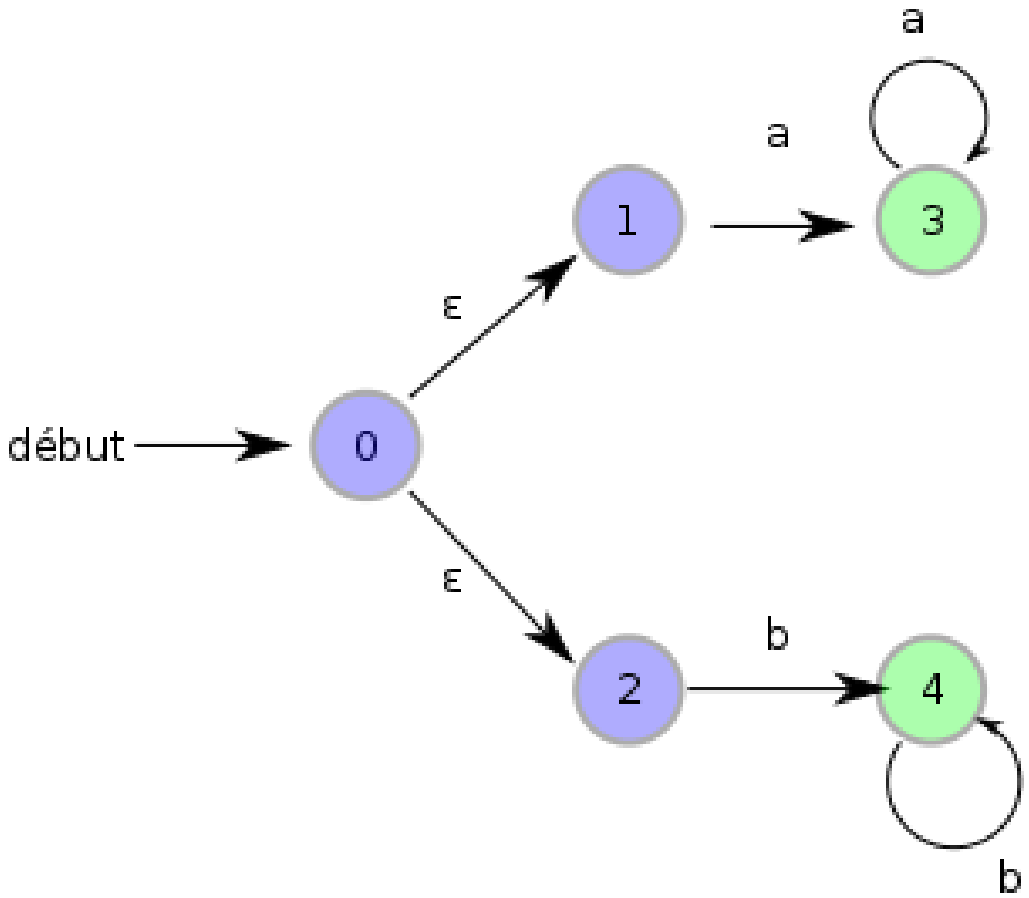


- associé à la table de transition :

Transiter		
Etat	Symbole d'entrée	
	a	b
0	$\{0, 1\}$	0
1	–	2
2	–	3

AFN : Un second exemple

- Soit l'unité lexicale $aa^*|bb^*$

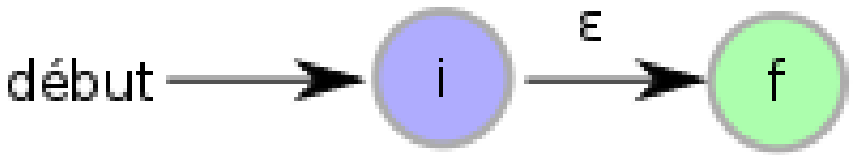


- Notez les transitions ϵ sur l'état de départ et les boucle sur les états d'acceptation.

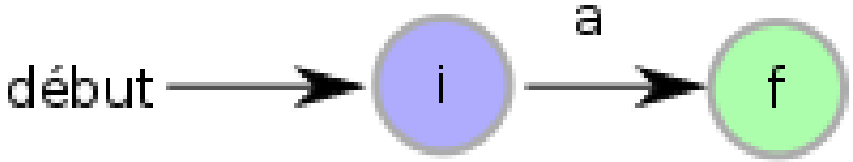
Construction d'un AFN à partir d'une expression régulière

Construction récursive :

1. Pour ϵ construire l'AFN :



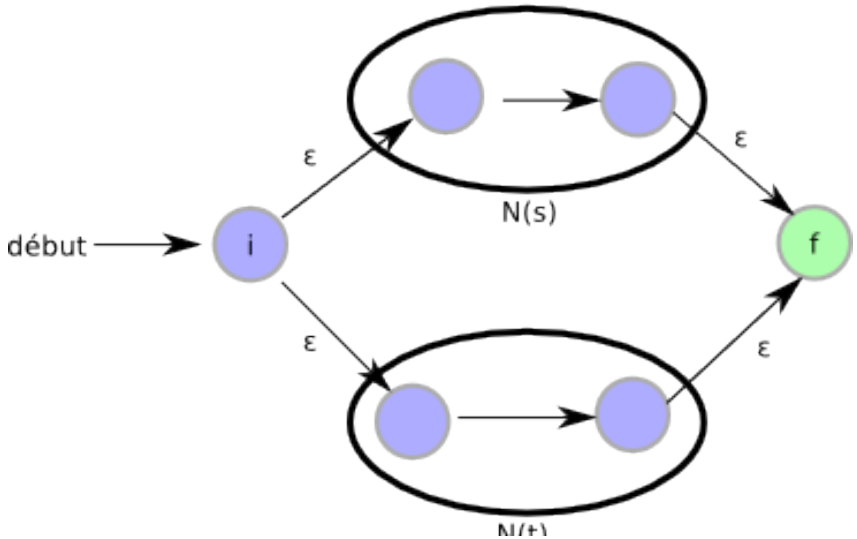
2. Pour $a \in \Sigma$ construire l'AFN :



