ISEN CIR ^{3}

Théorie des langages

1. Automates finis

2021-2022

Rappel : Types de grammaires

T0
- **Pas de restrictions**
- Machine de Turing

T1
- **Langages contextuels**
- Automates linéairement bornés

T2
- **Langages algébriques**
- Automates à pile

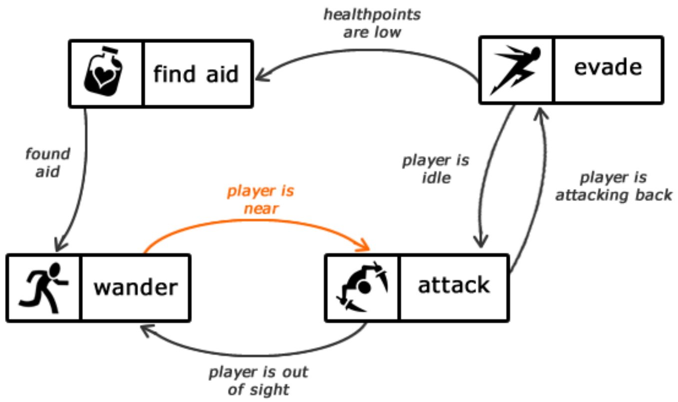
T3
- **Langages réguliers**
- Automates finis / Expressions régulières

- Les automates finis sont des machines abstraites qui effectuent des opérations sur une mémoire à taille limités. Les données en entrées par contre peuvent être supérieures (même très longues).

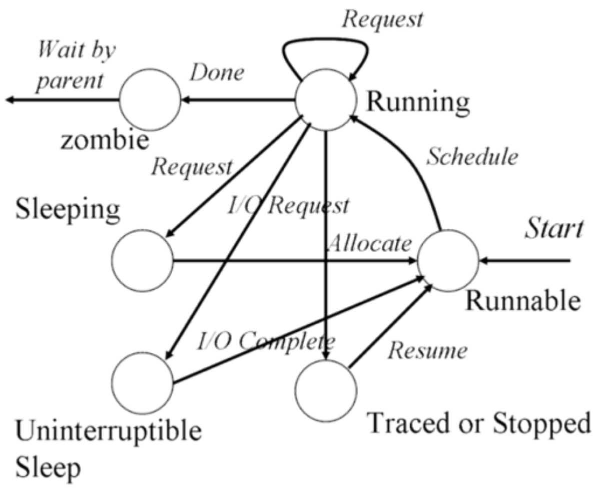
- Utilisés dans différents domaines (communication, électronique, commande numériques, jeux vidéos, …).

- Utilisée en théorie de langages pour reconnaitre des langages (réguliers).

