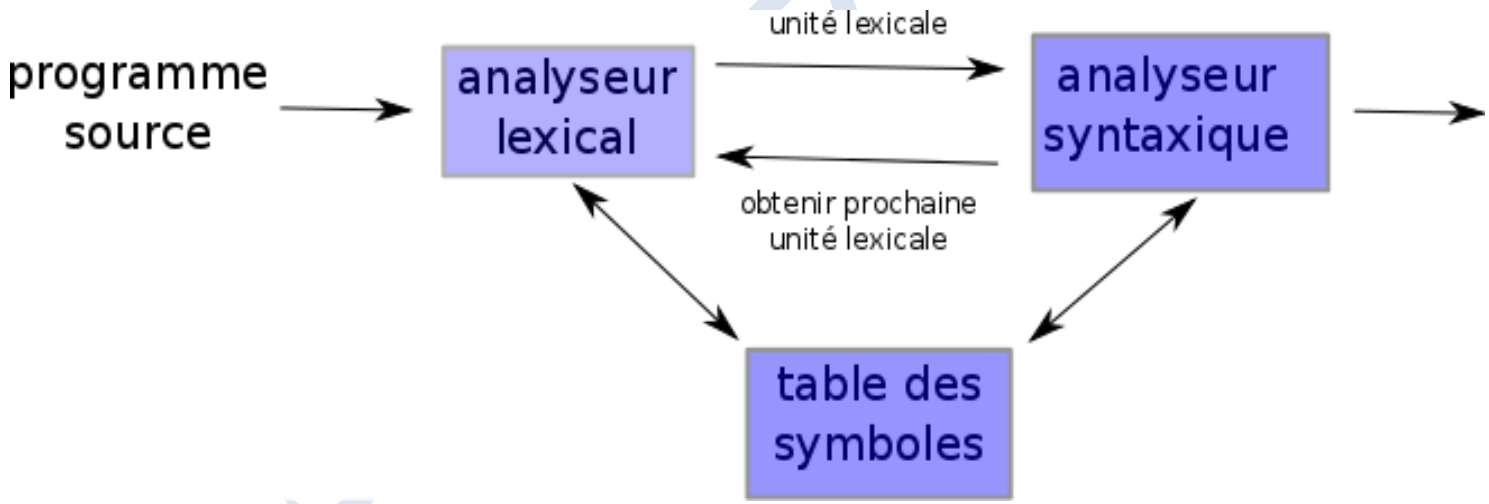
Eléments de théorie des langages

Objectifs:

- Transformation d'un ensemble de caractères en concepts.
- On distingue les concepts suivants : mots clés, opérateurs, identificateurs, symboles de ponctuation, chaînes littérales.

Unité lexicale : correspond à une entité renvoyée par l'analyseur lexical. Par exemple <, >, <=, >= sont tous des opérateurs relationnels.



Eléments de théorie des langages

Alphabet:

- Un alphabet Σ est un ensemble fini de symboles. L'alphabet binaire {0, 1} et les codes ASCII, EBECEDIC sont des exemples d'alphabets.
- 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*.

Eléments de théorie des langages

Langages:

- Un langage est un ensemble a priori quelconque de chaînes construites sur un alphabet. L'ensemble des codes source C, des nombres binaires sont des langages.

- ∅ désigne le langage vide (notation ensembliste).

- 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*. Exemple: x=anti, y=coagulant, xy=anticoagulant.

- Exponentiation : $s^0 = \varepsilon$, $s^1 = s$, $s^n = s^{n-1}s$

Opérations sur les langages

Définitions:

- Union ($L \cup M$) : $L \cup M = \{s \mid s \in L \text{ ou } s \in M\}$
- Concaténation (LM) :

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

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

- Fermeture de Kleene (L^*) :

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

- Fermeture positive (L^+) :

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

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'ensemble 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 définit un langage $L(r)$.

1. ε est une expression régulière qui dénote $L(\varepsilon) = \{\varepsilon\}$,
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 à partir de la construction ci-dessus est régulière.

Expressions régulières

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

- $L(a|b) = \{a, b\}$,
- $L((a|b)(a|b)) = \{aa, ab, ba, bb\}$,
- $L(a^*) = \{\varepsilon, a, a^2, \ldots\}$,
- $\{a, a^2, b, b^2, ab, ab^2, a^2b^2, \ldots\} \subset L((a|b)^*)$,
- On évite des parenthèses inutiles en utilisant les conventions suivantes :
- $*$ a la plus haute priorité,
- la concaténation a 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 $\|$
$\varepsilon r = r\varepsilon = r$	ε est un élément neutre pour la concaténation
$r^* = (r\|\varepsilon)^*$	
$r^{**} = r^*$	$*$ est idempotent

Définitions régulières

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

- Soit Σ un alphabet, et un ensemble de couples

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

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

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

Exemples de définitions régulières

- Identificateurs de variables

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

- Nombres :

$$\begin{aligned} \text{pm} &\rightarrow (+|-) \\ \text{chiffre} &\rightarrow 0|1|2|3|4|5|6|7|8|9 \\ \text{frac} &\rightarrow .\text{chiffre}^+ \\ \text{exp} &\rightarrow E\ (\text{pm}|\varepsilon)\ \text{chiffre}^+ \\ \text{nb} &\rightarrow (\text{pm}|\varepsilon)\ \text{chiffre}^+(\text{frac}|\varepsilon)|\text{frac}\ (\text{exp}|\varepsilon) \end{aligned}$$

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

Notations abrégées et limites des expressions régulières

Notations abrégées:

- L'expression $(r|\varepsilon)$ se note également $r?$. Ainsi $\text{exp} \rightarrow E$ $(\text{pm}|\varepsilon)$ chiffre^+, se note également $\text{exp} \rightarrow E$ $\text{pm}?$ chiffre^+,

- $(a|b|c)$ se note également $[abc]$. De même $(a|b|\ldots,|z)$ se note $[a-z]$ et $[a-zA-Z\ 0-9]$ 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:

- Les expressions régulières ne permettent de coder qu'un nombre limité de langages.
- Les expressions régulières ne permettent pas de compter. Ainsi le langage:

$L = \{wcw \mid w \in L(\{a, b\}^*)\}$ n'est pas régulier.

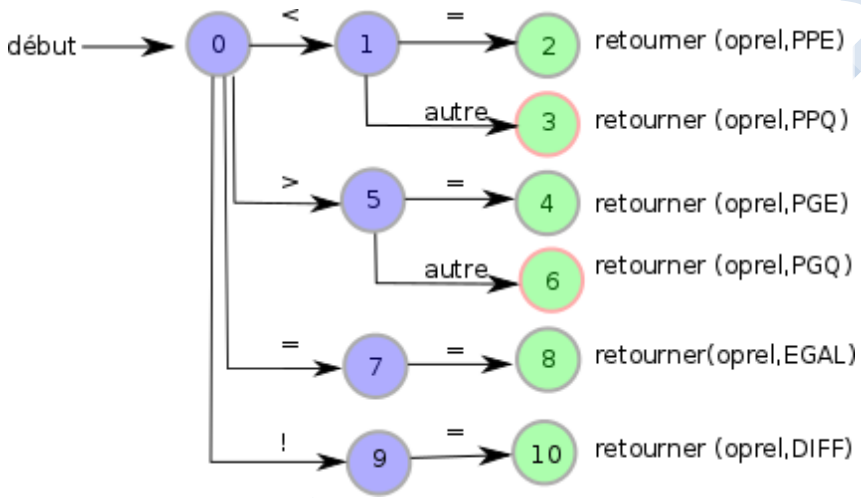
Unités lexicales et diagrammes de transition

Un diagramme de transitions 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'entrée lorsqu'il est appelé par l'analyseur syntaxique.

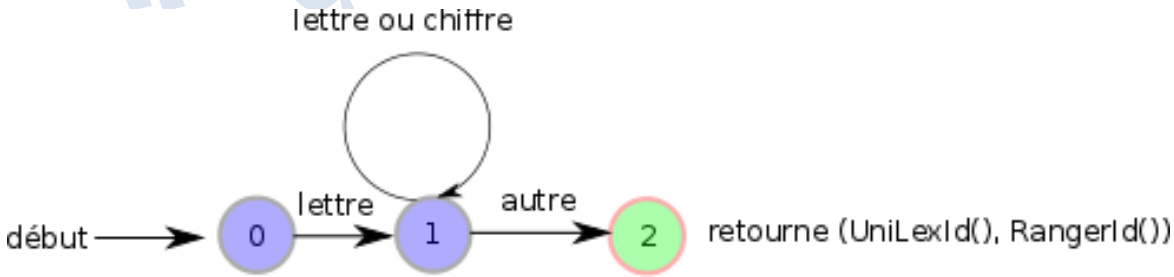
- Un diagramme de transitions est constitué d'états et d'arcs :
- Etats non finaux.
- Etats finaux ou états d'acceptation.
- Les arcs codent les transitions entre états. Ils ont des étiquettes indiquant sur qu'elle entrée s'effectue chaque changement d'état. L'étiquette autre, code tout caractère autre que ceux indiqués par les changements d'états sur l'état courant. On définit les diagrammes déterministes (i.e. une même étiquette ne peut envoyer sur deux états différents).