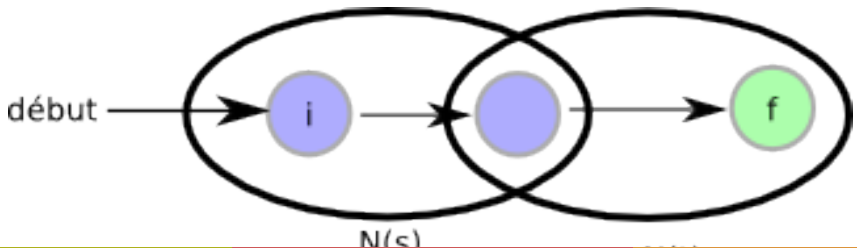
Construction d'un AFN à partir d'une expression régulière

Construction récursive :

3. Soient $N(s)$ et $N(t)$ les AFN des expressions s et t. L'AFN $N(s|t)$ est défini par :



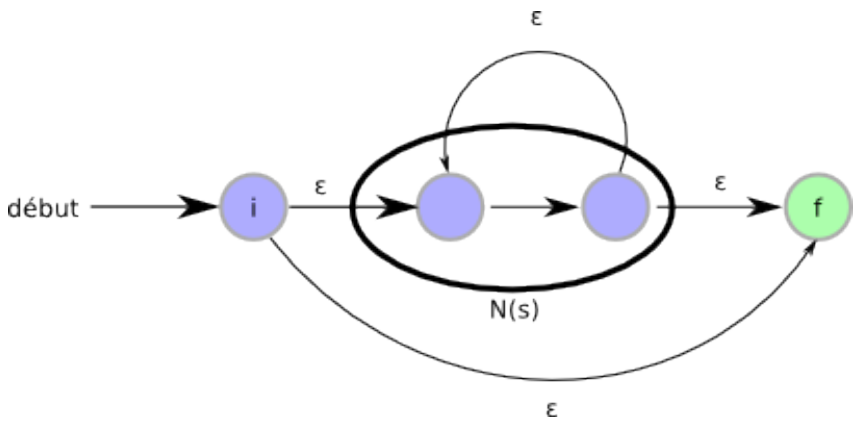
4. Soient $N(s)$ et $N(t)$ les AFN des expressions s et t. L'AFN $N(st)$ est défini par :



Construction d'un AFN à partir d'une expression régulière

Construction récursive :

4. Soit $N(s)$ l'AFN de l'expression s. L'AFN $N(s^*)$ est défini par :



5. L'AFN associé à (s) est $N(s)$ lui même.

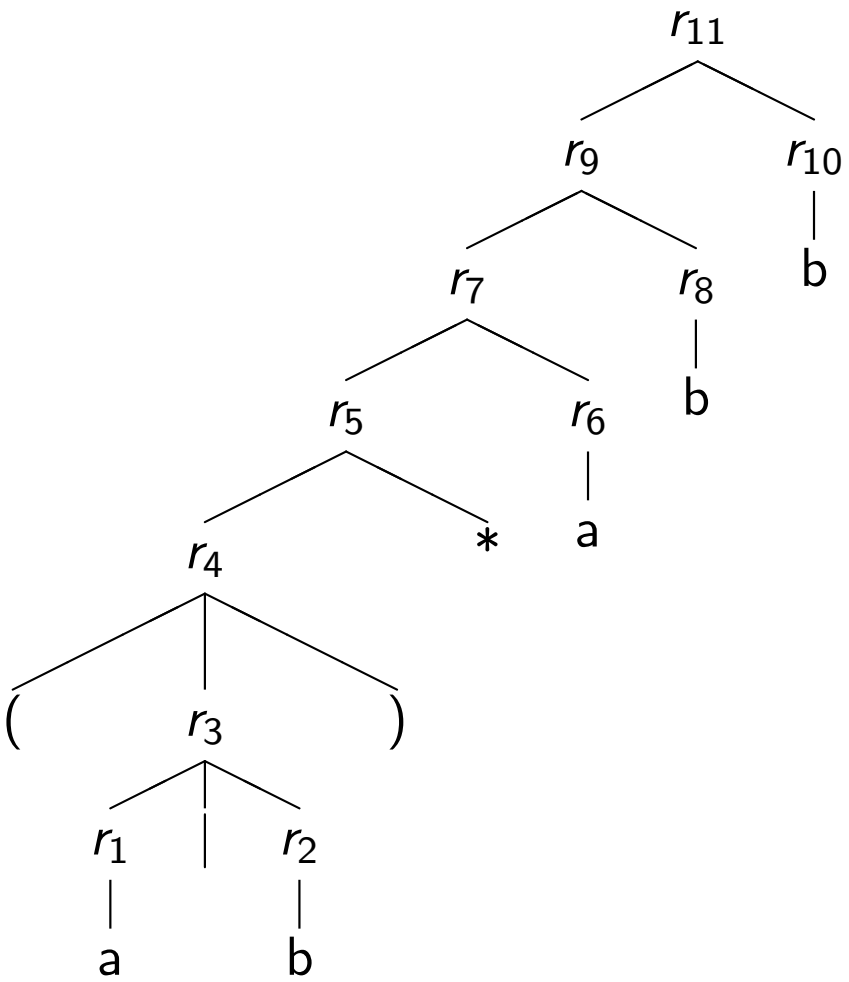
Propriétés de l'AFN construit

Les propriétés suivante se montrent facilement en suivant le processus de construction.

- $N(r)$ a au plus deux fois plus d'états qu'il n'y a de symboles dans r.
- $N(r)$ a exactement un état de départ et un état d'acceptation. L'état d'acceptation n'a pas de transition sortante.
- Chaque état de $N(r)$ a soit une transition sortante sur un symbole de Σ, soit au plus deux transistions sortantes sur ϵ.

