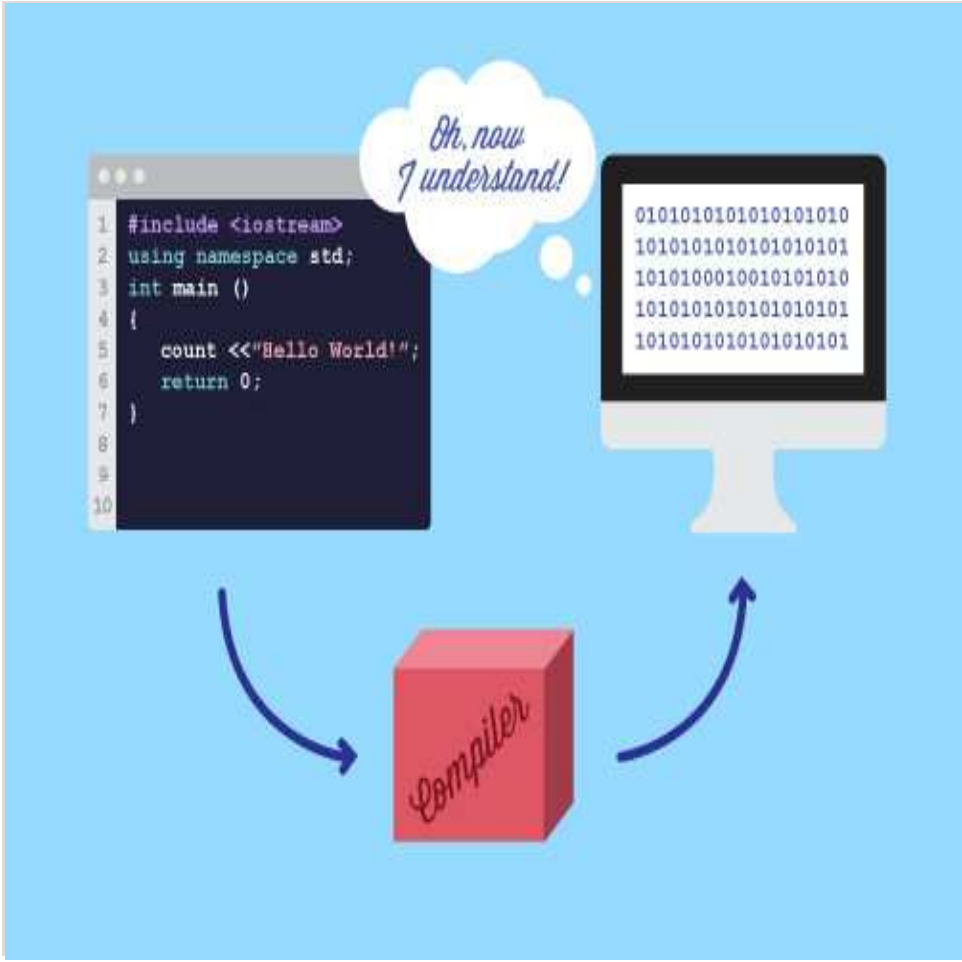
TECHNIQUES DE COMPILATION



2ème SI
Houda Benali

ISSAT Mateur 2020/2021

2 - THÉORIE DES LANGAGES : NOTIONS DE BASE

Les automates finis

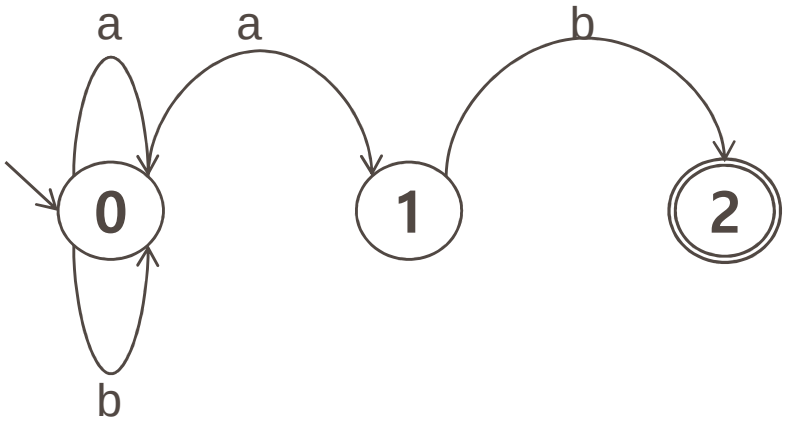
- Modèle permettant de répondre “oui” ou “non” selon que la chaîne de caractères en entrée répond ou non à un modèle donné d’expression régulière.
- Un automate fini est une généralisation des diagrammes de transition
- Deux modèles équivalents, déterministe (AFD) et non déterministe (AFN).

Les automates finis

- Automate fini non déterministe (AFN) ou déterministe (AFD) :
  - Ensemble fini d'états E ;
  - Alphabet d'entrée fini Σ
  - Fonction de transition T
  - État initial e0 ;
  - Ensemble d'états terminaux F ;
- AFD : T : $E \times \Sigma \rightarrow E$
  - au plus une transition par couple état-lettre
- AFN : T : $E \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^E$
  - plusieurs transitions possibles par couple ;
  - possibilités de transitions vides ou ε-transitions.