Diagrammes de transitions : Exemples

- oprel → (< | > | <= | >= | == |! =)



- id → lettre(lettre|chiffre)*.



Automates Finis Non déterministes (AFN)

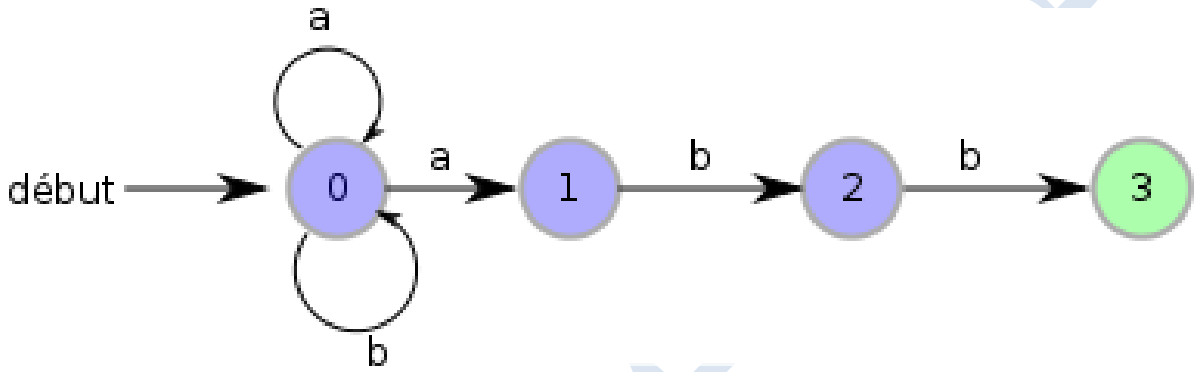
Un AFN est un modèle mathématique défini par :

- Un ensemble d'états E,
- Un ensemble de symboles d'entrée 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 :

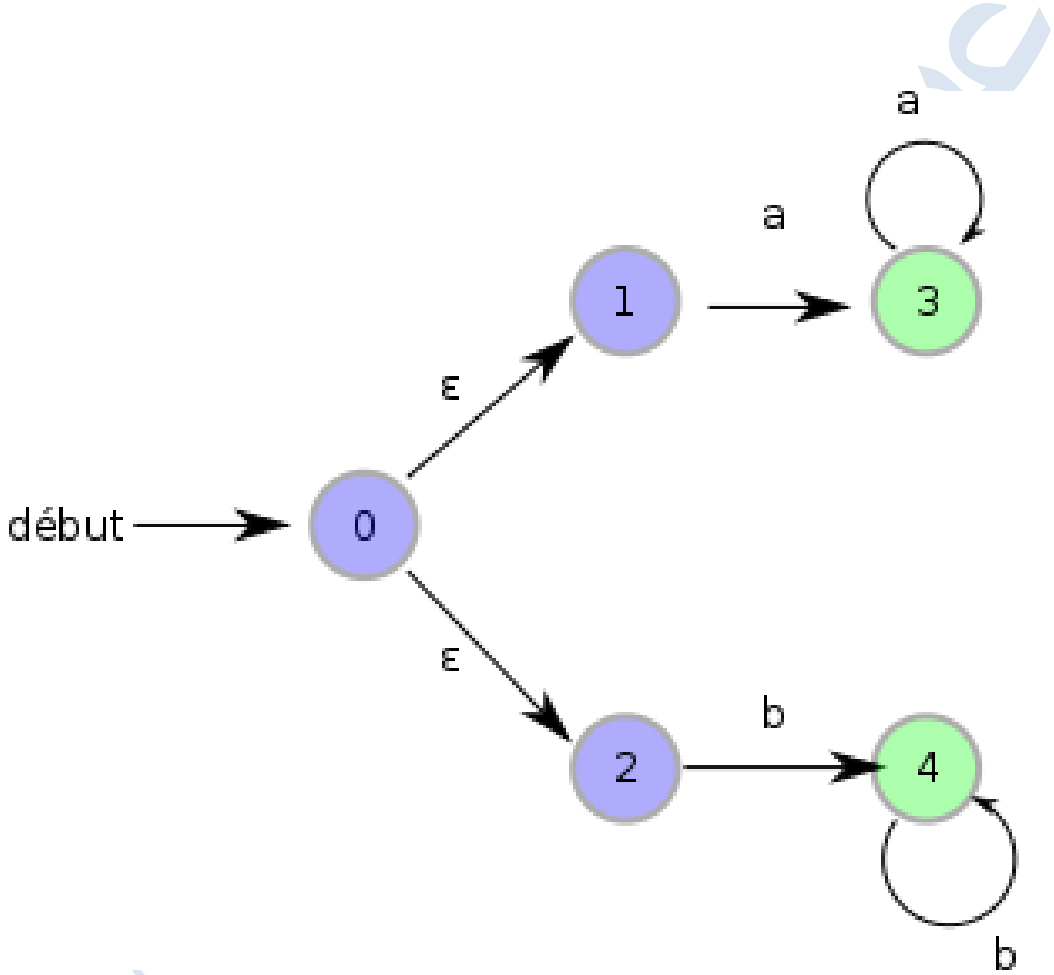


associé à la table de transitions:

Transiter		
Etat	Symbole d'entrée	
	a	b
0	{0,1}	0
1	—	2
2	—	3

AFN : Un second exemple

- Soit l'unité lexicale $aa^*|bb^*$

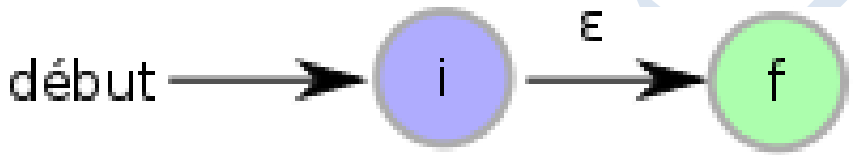


Notez les transitions Ɛ sur l'état de départ et les boucles sur les états d'acceptation.

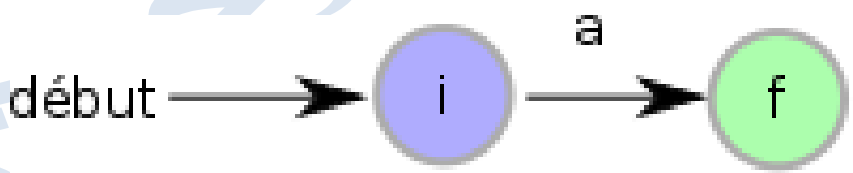
Construction d'un AFN à partir d'une expression régulière

Construction récursive:

1. Pour Ɛ construire l'AFN :



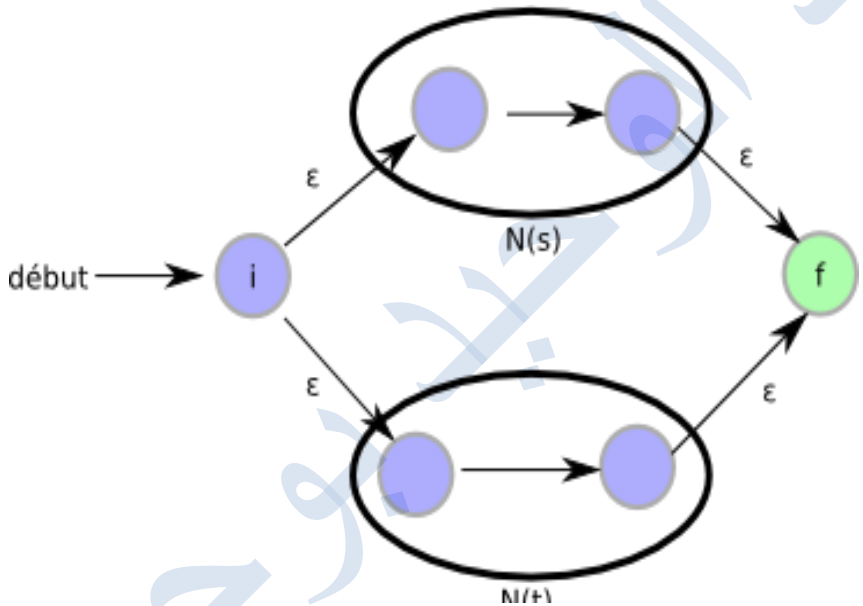
2. Pour $a \in \Sigma$ construire l'AFN :