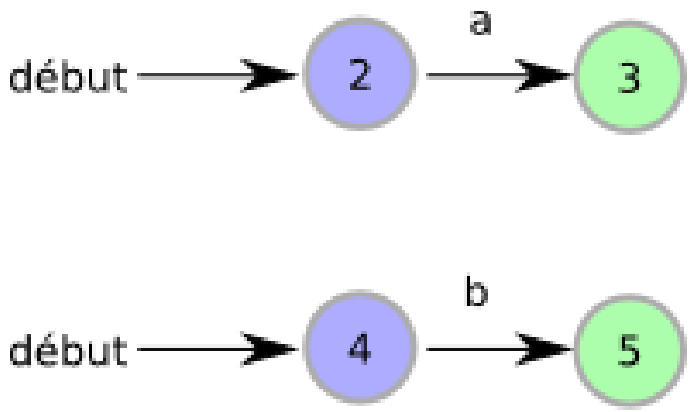
Exemple

- Évaluation de $(a|b)^*abb$. Son arbre syntaxique associé est (cf évaluation d'expressions cours 1A algo) :



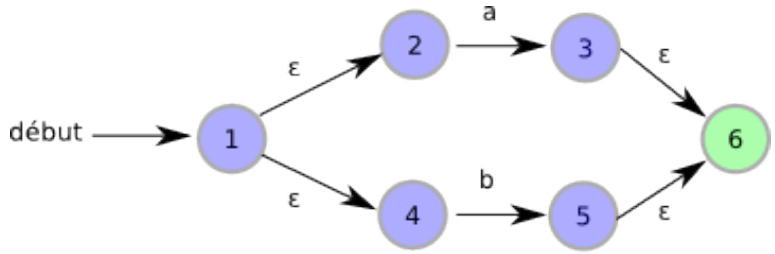
Exemple de construction d'AFN

- Construisons les AFN, $N(r_1)$ et $N(r_2)$:

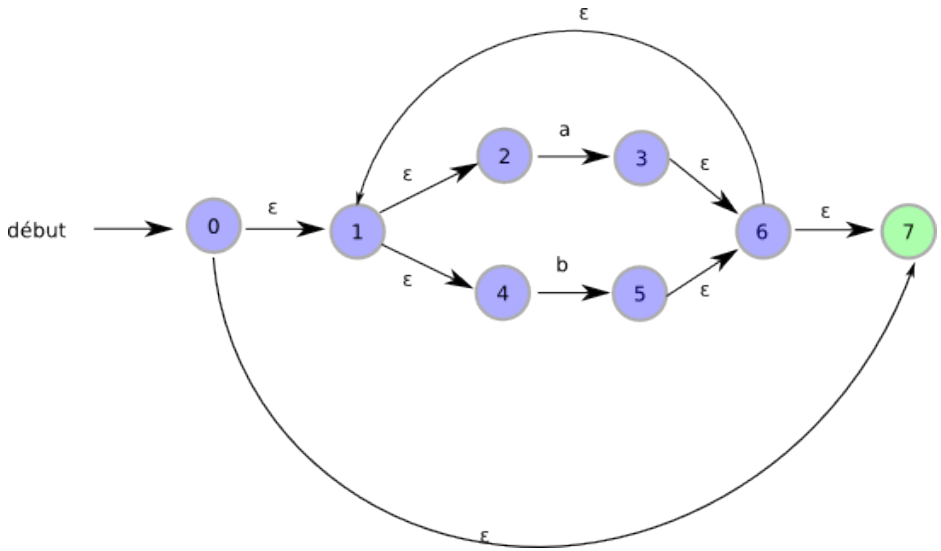


Exemple de construction d'AFN

- Construisons l'AFN de $r_3 = r_1|r_2$.