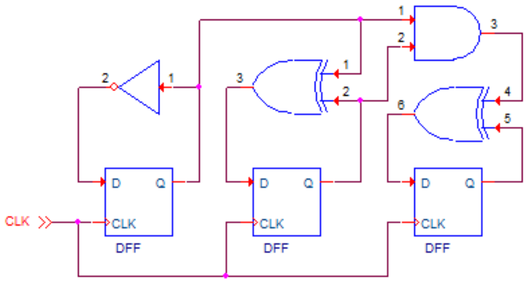
Applications



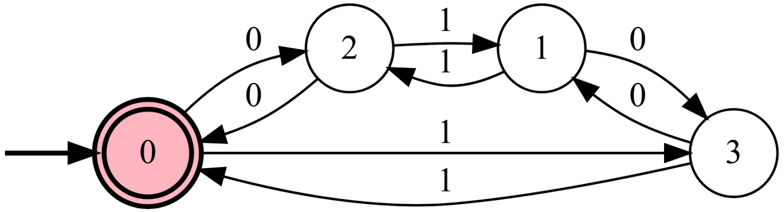
En jeux vidéo



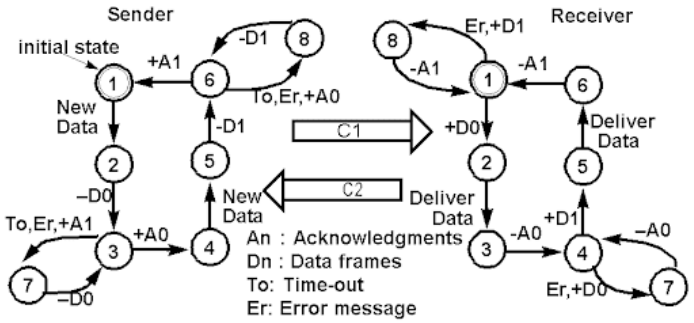
En systèmes d'exploitation



En électronique



En THL



En communication

- Un automate fini est un quintuplet **$M = (E, A, T, e_0, E_F)$:**

  E : Ensemble fini des états

  A : Alphabet du langage

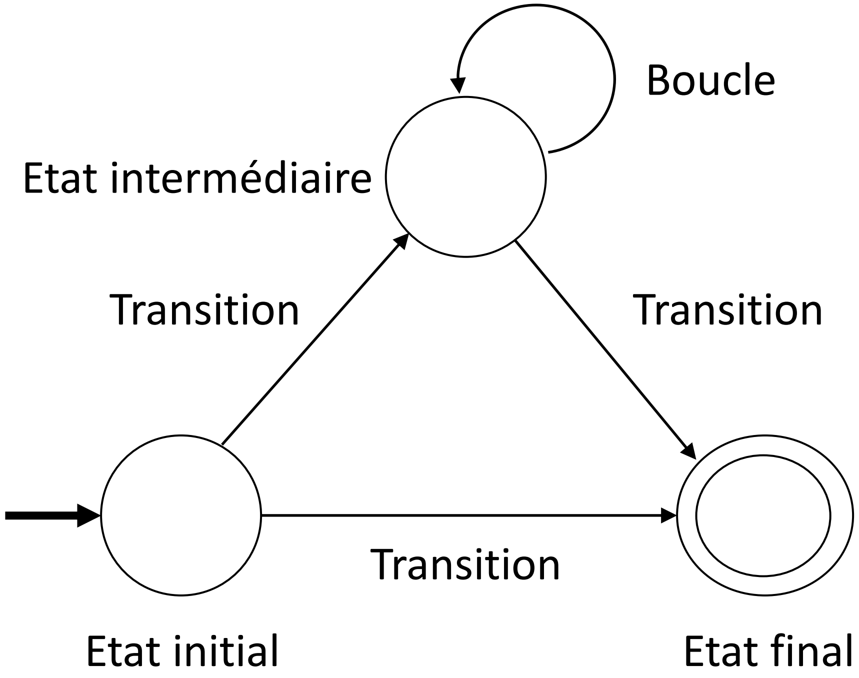
  $T : E \times A \rightarrow E$: Relation (ou fonction) de transition

  $e_0 \in E$: Etat initial (ou état de départ) Un seul

  $E_F \subset E$: Ensemble des états finaux Il peut y avoir plusieurs

Une **ε-transition** est une transition avec le mot vide ou transition inconditionnelle.

- Le diagramme de l'automate est un graphe orienté dont les sommets sont des états de l'automate, et les arcs (ou les arêtes) représentent les transitions (étiquetée par des symboles de l’alphabet ou ε).