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

Construction récursive:

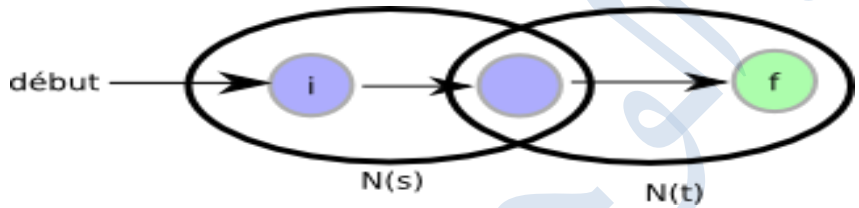
3. Soient $N(s)$ et $N(t)$ les AFN des expressions s et t. L'AFN $N(s|t)$ est défini par :



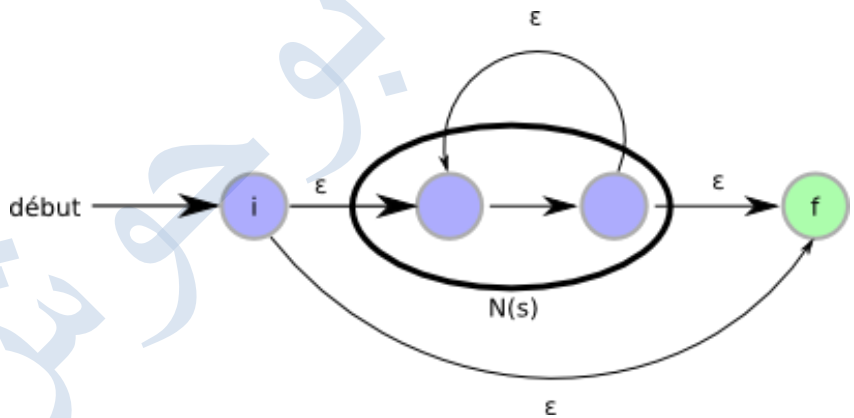
Construction d'un AFN à partir d'une expression régulière

Construction récursive:

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



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.

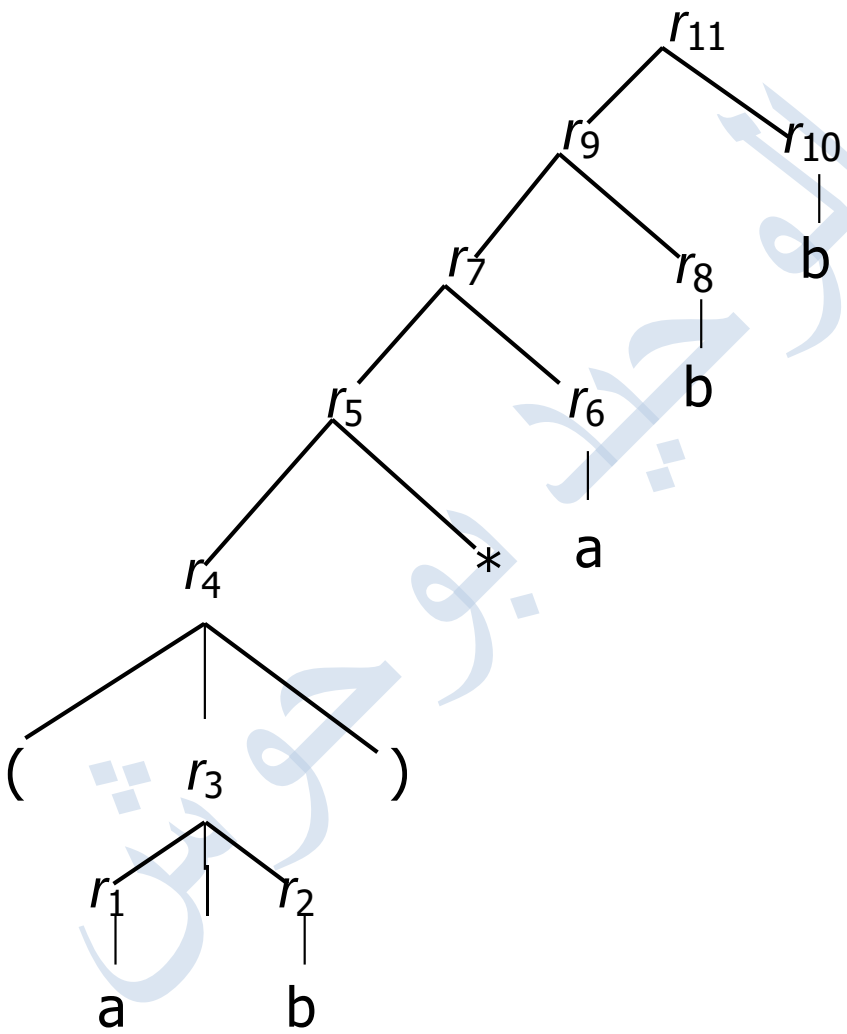
Propriétés de l'AFN construit

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

- $N(r)$ 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 plusieurs transitions sortantes sur ε.

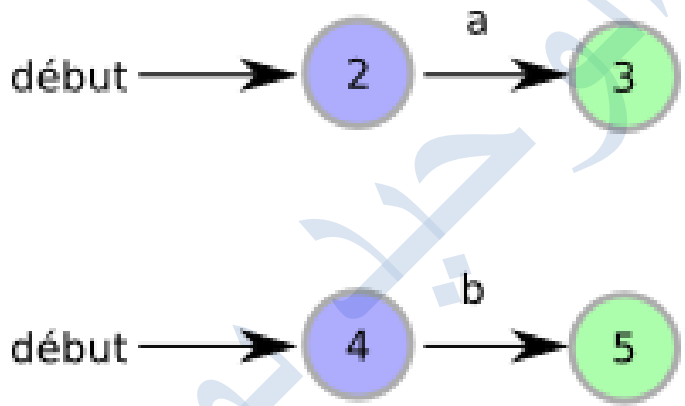
Exemple

- Évaluation de $(a|b)^*abb$. L'arbre syntaxique associé est:



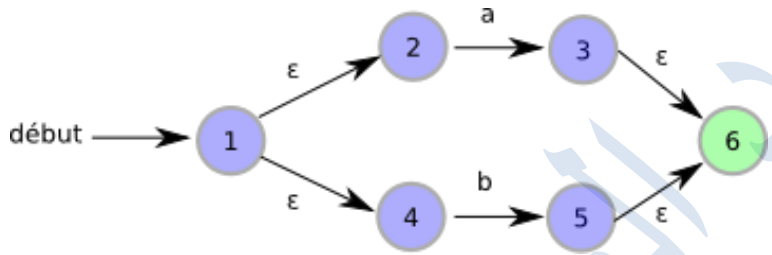
Exemple de construction d'AFN

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

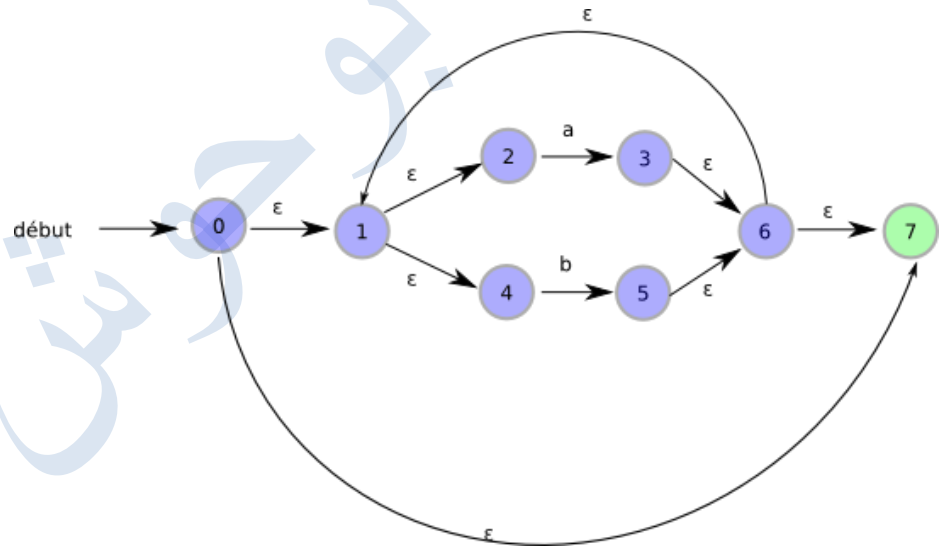


Exemple de construction d'AFN

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

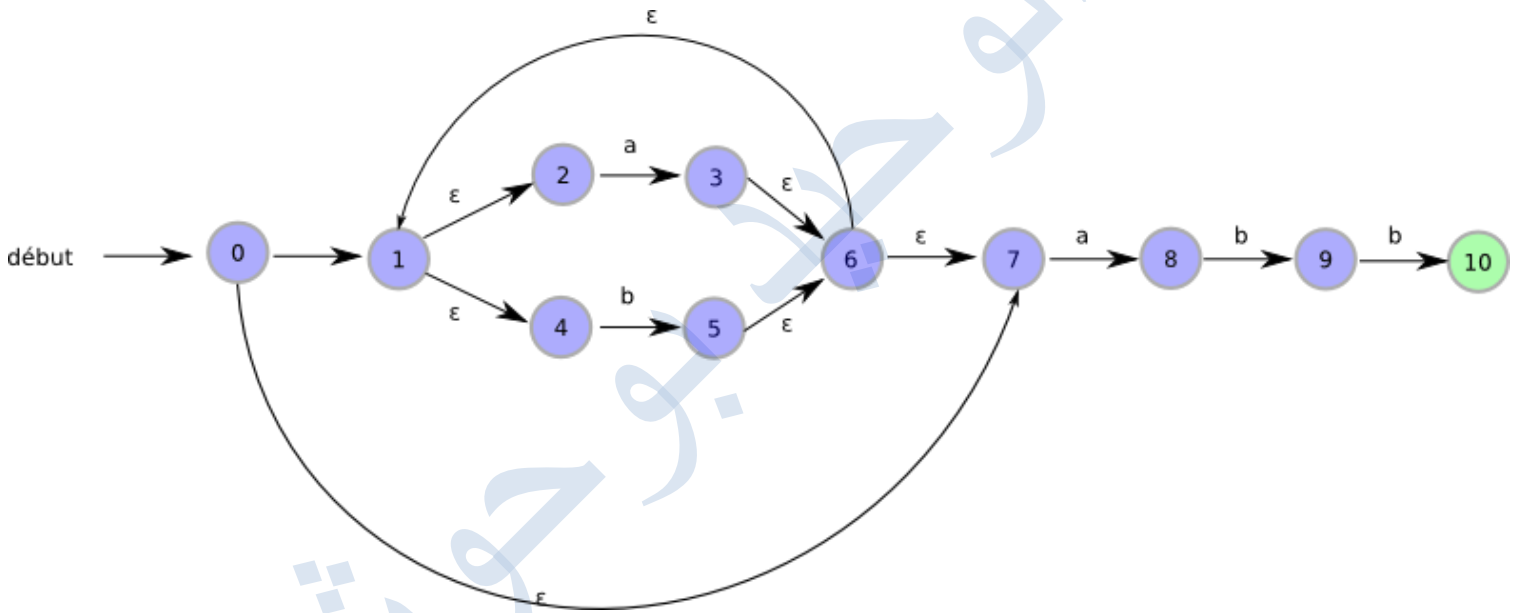


- $N(r_4) = N(r_3)$, construisons l'AFN de $r_5 = (r_1|r_2)^*$



Exemple de construction d'AFN

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



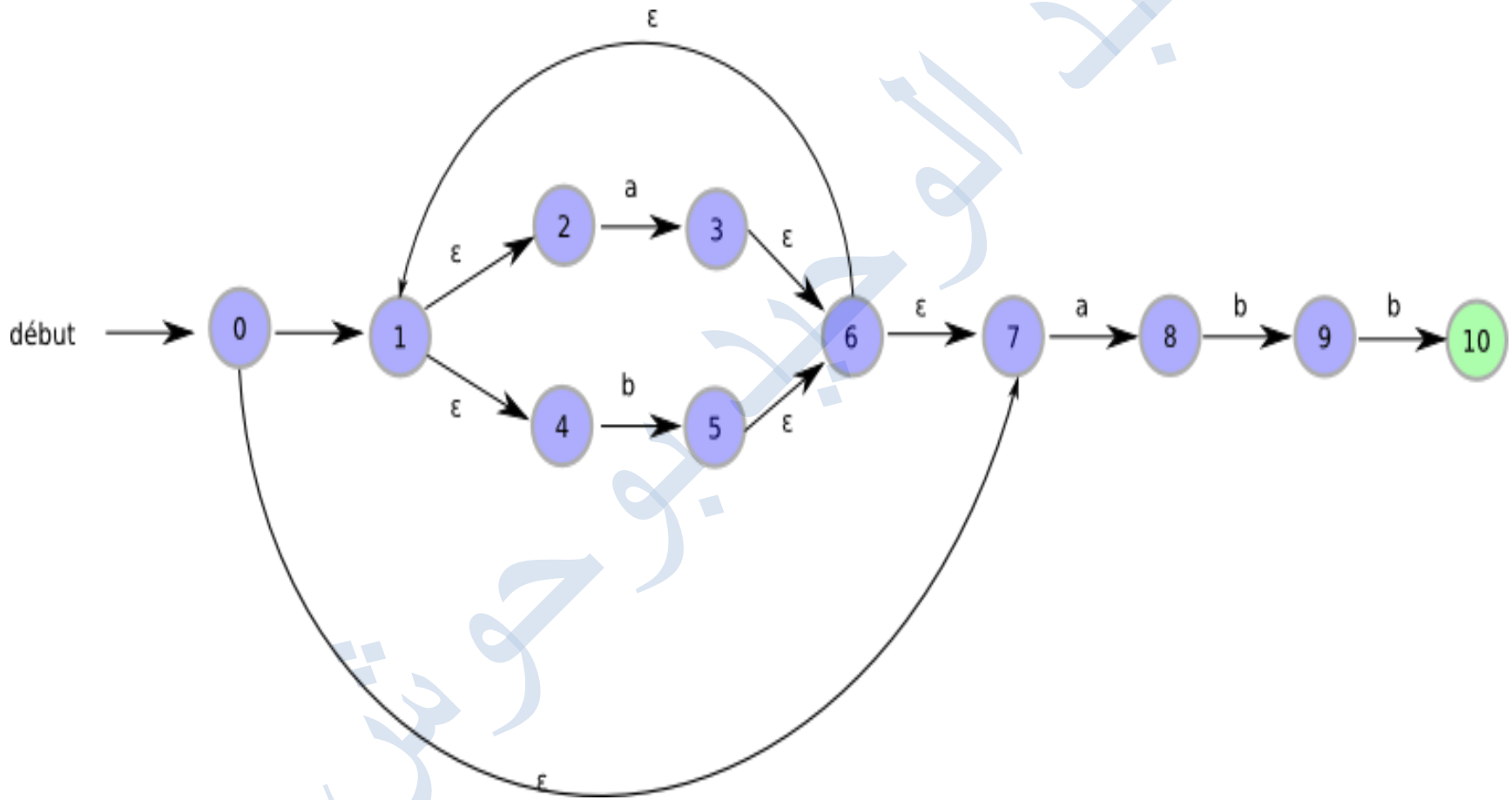
Reconnaissance d'une expression régulière par un AFN

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'.
- Ɛ-fermeture(E) : renvoie l'ensemble des états accessibles depuis E par une Ɛ-transition.

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

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



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

Considérons le lexème *s* = *aabb*.

1. E=Ɛ-fermeture(0)={0, 1, 2, 4, 7} ; c='a'
2. Transiter(E,'a')={3, 8} ;
   Ɛ-fermeture(Transiter(E,'a')={1, 2, 3, 4, 6, 7, 8} ;c='a'
3. Transiter(E,'a')={3, 8} ;
   Ɛ-fermeture(Transiter(E,'a')={1, 2, 3, 4, 6, 7, 8} ;c='b'
4. Transiter(E,'b')={5, 9} ;
   Ɛ-fermeture(Transiter(E,'a')={1, 2, 4, 5, 6, 7, 9} ;c='b'
5. Transiter(E,'b')={5, 10} ;
   Ɛ-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 Ɛ-fermetures et les mêmes transitions. Un automate fini déterministe (AFD) est un automate où:

- Aucun état n'a de Ɛ-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 AFD consiste donc simplement à tester s'il existe un chemin de l'état de départ à un état d'acceptation.