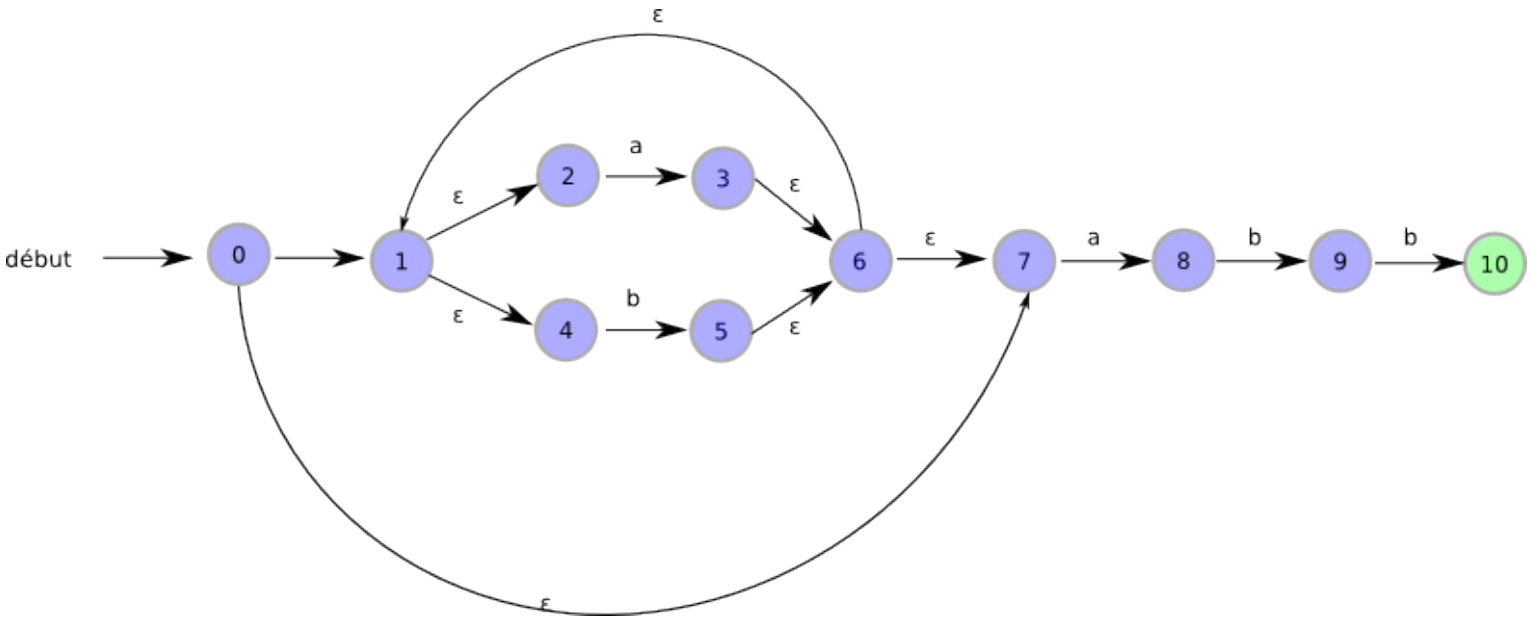


- $N(r_4) = N(r_3)$, construisons l'AFN de $r_5 = (r_1|r_2)^*$



Exemple de construction d'AFN

- Construisons directement r_{11} par 3 applications successives de la règle 4. Pour obtenir l'AFN final :



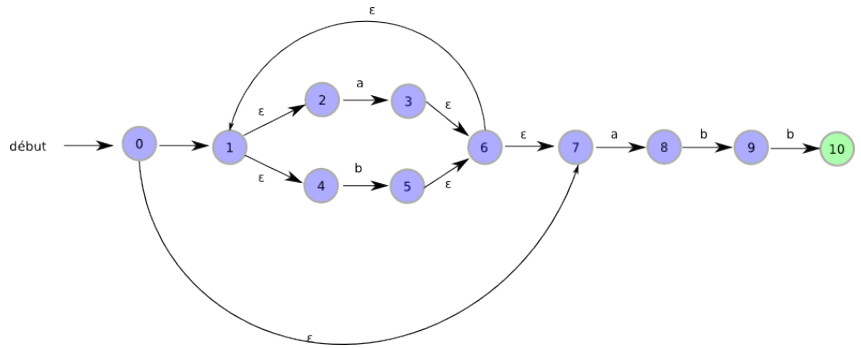
Reconnaissance d'une expression régulière par un AFN

L'algorithme calcule itérativement un ensemble d'états E en utilisant les fonctions :

- Transiter(E,a) : retourne l'ensemble des états accessibles via les états E par une transition 'a'.
- ϵ-fermeture(E) : renvoie l'ensemble des états accessibles depuis E par une ϵ transition.

fonction recAFN **(**e_0,F**) :** Booléen
    Déclaration E : Ensemble d'états,c : caractère
début
    E $\leftarrow$ ϵ-fermeture(e_0)
    c $\leftarrow$ carSuivant()
    tant que $c \neq$ fdf **faire**
        E $\leftarrow$ ϵ-fermeture(Transiter(E,c))
        c $\leftarrow$ carSuivant()
    fintantque
    retourner $E \cap F \neq \emptyset$
fin

Déroulement de la reconnaissance d'une expression régulière par un AFN



Considérons le lexème $s = aabb$.

1. E=ϵ-fermeture(0)=$\{0, 1, 2, 4, 7\}$; c='a'
2. Transiter(E,'a')=$\{3, 8\}$;
   ϵ-fermeture(Transiter(E,'a')=$\{1, 2, 3, 4, 6, 7, 8\}$;c='a'
3. Transiter(E,'a')=$\{3, 8\}$;
   ϵ-fermeture(Transiter(E,'a')=$\{1, 2, 3, 4, 6, 7, 8\}$;c='b'
4. Transiter(E,'b')=$\{5, 9\}$;
   ϵ-fermeture(Transiter(E,'a')=$\{1, 2, 4, 5, 6, 7, 9\}$;c='b'
5. Transiter(E,'b')=$\{5, 10\}$;
   ϵ-fermeture(Transiter(E,'a')=$\{1, 2, 4, 5, 6, 7, 10\}$;c='fdf'

Des AFN aux Automates finis déterministes (AFD)

L'utilisation d'un AFN pour reconnaître un lexème est parfaite. En revanche si on doit traiter des milliers ou des millions de chaînes il est certain que l'on effectuera de nombreuses fois les mêmes ϵ-fermeture et les mêmes transitions. Un automate fini déterministe (AFD) est un automate où :

- Aucun état n'a de ϵ-transition
- Pour chaque état e et chaque symbole $a \in \Sigma$ il existe au plus une transition étiquetée 'a' sortant de e.

La reconnaissance d'un lexème par un AFN consiste donc simplement a tester si il existe un chemin de l'état d'entré à un état d'acceptation.

Reconnaissance d'un lexème par un AFD

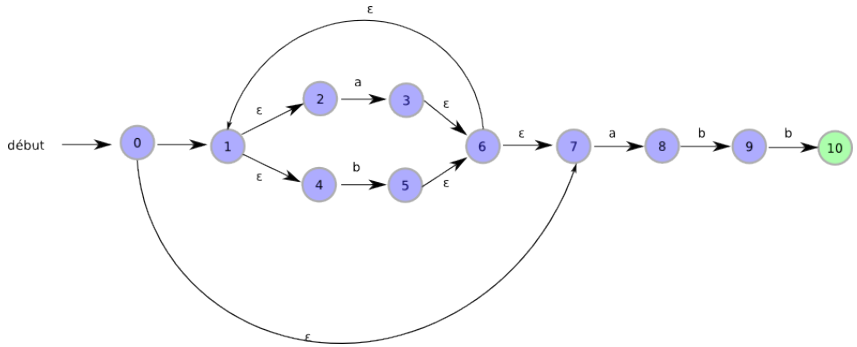
fonction recAFD **(**e_0,F**) :** Booléen
    Déclaration e : état,c : caractère
début
    e $\leftarrow$ e_0
    c $\leftarrow$ carSuivant()
    tant que $c \neq$ fdf **faire**
        e $\leftarrow$ Transiter(e,c)
        c $\leftarrow$ carSuivant()
    fintantque
    retourner $e \in F$
fin

Transformation d'un AFN en AFD

La reconnaissance d'un lexème par un AFD calcule une suite d'ensembles d'états. L'idée de base est de pré calculer ces ensembles d'états et de les stocker dans l'AFD avec leurs transitions (table Dtran).