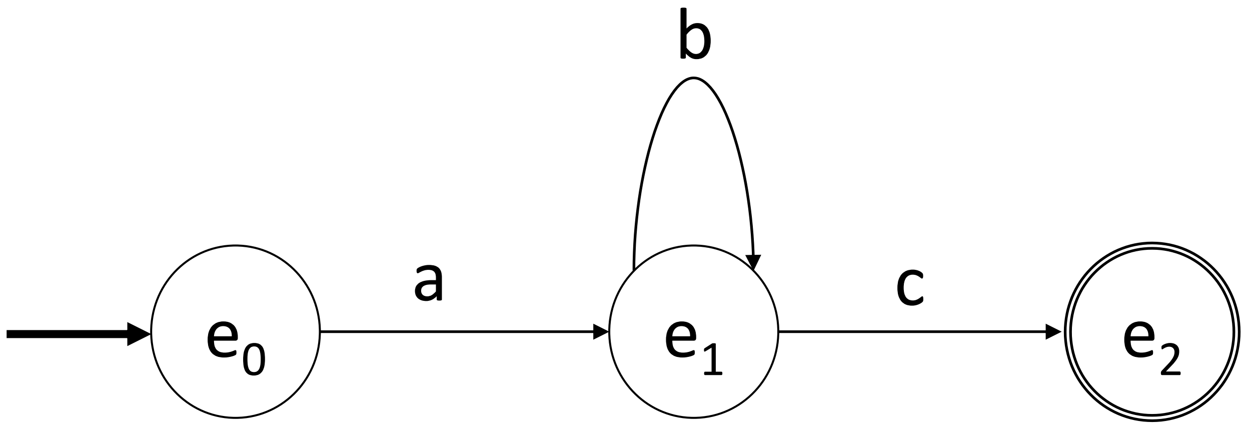
- $E = \{e_0, e_1, e_2\}$
- $A = \{a, b, c\}$
- $T = \{(e_0,a) \longmapsto e_1, (e_1,b) \longmapsto e_1, (e_1,c) \longmapsto e_2\}$
- Etat initial : e_0
- $F = \{e_2\}$

- Exemples de mots:

ac

abc

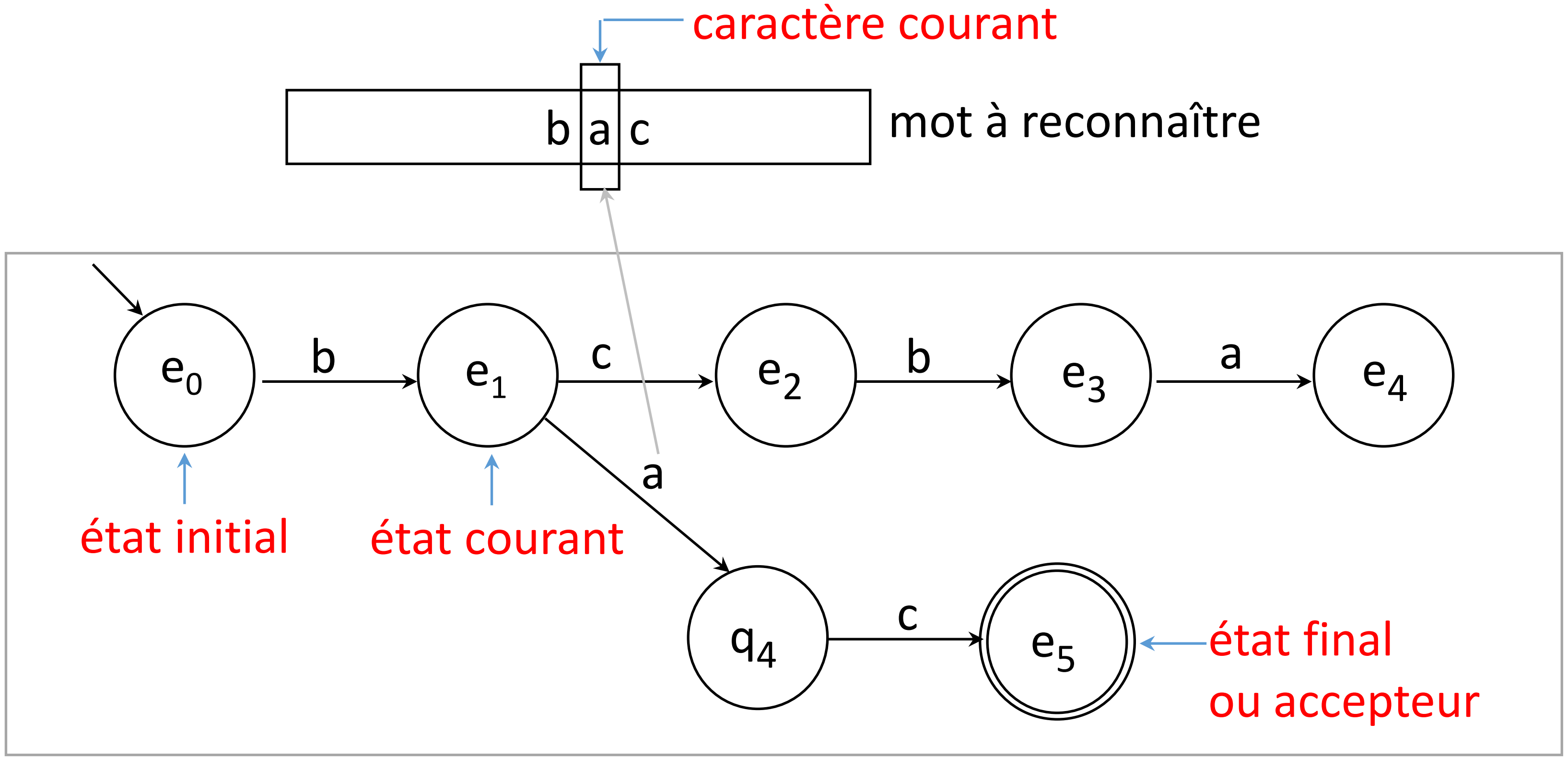
abbbbbbbbc



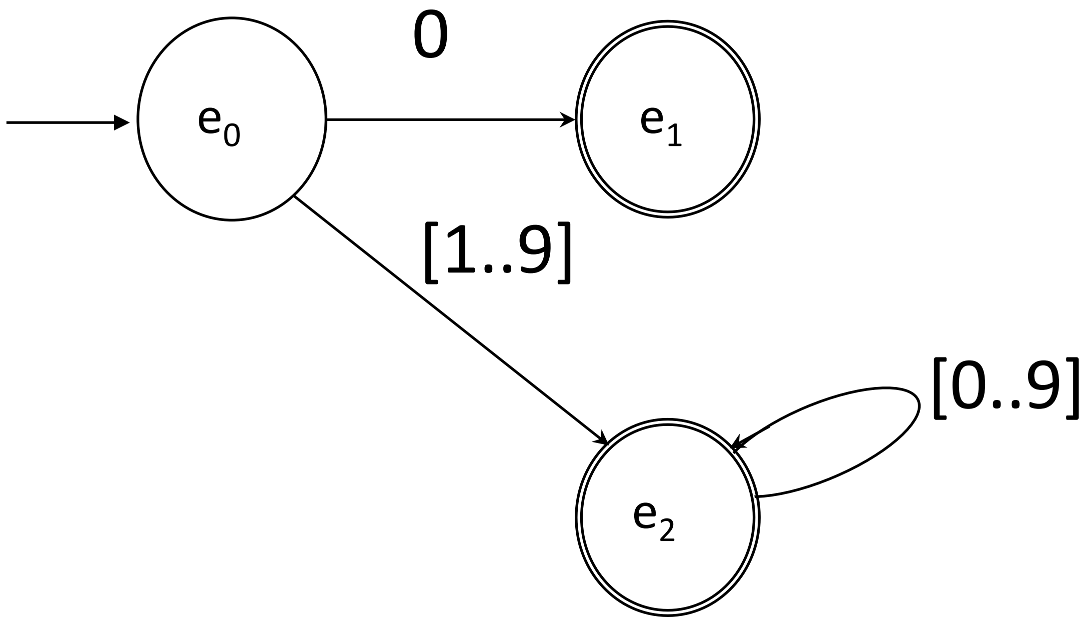
- Un automate accepte un mot **m** si on peut trouver un chemin :
    - Qui commence dans l’état initial
    - Qui suit des transitions
    - Qui finit dans un des états finaux
    - La concaténation des étiquettes des transitions donne **m**

- Les mots acceptés forment le langage de l’automate L(A)