Reconnaissance d'un lexème par un AFD

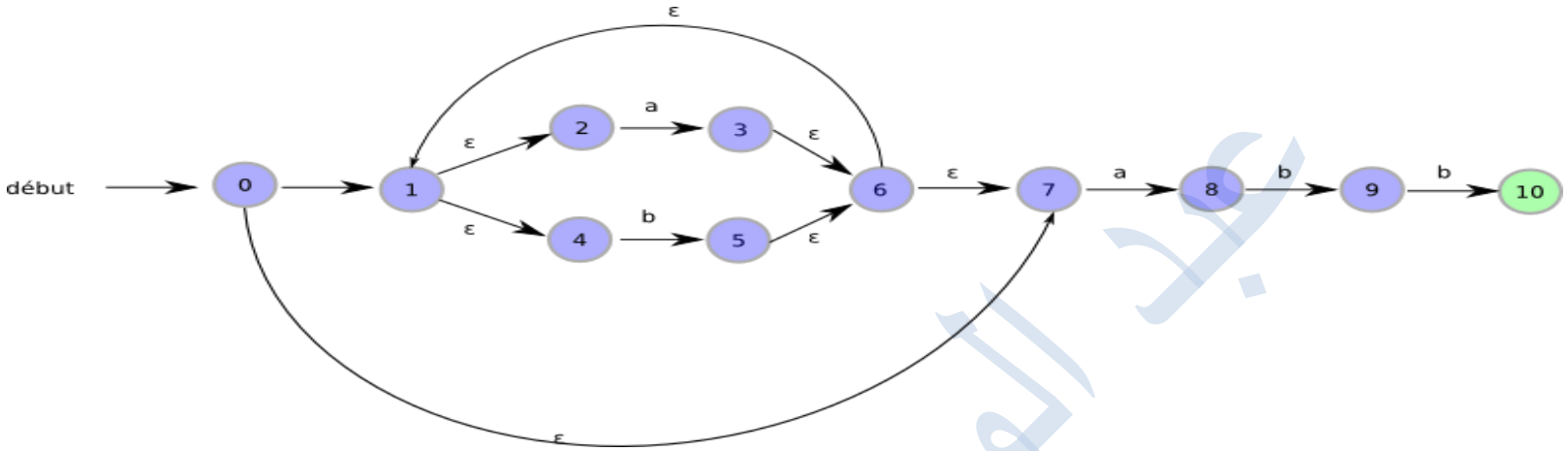
fonction recAFD (e_0,F) : Booléen
    Déclaration e : état, c : caractère
début
    e ← e_0
    c ← carSuivant()
    tant que $c \neq$ fdf faire
     e ← Transiter(e,c)
     c ← carSuivant()
    fintantque
    retourner $e \in F$
fin

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 Dtrans).

fonction recAFD (AFN(e_0,F)) : AFD
    Déclaration Détats : ensemble d'états
début
    Détats ← Ɛ-fermeture({e_0})
    tant que il existe *T* non marqué dans Détats faire
        marquer T
        Pour chaque *a* dans Σ faire
            U ← Ɛ-fermeture(Transiter(T, a))
            si *U* ∉ Détats alors
                Détats ← Détats ∪ {*U*}
            finsi
            Dtrans[T,a] ← U
        Finpour
    Fintantque
    retourner AFD(Dtrans, Détats)
fin

Exemple de transformation

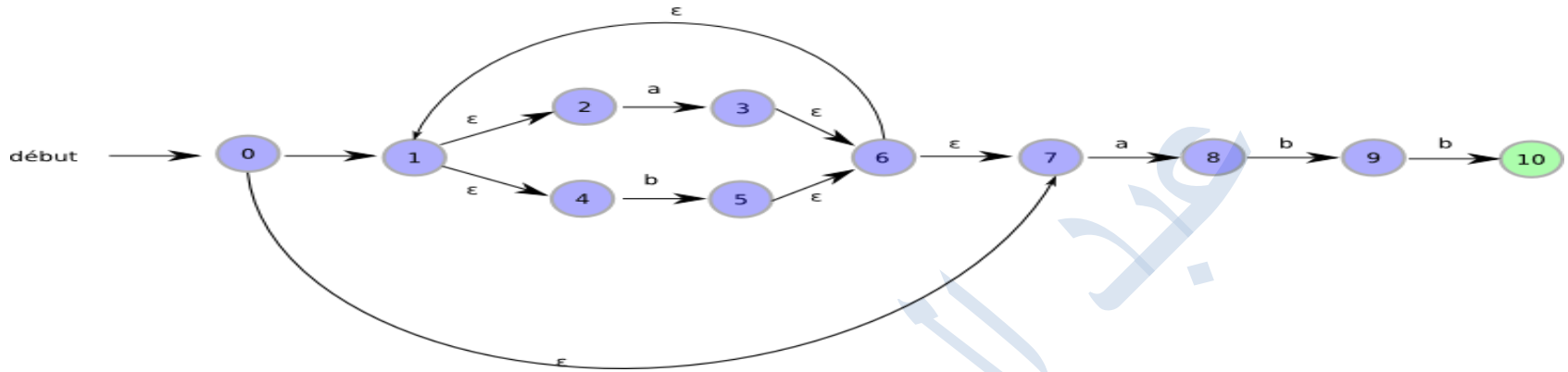


- Ɛ-fermeture({0})={0, 1, 2, 4, 7}=A

Transiter		
Etat	Symbole d'entrée	
	a	b
A		

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

Exemple de transformation



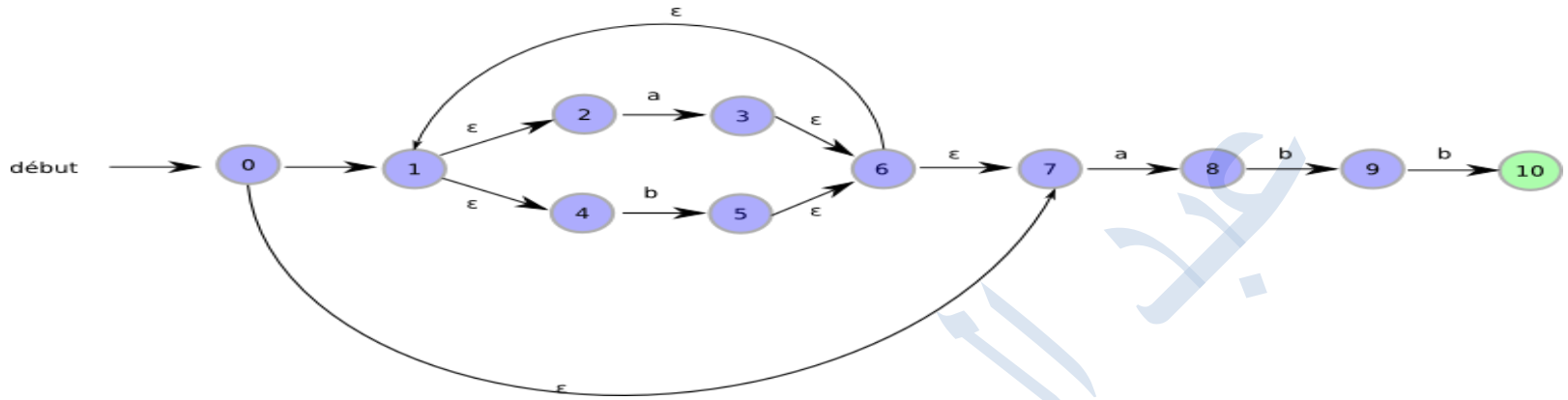
- Ɛ-fermeture({0})={0,1,2,4,7}=A
- Ɛ-fermeture(Transiter(A,a))=Ɛ-fermeture({3, 8})={1, 2, 3, 4, 6, 7, 8}=B

Transiter		
Etat	Symbole d'entrée	
	a	b
A	B	
B		

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

B = {1,2,3,4,6,7,8}

Exemple de transformation

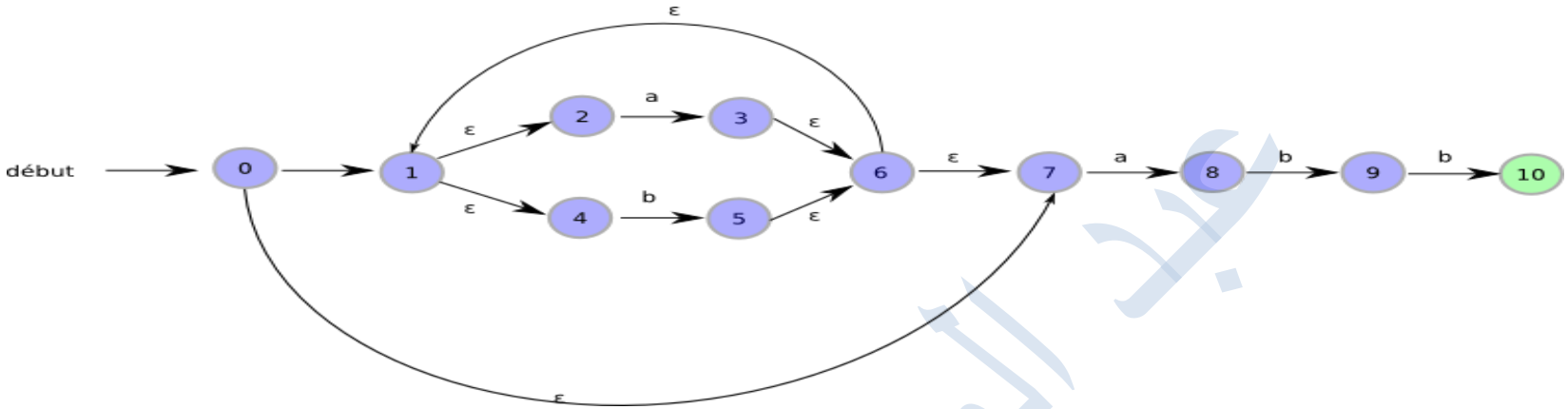


- Ɛ-fermeture({0})={0, 1, 2, 4, 7}=A
- Ɛ-fermeture(Transiter(A,b))=Ɛ-fermeture({5})={1, 2, 4, 5, 6, 7}=C