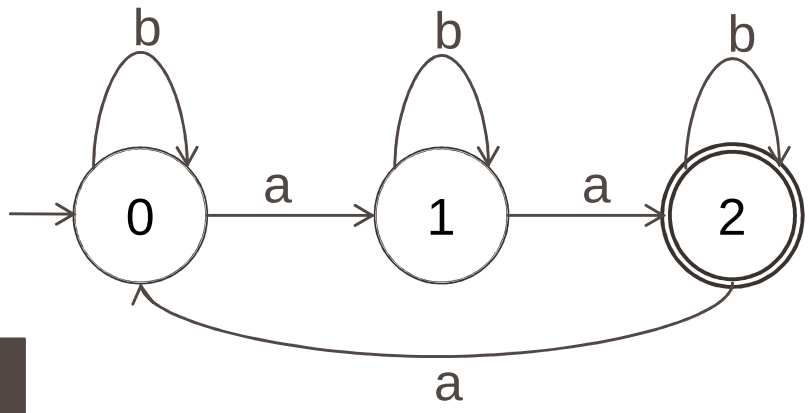
Les automates finis

- **Langage reconnu par un automate fini**
  - N : automate fini, L(N) langage reconnu par N.
  - Un langage reconnu par un automate est l'ensemble de chaines qui permettent de passer de l'état initial à l'état terminal.
  - ***Exemple :*** $\Sigma = \{a, b\}$; ER = $(a|b)^*ab$ ➔ AFN



Les automates finis

- ***Exemple avec table de transition***



AFD

Table de transition

T	a	b
→**0**	1	0
1	2	1
←**2**	0	2

Expression régulière

- Les expressions régulières sont une importante notation pour spécifier formellement des modèles. Pour les définir correctement il nous faut faire l'effort d'apprendre un peu de vocabulaire :
  - On appelle alphabet un ensemble fini non vide Σ de symboles (lettres de 1 ou plusieurs caractères)
  - On appelle mot toute séquence finie d'éléments de Σ
  - On note ε le mot vide
  - On note Σ^* l'ensemble infini contenant tous les mots possibles sur Σ.

Expression régulière

- Elle permet de **spécifier** a priori un langage, alors que les automates permettent de tester **l'appartenance** d'un mot à un langage déjà spécifié.

- Exemples :
  - (a + b)* représente tous les mots sur {a, b}
  - a*(ba*)* représente le même langage
  - (a + b)*aab représente les mots se terminant par aab.

Expression régulière: Exemple

- Une ER décrivant les identificateurs en C pourrait être
  - Ident=(a|b|…|z|A|…|Z|_)(a|b|…|z|A|…|Z|_|0|…|9)*
  - ***Remarque:*** un peu illisible.
    - ➢ On utilise donc les définitions régulières.
  - Une définition régulière est une suite de définition de la forme
    - $d1 \rightarrow r1$ avec $r1 \in \Sigma^*$
    - $d2 \rightarrow r2$ avec $r2 \in (\Sigma \cup \{d1\})^*$
    - ***…***
    - $dn \rightarrow rn$ avec $rn \in (\Sigma \cup \{d1, d2, .. , dn\text{-}1\})^*$

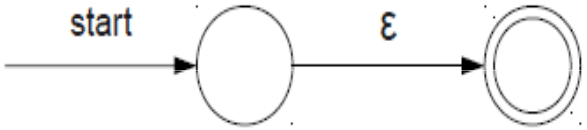
Expression régulière: Exemple

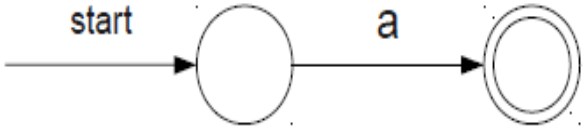
- Une ER décrivant les identificateurs en C pourrait être
  - Ident=(a|b|…|z|A|…|Z|_)(a|b|…|z|A|…|Z|_|0|…|9)*
  - ***Remarque:*** un peu illisible.
    - ➢ On utilise donc les définitions régulières.

IDENT (identificateurs) en C devient :

- **Lettre → A-Z|a-z**
- **Chiffre → 0-9**
- **Sep → _**
- **Ident → (lettre|sep)(lettre|sep|chiffre)***

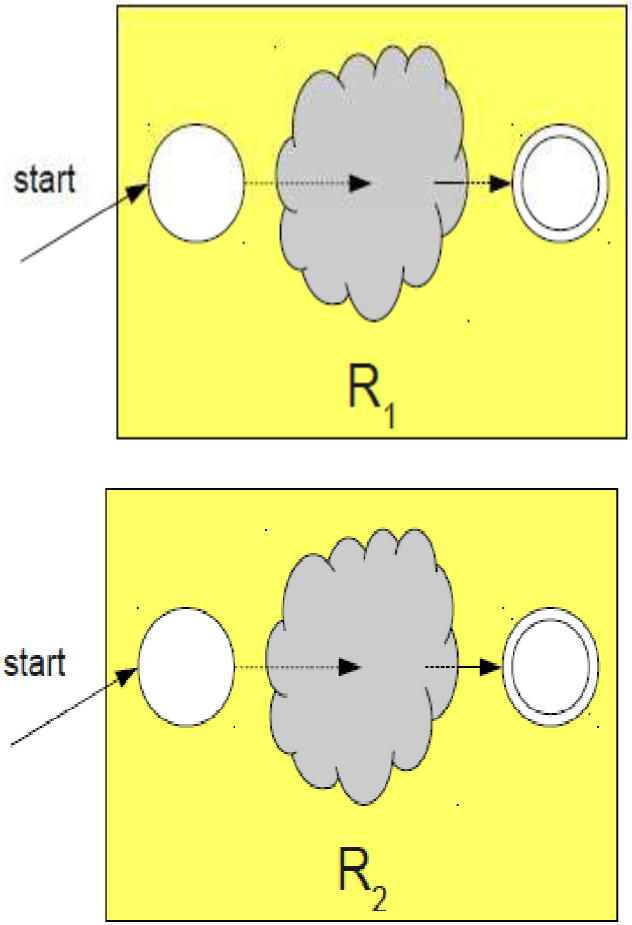
Construction d’un AFN à partir d’une ER

- Construction d’un AFN à partir d’une ER (construction de Thompson)
- pour ε : ***N(ε)*** 