fonction recAFD **(**AFN(e_0,F)**) :** AFD
    Déclaration Détat : ensemble d'états
début
    D-état $\leftarrow$ ϵ-fermeture($\{e_0\}$)
    tant que il existe T non marqué dans Détat **faire**
        marquer T
        Pour chaque a **dans** Σ **faire**
            U $\leftarrow$ ϵ-fermeture(Transiter(T,a))
            si $U \notin$ Détat **alors**
                Détat $\leftarrow$ Détat$\cup\{U\}$
            finsi
            Dtran[T,a] $\leftarrow$ U
        finpour
    fintantque
    retourner AFD(Dtran,Détat)
fin

Exemple de transformation

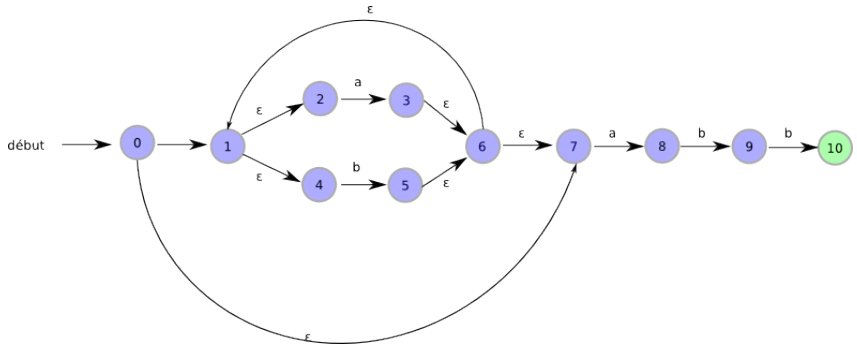


- ϵ-fermeture($\{0\}$)=$\{0, 1, 2, 4, 7\}$=A

Transiter		
Etat	Symbole d'entrée	
	a	b
A		

$$A = \{0, 1, 2, 4, 7\}$$

Exemple de transformation

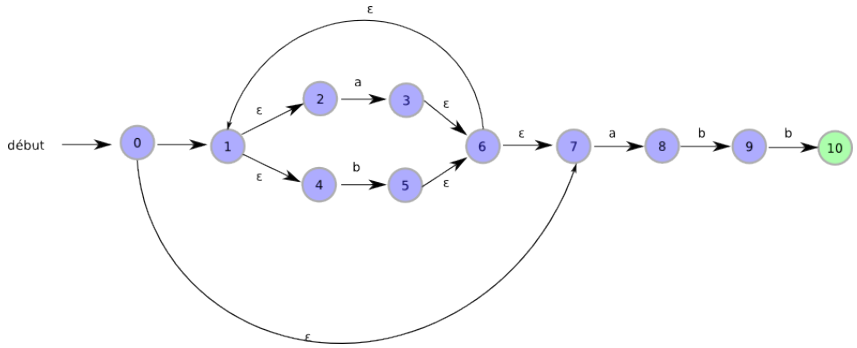


- ϵ-fermeture($\{0\}$)=$\{0, 1, 2, 4, 7\}$=A
- ϵ-fermeture(Transiter(A,a))=ϵ-fermeture($\{3, 8\}$)=$\{1, 2, 3, 4, 6, 7, 8\}$=B

Transiter		
Etat	Symbole d'entrée	
	a	b
A	B	
B		

$$\begin{aligned} A &= \{0, 1, 2, 4, 7\} \\ B &= \{1, 2, 3, 4, 6, 7, 8\} \end{aligned}$$

Exemple de transformation

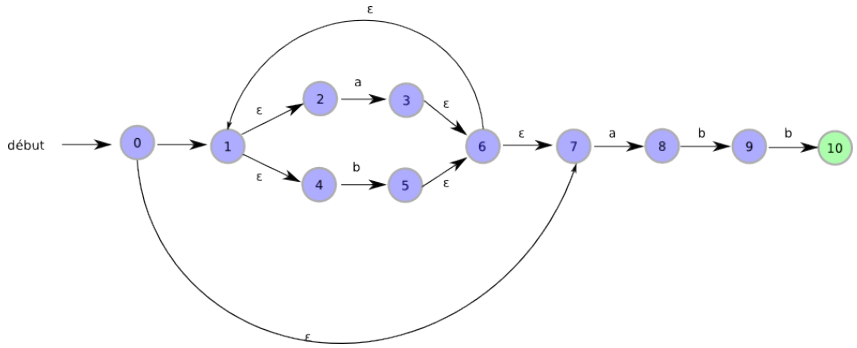


- ϵ-fermeture($\{0\}$)=$\{0, 1, 2, 4, 7\}$=A
- ϵ-fermeture(Transiter(A,b))=ϵ-fermeture($\{5\}$)=$\{1, 2, 4, 5, 6, 7\}$=C

Transiter		
Etat	Symbole d'entrée	
	a	b
A	B	C
B		
C		

$$
\begin{aligned}
A &= \{0, 1, 2, 4, 7\} \\
B &= \{1, 2, 3, 4, 6, 7, 8\} \\
C &= \{1, 2, 4, 5, 6, 7\}
\end{aligned}
$$

Exemple de transformation

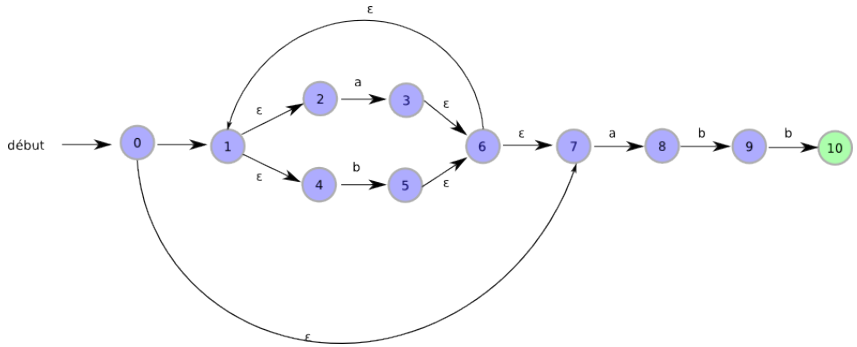


- ϵ-fermeture($\{0\}$)=$\{0, 1, 2, 4, 7\}$=A
- ϵ-fermeture(Transiter(B,a))=ϵ-fermeture($\{3, 8\}$)=B