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

A = {0, 1, 2, 4, 7}
B = {1, 2, 3, 4, 6, 7, 8}
C = {1, 2, 4, 5, 6, 7}

Exemple de transformation

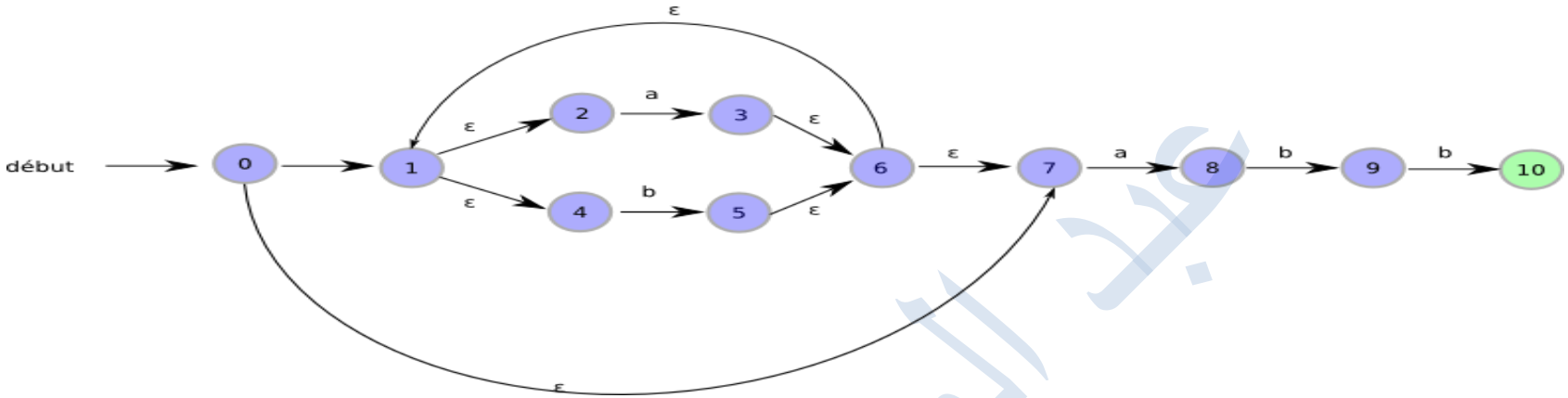


- Ɛ-fermeture({0})={0, 1, 2, 4, 7}=A
- Ɛ-fermeture(Transiter(B,a))=Ɛ-fermeture({3, 8})=B

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

A = {0, 1, 2, 4, 7}
B = {1, 2, 3, 4, 6, 7, 8}
C = {1, 2, 4, 5, 6, 7}

Exemple de transformation

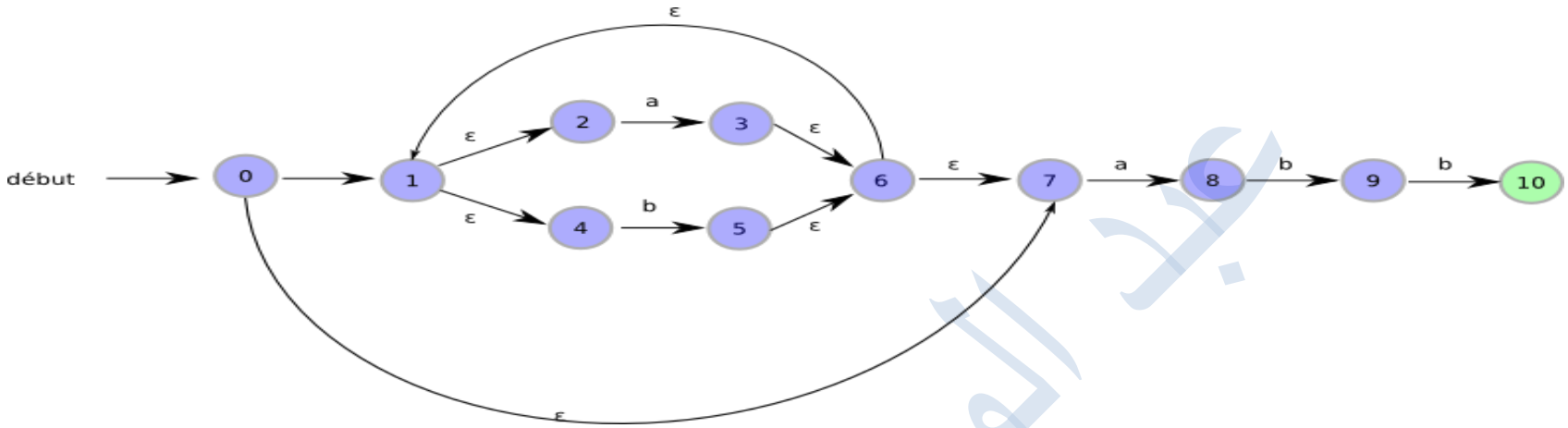


- Ɛ-fermeture({0})={0, 1, 2, 4, 7}=A
- Ɛ-fermeture(Transiter(B,b))= Ɛ-fermeture({5, 9})={1, 2, 4, 5, 6, 7, 9}=D

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

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}

Exemple de transformation

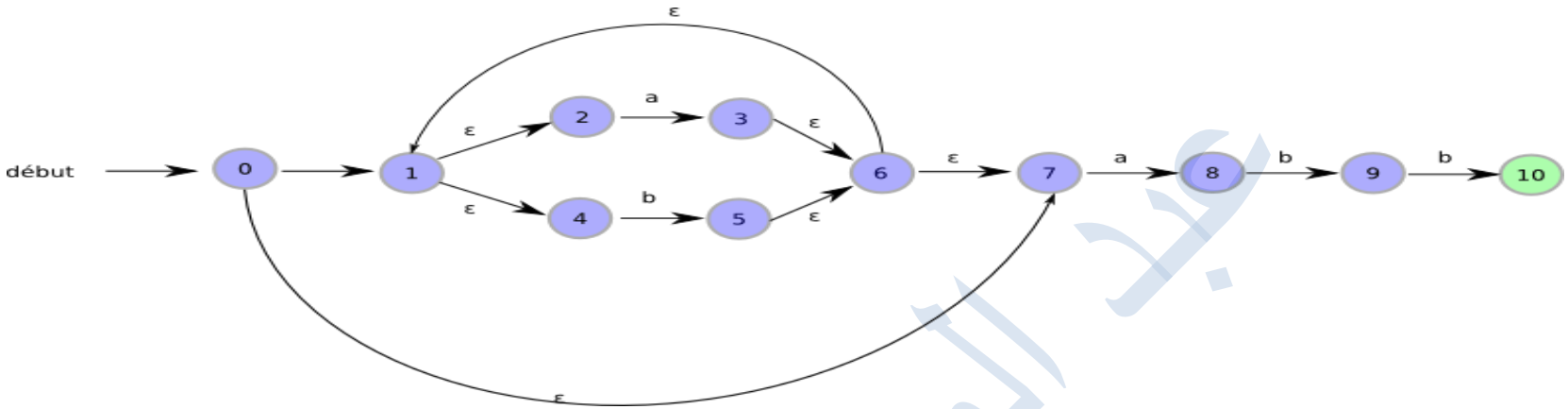


- Ɛ-fermeture({0})={0, 1, 2, 4, 7}=A
- Ɛ-fermeture(Transiter(C,a))= Ɛ-fermeture({3, 8})=B