- pour **a** : ***N(a)*** 

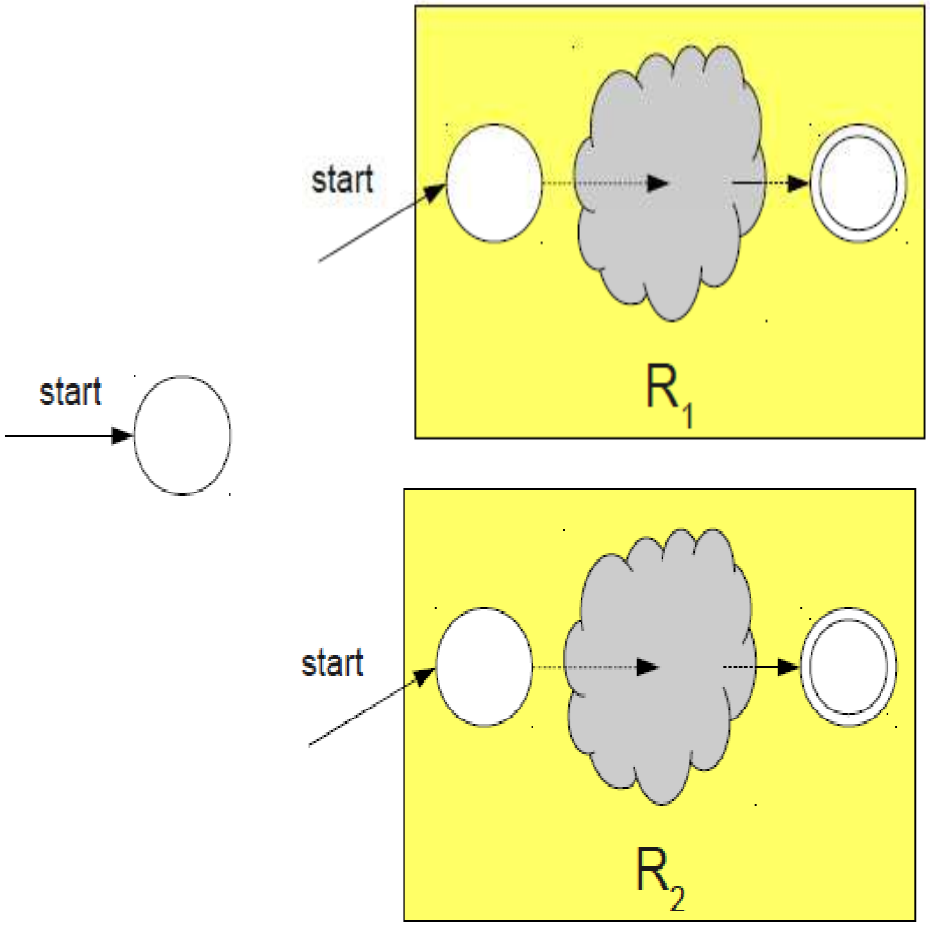
Construction d'un AFN à partir d'une ER

- pour $R_1 \mid R_2$: $N(R_1 \mid R_2)$



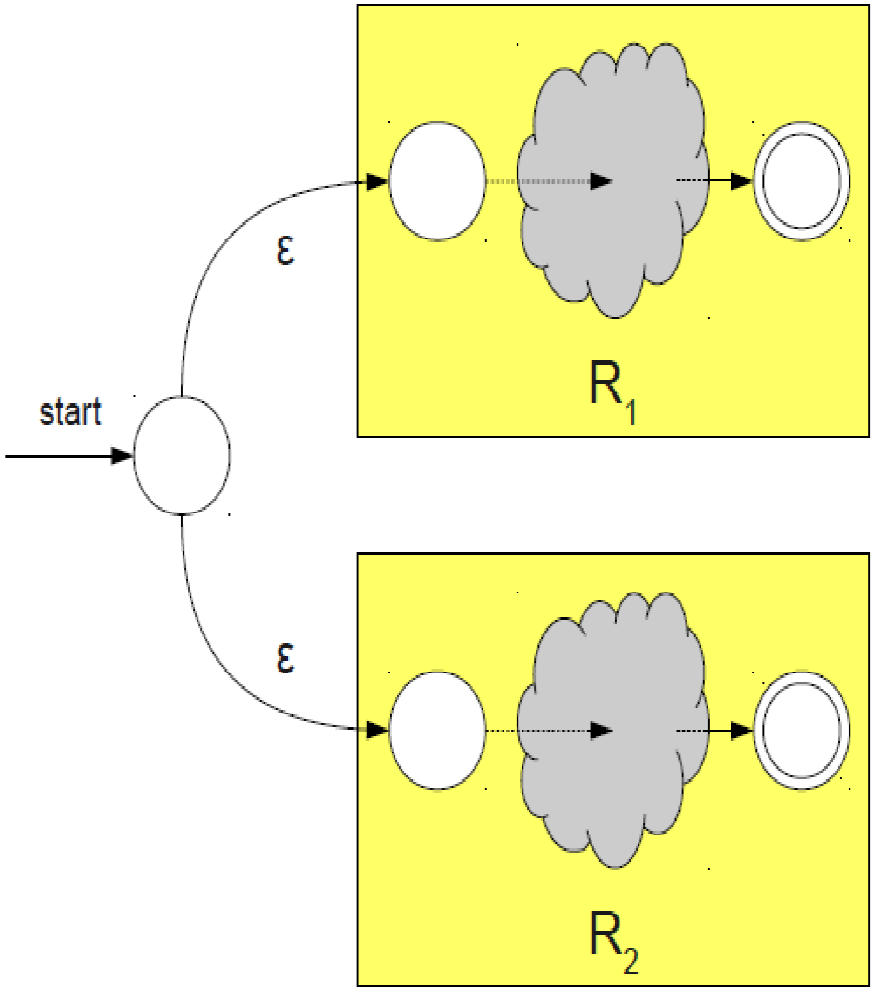
Construction d'un AFN à partir d'une ER

- pour $R_1 \mid R_2$: $N(R_1 \mid R_2)$



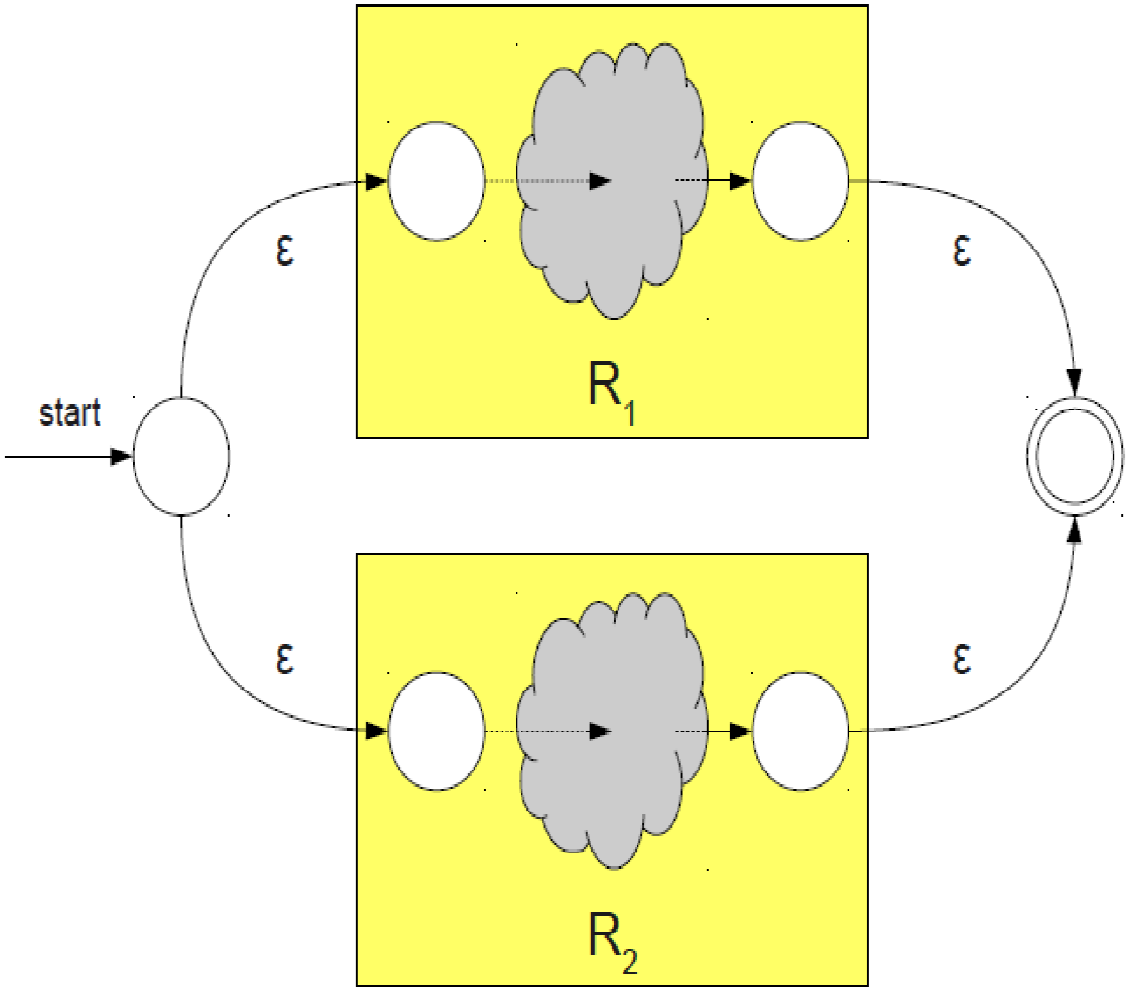
Construction d'un AFN à partir d'une ER

- pour $R_1 \mid R_2$: $N(R_1 \mid R_2)$



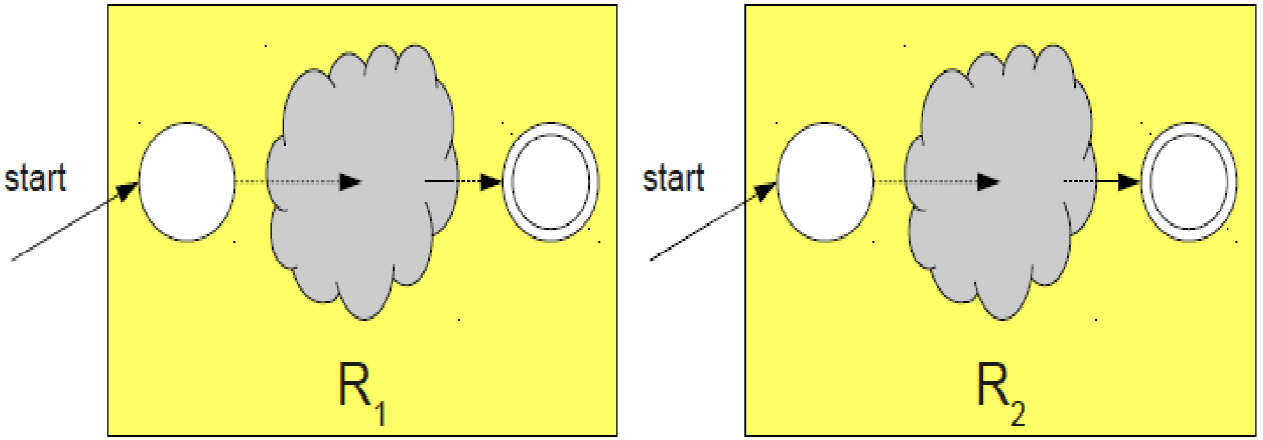
Construction d'un AFN à partir d'une ER

- pour $R_1 \mid R_2$: $N(R_1 \mid R_2)$



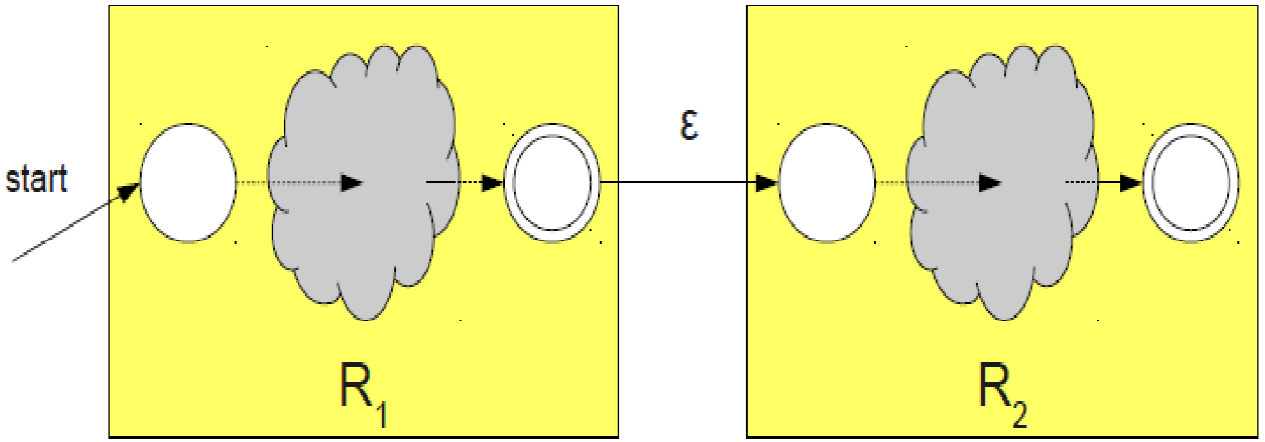
Construction d'un AFN à partir d'une ER

- pour $\mathbf{R_1R_2}$: $\boldsymbol{N(R_1R_2)}$