Transiter		
Etat	Symbole d'entrée	
	a	b
A	B	C
B	B	
C		

$$
\begin{aligned}
A &= \{0, 1, 2, 4, 7\} \\
B &= \{1, 2, 3, 4, 6, 7, 8\} \\
C &= \{1, 2, 4, 5, 6, 7\}
\end{aligned}
$$

Exemple de transformation

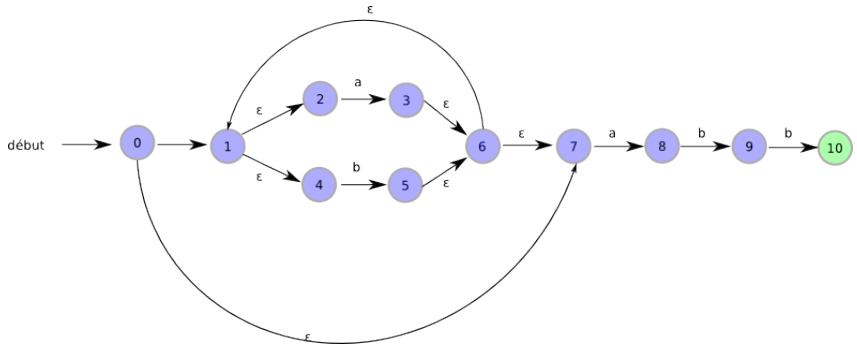


- ϵ-fermeture($\{0\}$)=$\{0, 1, 2, 4, 7\}$=A
- ϵ-fermeture(Transiter(B,b))= ϵ-fermeture($\{5, 9\}$)=$\{1, 2, 4, 5, 6, 7, 9\}$=D

Transiter		
Etat	Symbole d'entrée	
	a	b
A	B	C
B	B	D
C		
D		

$$
\begin{aligned}
A &= \{0, 1, 2, 4, 7\} \\
B &= \{1, 2, 3, 4, 6, 7, 8\} \\
C &= \{1, 2, 4, 5, 6, 7\} \\
D &= \{1, 2, 4, 5, 6, 7, 9\}
\end{aligned}
$$

Exemple de transformation

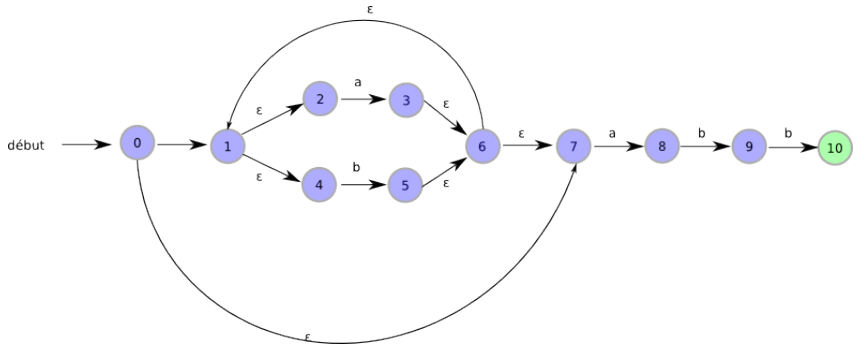


- ϵ-fermeture($\{0\}$)=$\{0, 1, 2, 4, 7\}$=A
- ϵ-fermeture(Transiter(C,a))= ϵ-fermeture($\{3, 8\}$)=B

Transiter		
Etat	Symbole d'entrée	
	a	b
A	B	C
B	B	D
C	B	
D		

$$
\begin{aligned}
A &= \{0, 1, 2, 4, 7\} \\
B &= \{1, 2, 3, 4, 6, 7, 8\} \\
C &= \{1, 2, 4, 5, 6, 7\} \\
D &= \{1, 2, 4, 5, 6, 7, 9\}
\end{aligned}
$$

Exemple de transformation

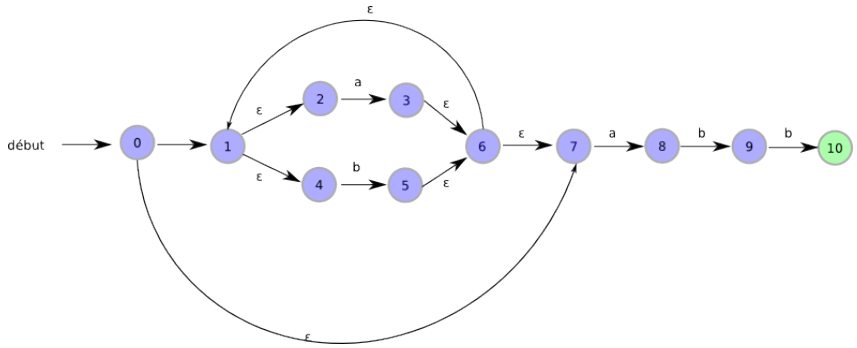


- ϵ-fermeture($\{0\}$)=$\{0, 1, 2, 4, 7\}$=A
- ϵ-fermeture(Transiter(C,b))= ϵ-fermeture($\{5\}$)=C

Transiter		
Etat	Symbole d'entrée	
	a	b
A	B	C
B	B	D
C	B	C
D		

$$
\begin{aligned}
A &= \{0, 1, 2, 4, 7\} \\
B &= \{1, 2, 3, 4, 6, 7, 8\} \\
C &= \{1, 2, 4, 5, 6, 7\} \\
D &= \{1, 2, 4, 5, 6, 7, 9\}
\end{aligned}
$$