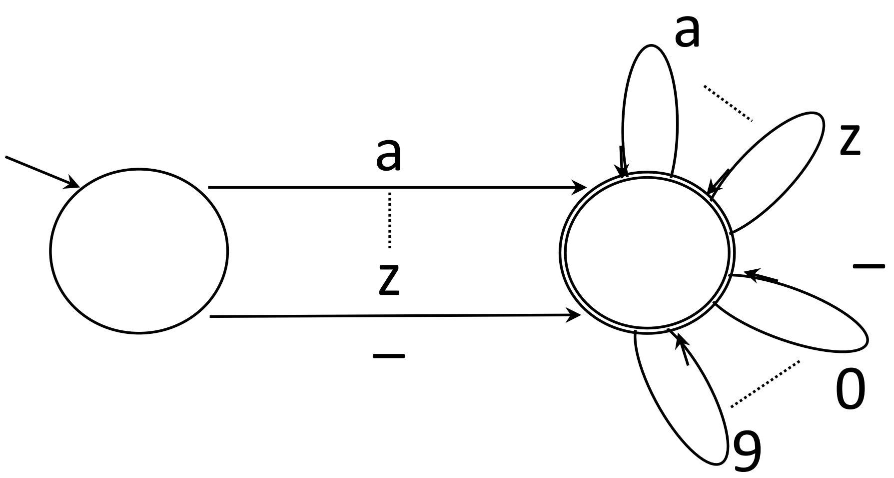
- Reconnaitre le mot **bac** avec l’automate suivant :



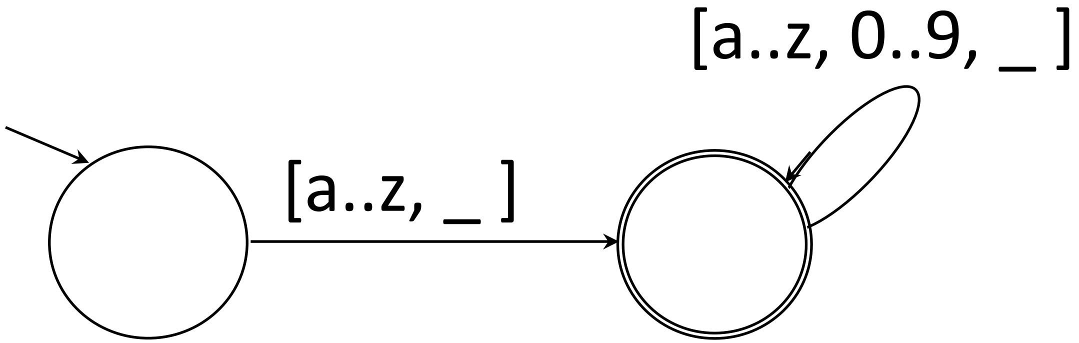
- Automate pour reconnaitre une constante naturelle
- Etats = $\{e_0, e_1, e_2\}$
- Etats finaux = $\{e_1, e_2\}$
- Etat initial = e_0



- Reconnaître les identificateurs



Ou plus simplement



Exemple 3

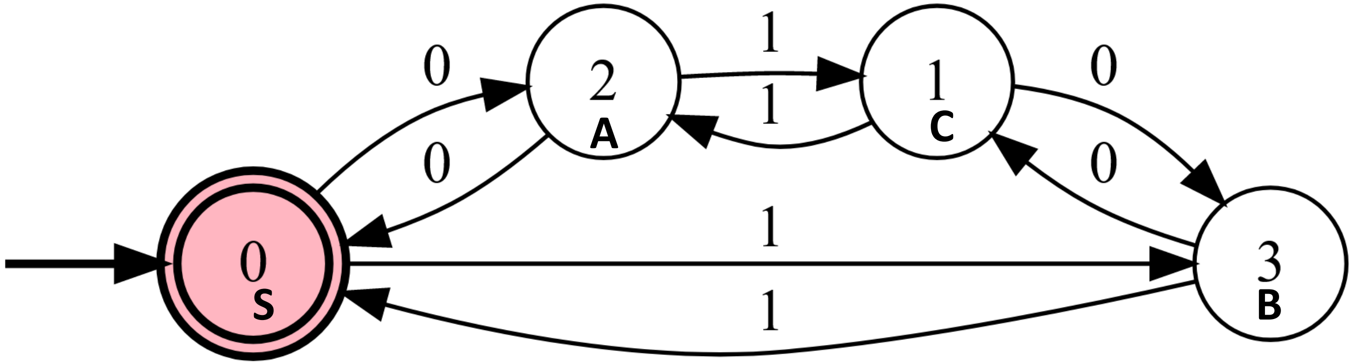
- Automate pour reconnaitre un binaire contenant un nombre pair de 0 et un nombre pair de 1