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

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}

Exemple de transformation

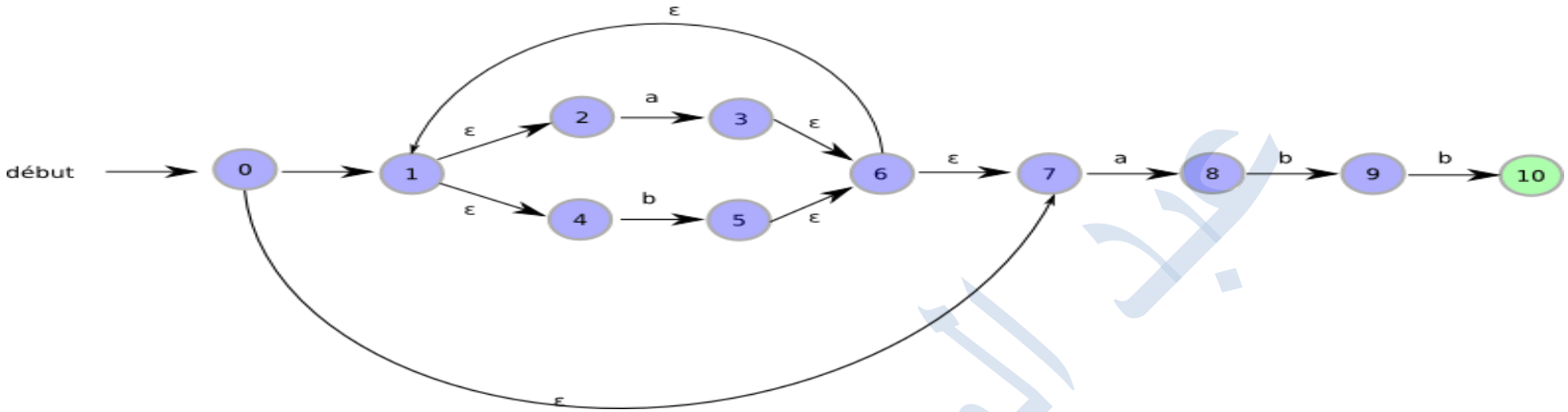


- Ɛ-fermeture({0})={0, 1, 2, 4, 7}=A
- Ɛ-fermeture(Transiter(C,b))= Ɛ-fermeture({5})=C

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

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}

Exemple de transformation

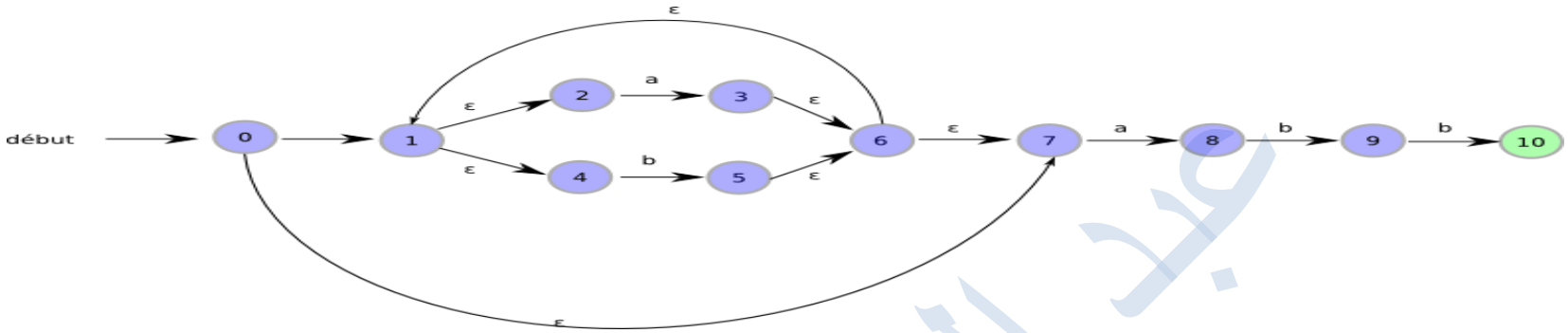


- Ɛ-fermeture({0})={0, 1, 2, 4, 7}=A
- Ɛ-fermeture(Transiter(D,a))= Ɛ-fermeture({3, 8})=B

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

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}

Exemple de transformation

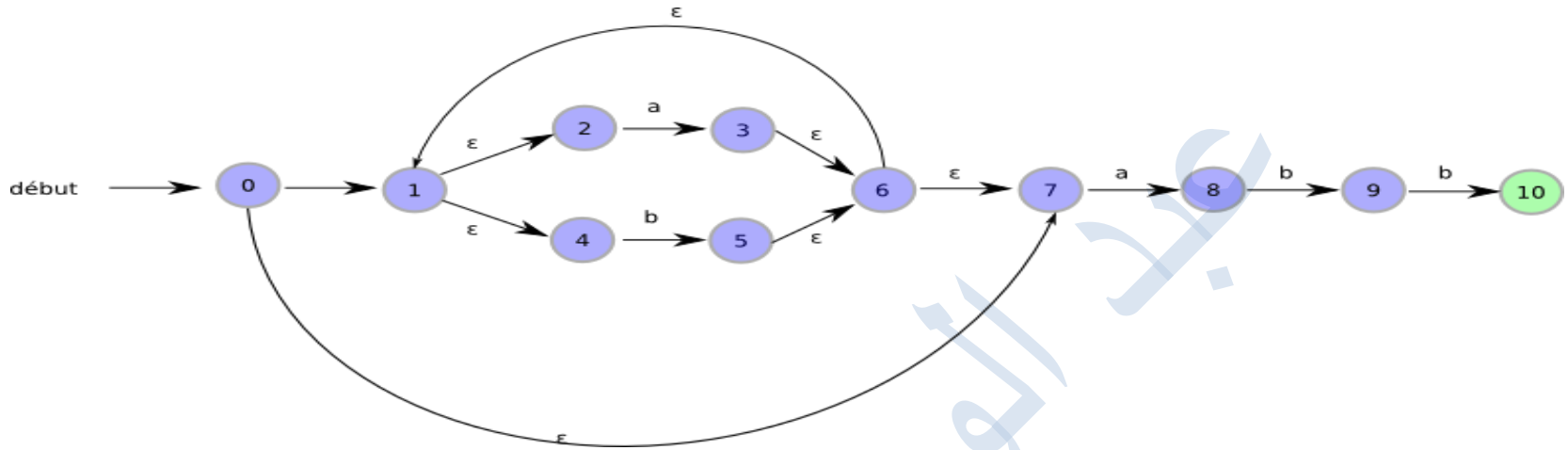


- Ɛ-fermeture({0})={0, 1, 2, 4, 7}=A
- Ɛ-fermeture(Transiter(D,b))= Ɛ-fermeture({5, 10})={1, 2, 4, 5, 6, 7, 10}=E

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

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}

Exemple de transformation

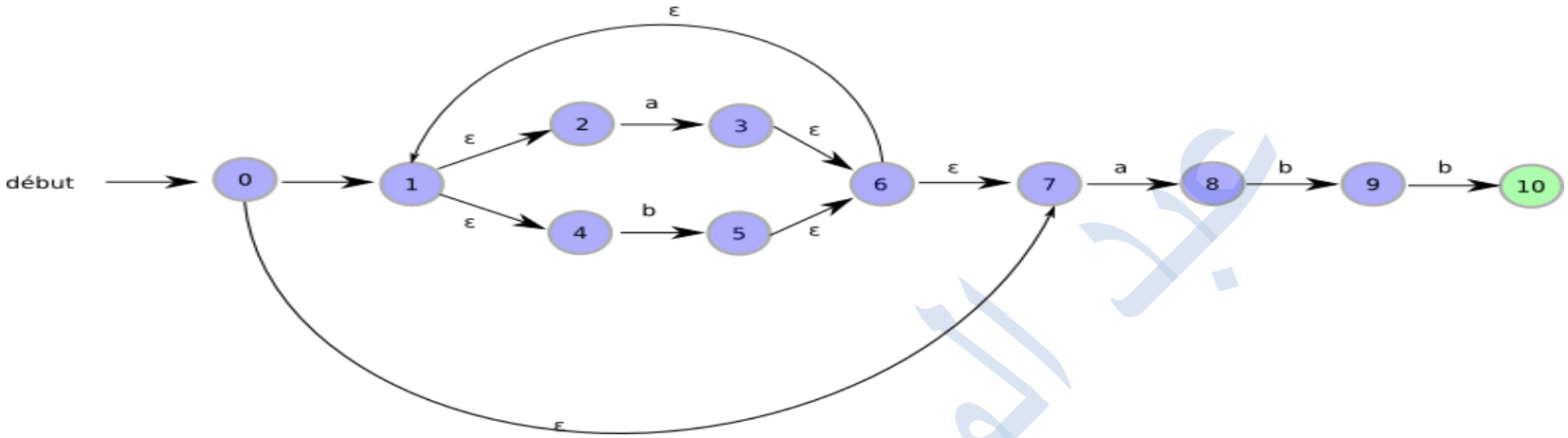


- Ɛ-fermeture({0})={0, 1, 2, 4, 7}=A
- Ɛ-fermeture(Transiter(E,a))= Ɛ-fermeture({3, 8})=B