Exemple de transformation

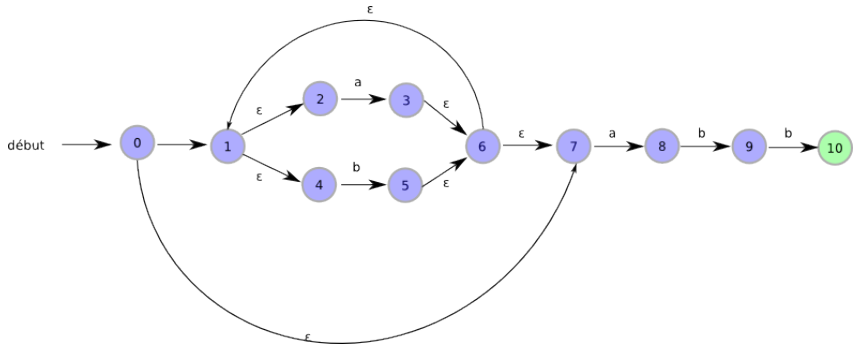


- ϵ-fermeture($\{0\}$)=$\{0, 1, 2, 4, 7\}$=A
- ϵ-fermeture(Transiter(D,a))= ϵ-fermeture($\{3, 8\}$)=B

Transiter		
Etat	Symbole d'entrée	
	a	b
A	B	C
B	B	D
C	B	C
D	B	

$$
\begin{aligned}
A &= \{0, 1, 2, 4, 7\} \\
B &= \{1, 2, 3, 4, 6, 7, 8\} \\
C &= \{1, 2, 4, 5, 6, 7\} \\
D &= \{1, 2, 4, 5, 6, 7, 9\}
\end{aligned}
$$

Exemple de transformation

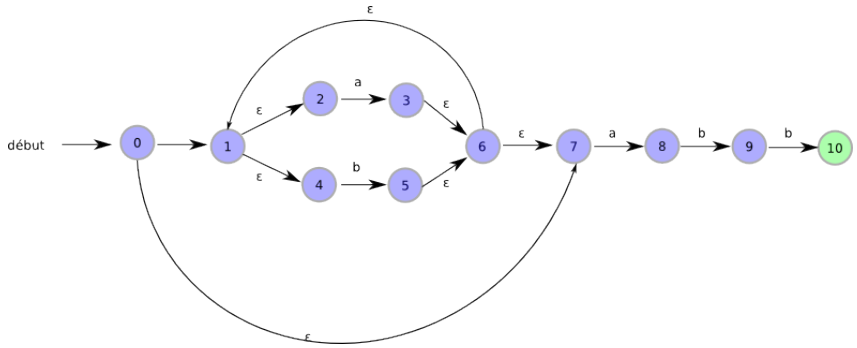


- ϵ-fermeture($\{0\}$)=$\{0, 1, 2, 4, 7\}$=A
- ϵ-fermeture(Transiter(D,b))=
  ϵ-fermeture($\{5, 10\}$)=$\{1, 2, 4, 5, 6, 7, 10\}$=E

Transiter		
Etat	Symbole d'entrée	
	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E		

$$
\begin{array}{lcl}
A & = & \{0, 1, 2, 4, 7\} \\
B & = & \{1, 2, 3, 4, 6, 7, 8\} \\
C & = & \{1, 2, 4, 5, 6, 7\} \\
D & = & \{1, 2, 4, 5, 6, 7, 9\} \\
E & = & \{1, 2, 4, 5, 6, 7, 10\}
\end{array}
$$

Exemple de transformation

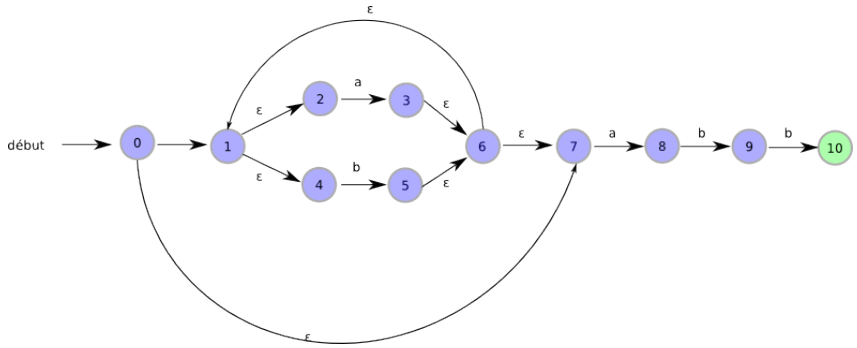


- ϵ-fermeture($\{0\}$)=$\{0, 1, 2, 4, 7\}$=A
- ϵ-fermeture(Transiter(E,a))= ϵ-fermeture($\{3, 8\}$)=B

Transiter		
Etat	Symbole d'entrée	
	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	

$$
\begin{aligned}
A &= \{0, 1, 2, 4, 7\} \\
B &= \{1, 2, 3, 4, 6, 7, 8\} \\
C &= \{1, 2, 4, 5, 6, 7\} \\
D &= \{1, 2, 4, 5, 6, 7, 9\} \\
E &= \{1, 2, 4, 5, 6, 7, 10\}
\end{aligned}
$$