$S \rightarrow 0A \mid 1B \mid \varepsilon$

$A \rightarrow 0S \mid 1C$

$B \rightarrow 0C \mid 1S$

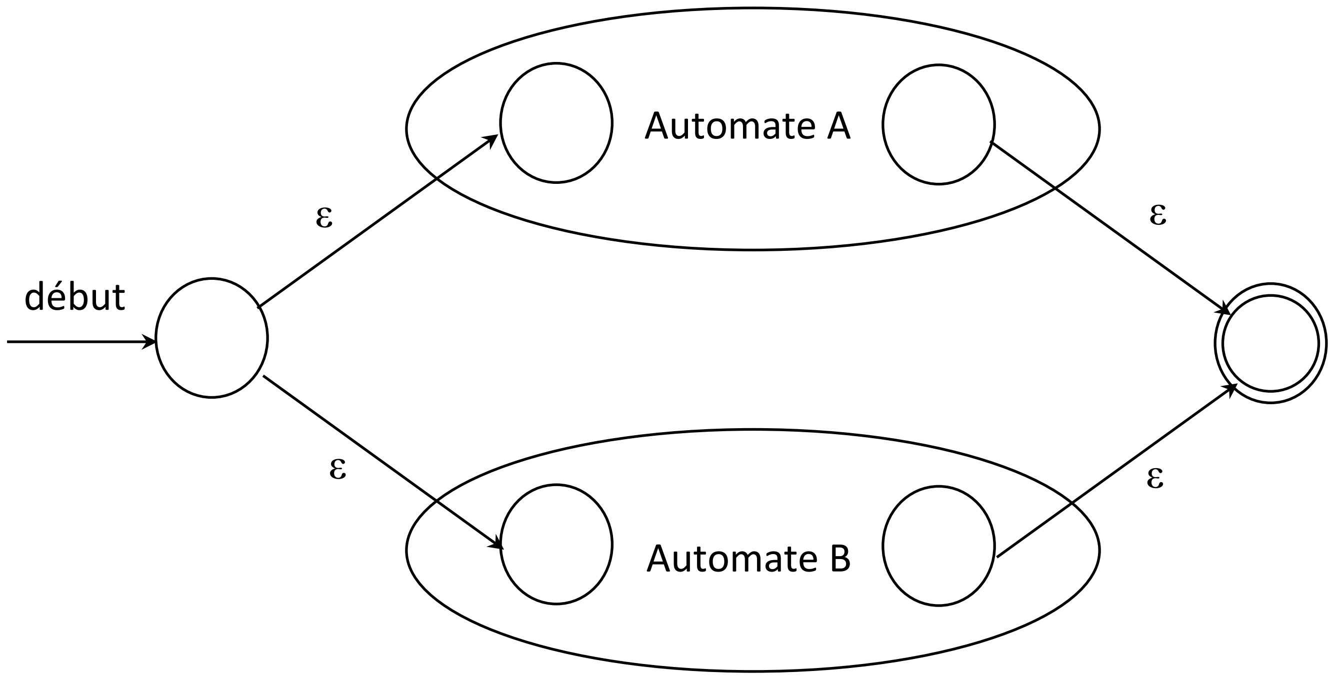
$C \rightarrow 0B \mid 1A$



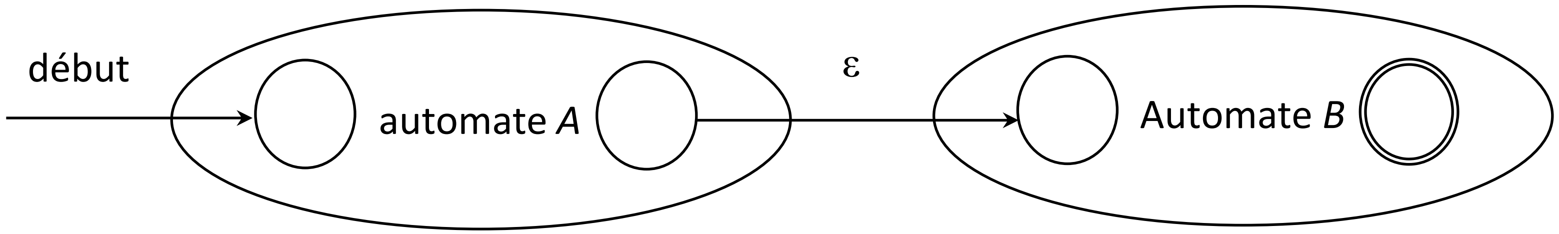
Opérations sur les automates

- Complémentation
- Produit (Concaténation)
- Union