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

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}

Exemple de transformation

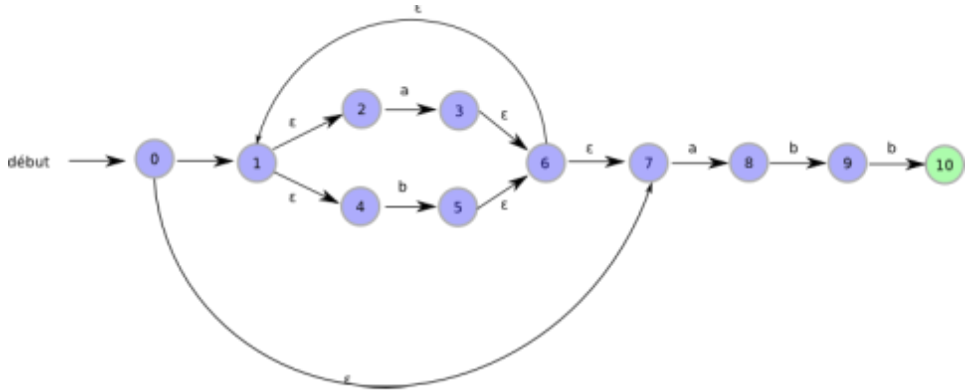


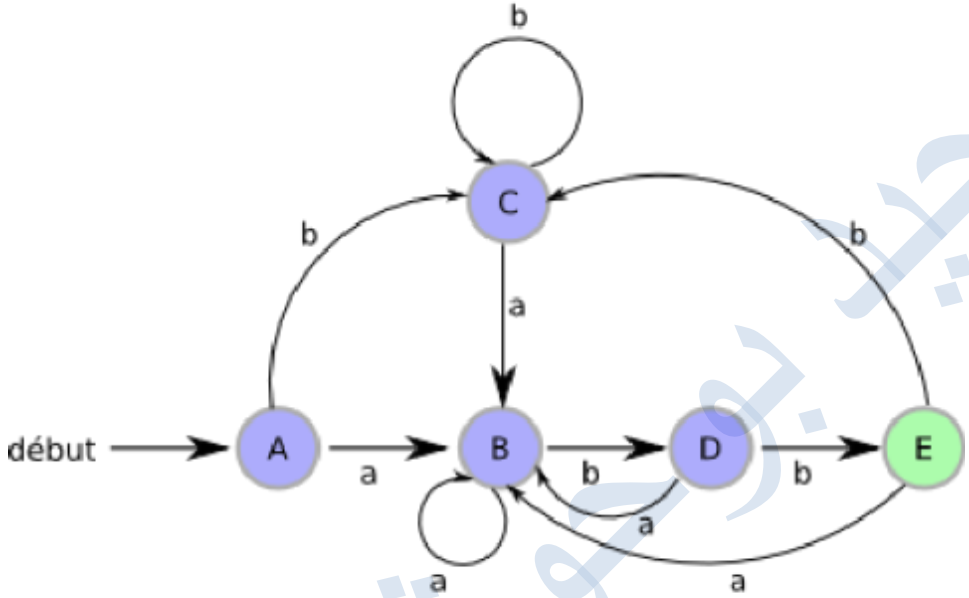
- Ɛ-fermeture({0})={0, 1, 2, 4, 7}=A
- Ɛ-fermeture(Transiter(E,b))= Ɛ-fermeture({5})=C

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

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}

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

Minimisation d'un AFD

- Chaque ensemble régulier est reconnu par un AFD qui est unique à nombre d'états minimal.

Algorithme Minimisation d'un AFD

Donnée: un AFD D avec:

- Un ensemble d'états E,
- Un alphabet d'entrée $\sum$,
- Des transitions pour tous les états,
- Un état de départ e_0,
- Un ensemble d'états d'acceptation F.

Résultat: un AFD D' ayant le plus petit nombre possible d'états.

Minimisation d'un AFD

Méthode:

1. Construire une partition initiale $\prod$ de l'ensemble des états avec deux groupes :
   - Les états d'acceptation F,
   - Les états de non-acceptation S-F.
2. Appliquer la procédure suivante à $\prod$ pour construire une nouvelle partition $\prod_n$

   Pour chaque groupe G de $\prod$ **Faire**
   - Partitionner G en sous-groupes de manière que 2 états e et t de G soient dans le même sous-groupe ssi pour tout symbole ***a***, les états ***e*** et ***t*** ont des transitions sur ***a*** vers des états du même groupe de $\prod$
   - Remplacer G dans $\prod_n$ par tous les sous-groupes ainsi formés

   Finpour

Minimisation d'un AFD

3. **Si** $\prod_n \neq \prod$ **alors** aller à (2) avec $\prod = \prod_n$
   Sinon aller à (4) en posant $\prod_f = \prod_n$

4. Choisir un état de chaque groupe de $\prod_f$ en tant que représentant de ce groupe. Les représentants seront les états de l'AFD réduit D'. Soit ***e*** un état représentatif, et supposons que sur la chaîne d'entrée ***a*** il y ait une transition de ***e*** vers ***t***.

   Soit ***r*** le représentant de ***t*** (***r*** peut être égal à ***t***) alors D' a une transition de ***e*** vers ***r*** sur ***a***. L'état de départ de D' est le représentant du groupe qui contient e_0 de D. Les états d'acceptation de D' sont les représentants des états de F.

5. Si D' a un état mort (càd. Un état qui n'est pas un état d'acceptation et qui a des transitions vers lui –même sur tous les symboles d'entrée), alors supprimer le de D'. Supprimer aussi tout état non accessible depuis l'état de départ.

Minimisation d'un AFD

Revenons à l'AFD précédent.

$\prod$= { (E), (A, B, C, D)}.

Le groupe (E) contient un seul élément donc (E) sera dans $\prod_n$.

Prenons le groupe (A, B, C, D). Sur ***a*** tous ces états transitent vers B.

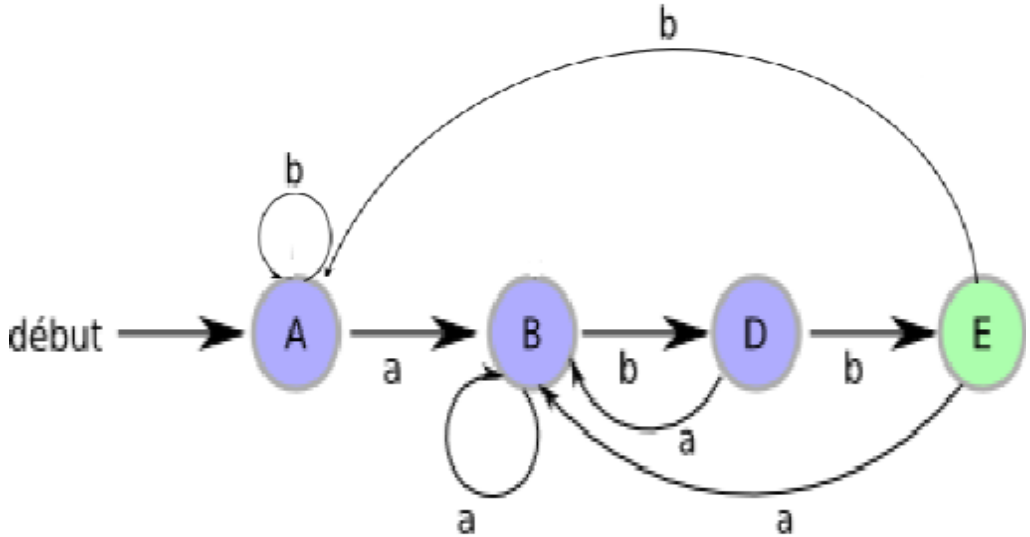
Par contre sur ***b*** (A, C) transitent vers C, B transite vers D et D transite vers E.

Ainsi $\prod_n$={(E), (A, C), (B), (D)}

Si on choisit A comme représentant du groupe (A, C) et B, D et E pour les groupes singletons, on obtient l'AFD réduit dont la table de transitions est ci-dessous.

Exemple de minimisation d'un AFD

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



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 nombre d'états de l'AFN.

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

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

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 Le

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

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;
 }
```