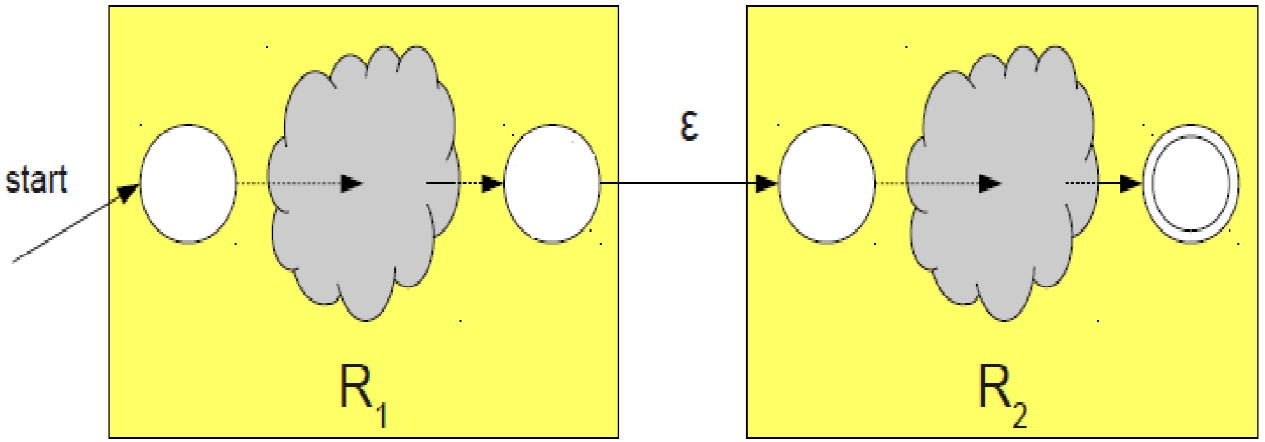
Construction d'un AFN à partir d'une ER

- pour $\mathbf{R_1R_2}$: $\boldsymbol{N(R_1R_2)}$



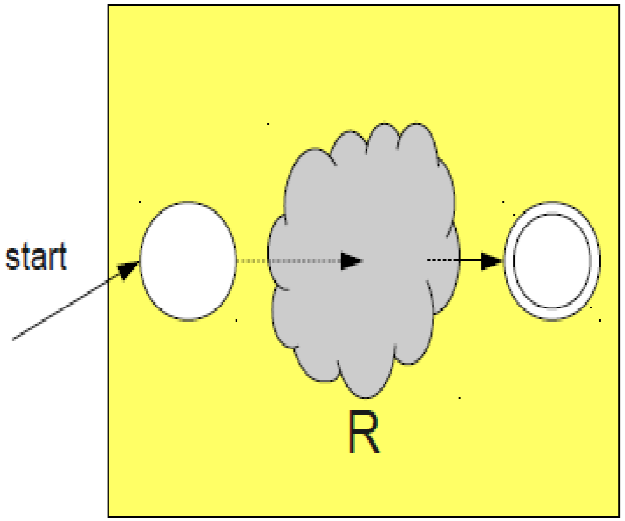
Construction d'un AFN à partir d'une ER

- pour $\mathbf{R_1R_2}$: $\boldsymbol{N(R_1R_2)}$



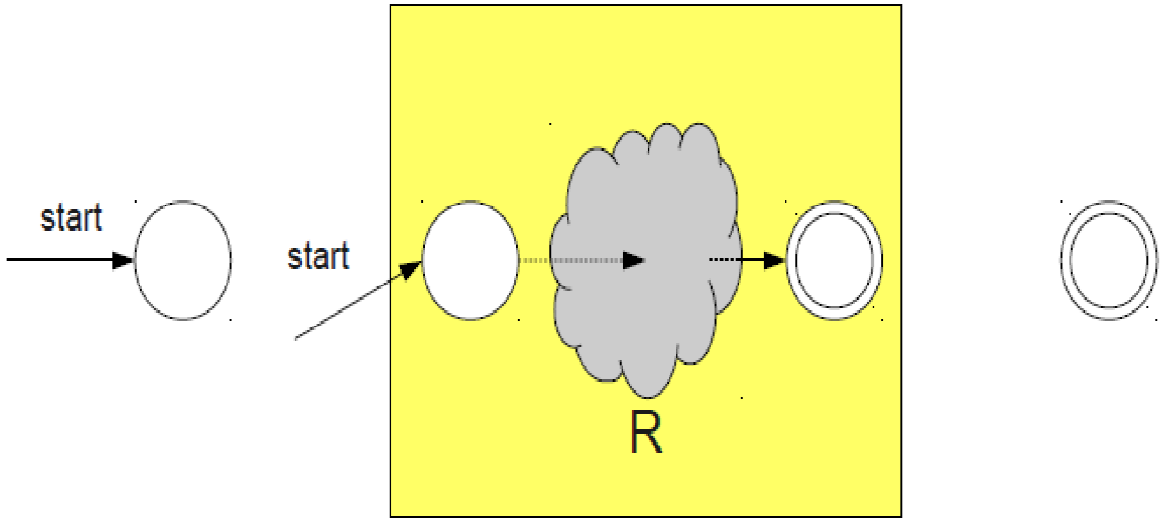
Construction d’un AFN à partir d’une ER

- pour **R*** **:** ***N(R*)***



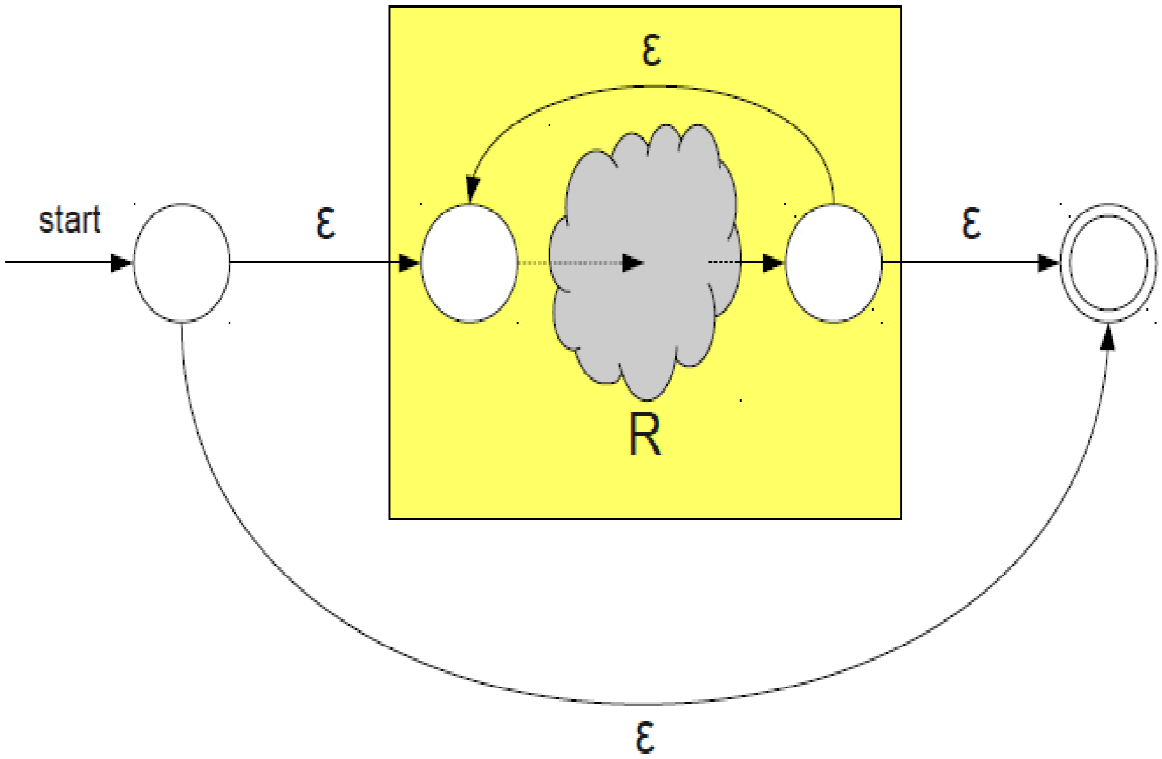
Construction d'un AFN à partir d'une ER

- pour **R*** **:** ***N(R*)***



Construction d'un AFN à partir d'une ER

- pour **R*** : ***N(R*)***



Construction d’un AFN à partir d’une ER