Exemple de transformation

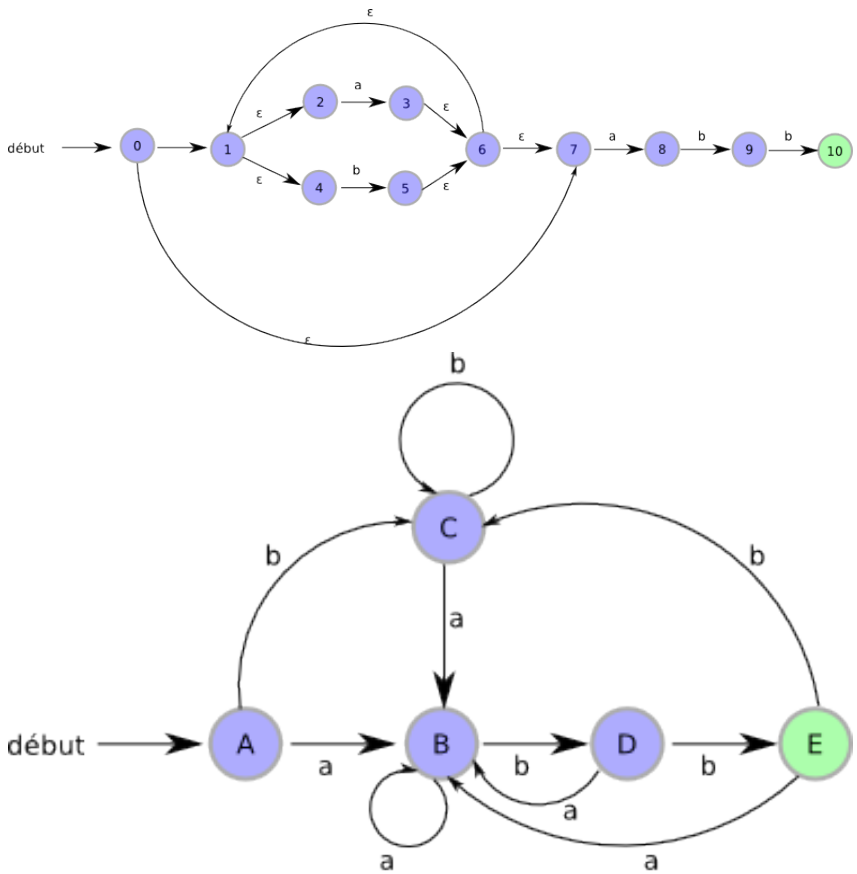


- ϵ-fermeture($\{0\}$)=$\{0, 1, 2, 4, 7\}$=A
- ϵ-fermeture(Transiter(E,b))= ϵ-fermeture($\{5\}$)=C

Transiter		
Etat	Symbole d'entrée	
	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

$$
\begin{array}{lcl}
A & = & \{0, 1, 2, 4, 7\} \\
B & = & \{1, 2, 3, 4, 6, 7, 8\} \\
C & = & \{1, 2, 4, 5, 6, 7\} \\
D & = & \{1, 2, 4, 5, 6, 7, 9\} \\
E & = & \{1, 2, 4, 5, 6, 7, 10\}
\end{array}
$$

Exemple de transformation



Transiter		
Etat	Symbole d'entrée	
	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

Discussion

- La reconnaissance d'un lexème par un AFD est plus rapide (une seule transition par état),
- En revanche le nombre d'états d'un AFD peut être exponentiel par rapport au nombres d'états de l'AFN.

Automate	Place	Temps
AFN	$\mathcal{O}(\|r\|)$	$\mathcal{O}(\|r\| * \|x\|)$
AFD	$\mathcal{O}(2^{\|r\|})$	$\mathcal{O}(\|x\|)$

- Différents compromis ou heuristiques (tel que l'évaluation paresseuse des transitions) sont donc utilisés.

Un outil d'analyse lexicale : Lex

- L'analyseur lexical Lex se combine avec l'analyseur syntaxique Yacc pour produire des compilateurs en C.
- Les cousins de Lex/Yacc pour le C++ se nomment Flex et Bisons.
- L'application de Lex sur un fichier produit un fichier `lex.yy.c` que l'on compile en utilisant la librairie Lex : libl.
- Ci joint un exemple de Makefile :

```
CC      = gcc
LEX     = lex
LEXFILE = monFichier.lex
OBJ     = lex.yy.o
OUTPUT  = nbId
LIB     = -ll

$(OUTPUT) :     $(OBJ)
                $(CC) $(OBJ) -o $@ $(LIB)

lex.yy.o :      lex.yy.c
                $(CC) -c lex.yy.c

lex.yy.c :      $(LEXFILE)
                $(LEX) $(LEXFILE)
```

Syntaxe des expressions régulières en Lex

x	le caractère "x"
.	n'importe quel caractère sauf \n
[xyz]	soit x, soit y, soit z
[^bz]	tous les caractères, sauf b et z
[a-z]	n'importe quel caractère entre a et z
[^a-z]	tous les caractères, sauf ceux compris entre a et z
r*	zéro r ou plus, où r est n'importe quelle expression régulière
r+	au moins un r
r ?	au plus un r (c'est-à-dire un r optionnel)
r{2,5}	entre 2 et 5 r
r{2,}	2 r ou plus
r{2}	exactement 2 r
rs	r suivi de s
r|s	r ou s
r/s	r, seulement s'il est suivi par s
^r	r, mais seulement en début de ligne
r$	r, mais seulement en fin de ligne

Syntaxe des expressions régulières en Lex

{r}	l'expansion de r
"[xyz\"foo"	la chaîne "[xyz"foo"
\X	si X est un "a", "b", "f", "n", "r", "t", ou "v", représente l'interprétation ANSI-C de \X.
\0	le caractère ASCII 0
\123	le caractère dont le code ASCII est 123, en octal
\x2A	le caractère dont le code ASCII est 2A, en hexadécimal
<<EOF>>	fin de fichier

Structure d'un fichier Lex

%{
Déclaration des variables C
%}
Déclaration des Unités lexicales
%%
Déclarations des actions associées à la reconnaissance d'unités lexicales
%%
Code C supplémentaire (déclarations de fonctions auxilières)

Exemple de fichier Lex

```
%{
  int nb_numbers=0,nb_id=0;
%}
pm [+-]
chiffre [0-9]
frac \.{chiffre}+
exp E{pm}?{chiffre}+
nb {pm}?(({chiffre}+{frac}?)|{frac}){exp}?
lettre [a-zA-z]
id {lettre}({lettre}|{chiffre})*
%%
{nb}  { nb_numbers++;printf("Nombre %s reconnu : %f\n",yytext,atof(yyte
{id}  { nb_id++;printf("Id %s reconnu\n",yytext);}
{nb}{id} {printf("Lexical error\n");}
%%
int main()
{
yylex();
 printf("Nb Numbers: %d, \nNb Id %d\n",nb_numbers,nb_id);
return 0;
}
```