- **Exemple :** r = (a|b)*aab
  - N(r)

À faire ensemble

Transformation d’un AFN en AFD

- Un AFD permet de résoudre le problème de choix multiple lors de la recherche d’un chemin validant un mot donné dans un automate non déterministe.
- Dans un AFD, un seul chemin permet la reconnaissance d’un mot.
  - *Lemme1 : deux automates N1 et N2 sont dits équivalents si L(N1) = L(N2)*
  - *Lemme 2 : pour tout automate non déterministe, il existe un automate déterministe qui lui est* ***équivalent.***

Transformation d’un AFN en AFD

- Il existe des algorithmes permettant de rendre déterministe un automate non déterministe (c’est-à-dire de construire un AFD qui reconnait le même langage que l’AFN donné).
- L‘AFD obtenu comporte en général plus d’états que l’AFN, donc le programme le simulant occupe plus de mémoire. **Mais il est plus rapide**.

Transformation d'un AFN en AFD

- ***Un AFN ne contenant pas de ε-transitions***
  - L'algorithme à suivre est le suivant :
    - Partir de l'état initial ;
    - Rajouter dans la table de transition tout les nouveaux 'états' produits avec leurs transitions ;
    - Recommencer 2 jusqu'à ce qu'il n'y ait plus de nouvel 'état' ;
    - Tous les 'états' contenant au moins un état terminal deviennent terminaux ;
    - Renuméroter alors les états.

Transformation d'un AFN en AFD

- ***Un AFN ne contenant pas de ε-transitions***
  - Exemple :
    - ER : (a|b)*aa(a|b)*
    - AFN :

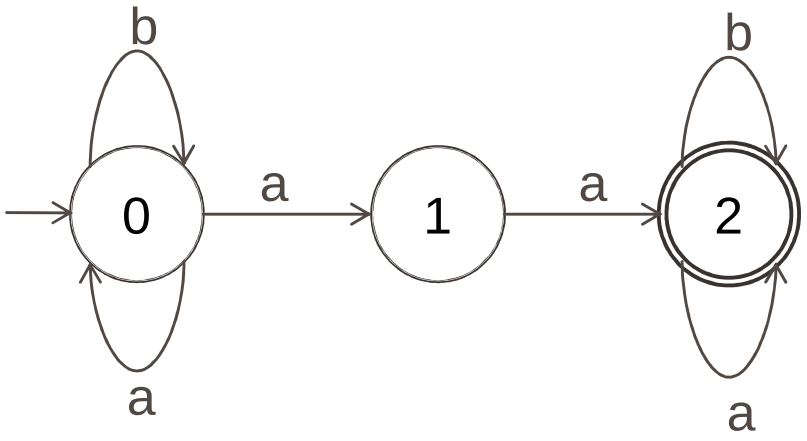
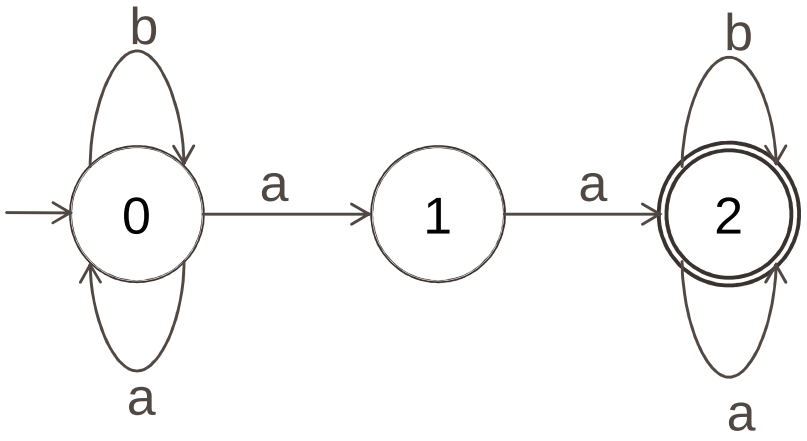


Table de transition de l'AFN

	a	b
→0	0,1	0
1	2	-
← 2	2	2

Transformation d'un AFN en AFD

- ***Un AFN ne contenant pas de ε-transitions***
  - Exemple :
    - ER : (a|b)*aa(a|b)*
    - Application de l'algorithme :



T	a	b
→ 0		

Transformation d'un AFN en AFD

- ***Un AFN ne contenant pas de ε-transitions***
  - Exemple :
    - ER : (a|b)*aa(a|b)*
    - Application de l'algorithme : dernière étape

	a	b
→ 0	1	0
1	2	0
← 2	2	3
← 3	2	3

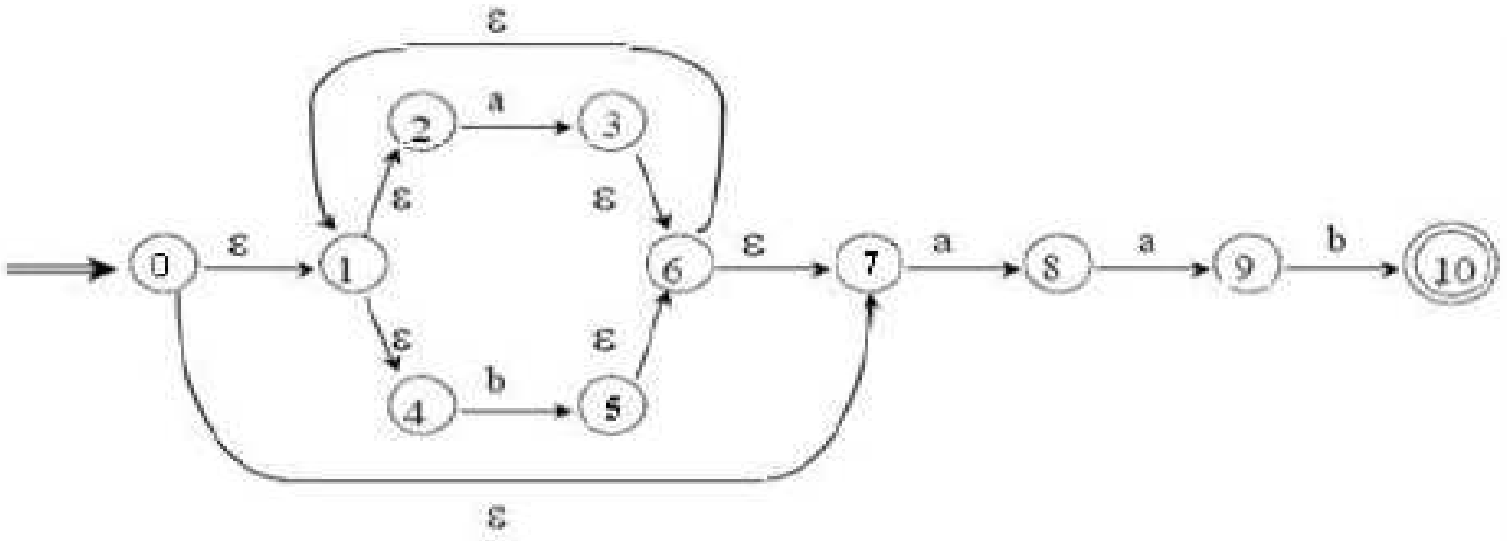
Transformation d'un AFN en AFD

- ***Cas général***
  - **Définition** :

    on appelle ε-fermeture d'un ensemble d'état E = {e_0, …, e_n} l'ensemble des états accessibles depuis un état e_i de E par des ε-transitions
  - **Algorithme** : Soit M un automate non déterministe défini par (E, Σ, e_0, F, T), l'algorithme de transformation en un AFD est le suivant:
    1. Construire l'état e'_0 tq e'_0= ε-fermeture (e_0) ;
    2. Rajouter dans la table de transition toutes les ε-fermetures des nouveaux 'états' produits avec leurs transitions ;
    3. Recommencer 2 jusqu'à ce qu' il n'y ait plus de nouvel 'état' ;
    4. Tous les 'états' contenant au moins un état terminal deviennent terminaux ;
    5. Renuméroter alors les états.

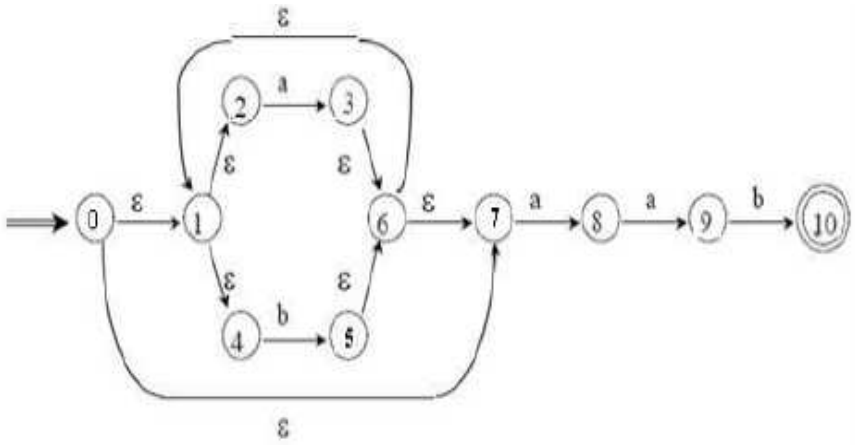
Transformation d’un AFN en AFD

- Exemple : r = (a|b)*aab
- N(r) par construction de Thompson



Transformation d'un AFN en AFD

- Exemple : r = (a|b)*aab
- ε-fermeture:



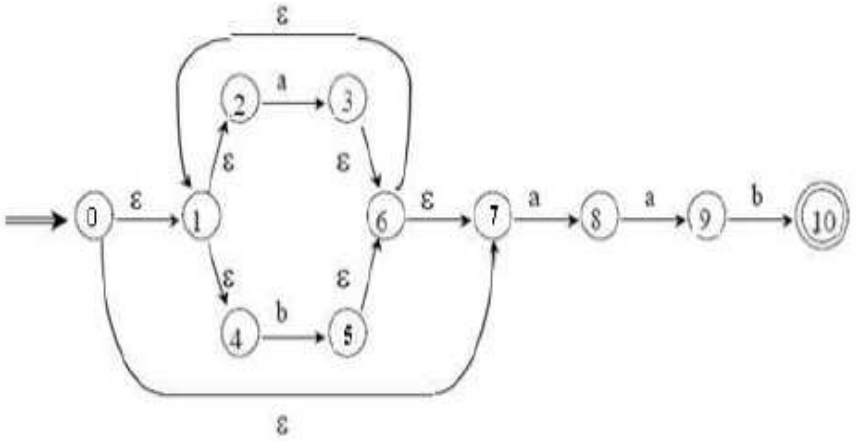
ε-fermeture (0) =
ε-fermeture (1) =
ε-fermeture (2) =
ε-fermeture (3) =
ε-fermeture (4) =
ε-fermeture (5) =
ε-fermeture (6) =
ε-fermeture (7) =
ε-fermeture (8) =
ε-fermeture (9) =
ε-fermeture (10) =

Transformation d'un AFN en AFD

- Exemple : r = (a|b)*aab
- Table de transition:

	a	b
→ A		

ε-fermeture (0) = {0, 1, 2, 4, 7} = A
ε-fermeture (1) = {1, 2, 4} = B
ε-fermeture (2) = {2} = C
ε-fermeture (3) = {3, 6, 1, 2, 4} = D
ε-fermeture (4) = {4}= E
ε-fermeture (5) = {5, 6, 7, 1, 2, 4}= F
ε-fermeture (6) = {6, 7, 1, 2, 4}= G
ε-fermeture (7) = {7}= H
ε-fermeture (8) = {8}= I
ε-fermeture (9) = {9}= J
ε-fermeture (10) = {10}= K



Transformation d’un AFN en AFD

- Exemple : r = (a|b)*aab
- AFD équivalent:

À faire ensemble

Minimiser un AFD

- Transformer le AFD en un autre AFD ayant le minimum nombre d'états et qui reconnait le même langage
  - Il faut éliminer les états dits « inaccessibles »
    - Par application de l'algorithme de marquage
  - En regroupant les états dits « congruents » (i.e. appartenant à la même classe d'équivalence) qui ont le même comportement sur les mêmes entrées

Minimiser un AFD : algorithme

1. Construire une partition initiale $\mathbf{P_i}$ de l'ensemble des états avec deux groupes : les états finaux F et les états non finaux E-F
2. Appliquer la procédure partitionner (en classes d'équivalence) à $\mathbf{P_i}$ pour construire une nouvelle partition $\mathbf{P_n}$.
3. Si $\mathbf{P_n} = \mathbf{P_i}$, soit $\mathbf{P_f} = \mathbf{P_i}$ et continuer à l'étape 4. Sinon répéter l'étape 2 avec $\mathbf{P_i} = \mathbf{P_n}$
4. Choisir un état dans chaque groupe de la partition P_f en tant que représentant de ce groupe. Les représentants seront les états de l'AFD réduit.
5. Si l'AFD réduit a un état mort **d**, alors supprimer **d** de l' AFD, supprimer aussi tout état **non accessible** depuis l'état de départ.

Minimiser un AFD : algorithme

- **Application**

T	a	b
→ **A**	B	C
B	B	D
C	B	C
D	B	E
← **E**	-	-

Pi = {E, ABCD}
Itération 1:
T(B, b) = D ➔ partitionner : Pi = {E, B, ACD}
Itération 2 :
T(D, b) = E ➔ partitionner Pi = {E, B, D, AC}
Pas d'autres partitions