- Intersection
- Fermeture (Répétition *)

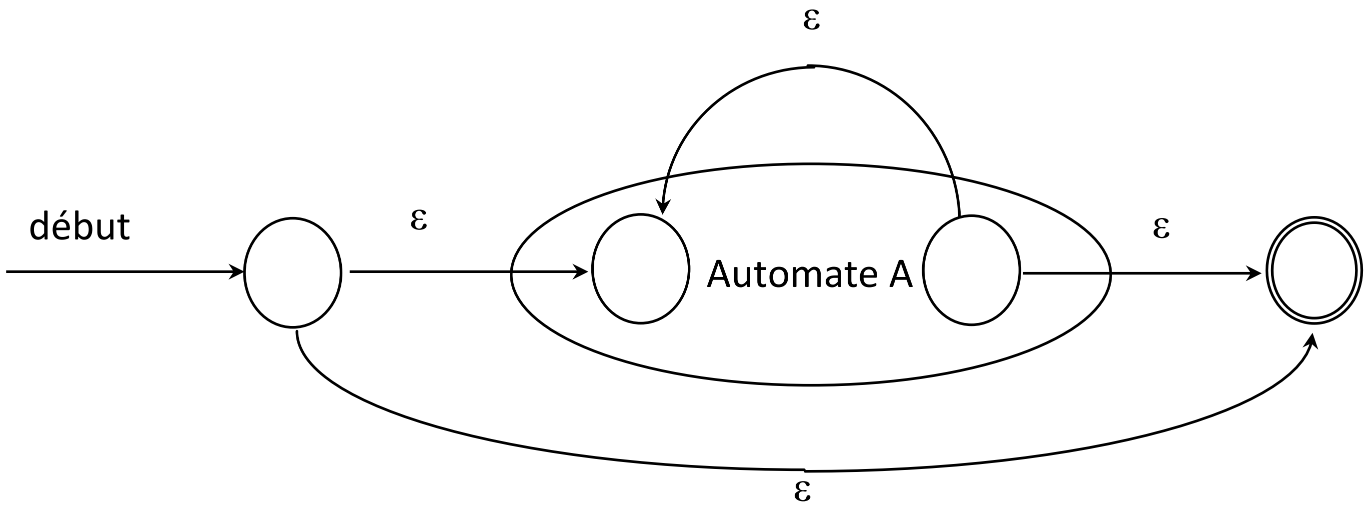
- Construction de l’automate d’union pour deux expressions régulières



- Construction de l’automate de concaténation pour deux langages réguliers



- Construction de l'automate de fermeture pour un langage régulier

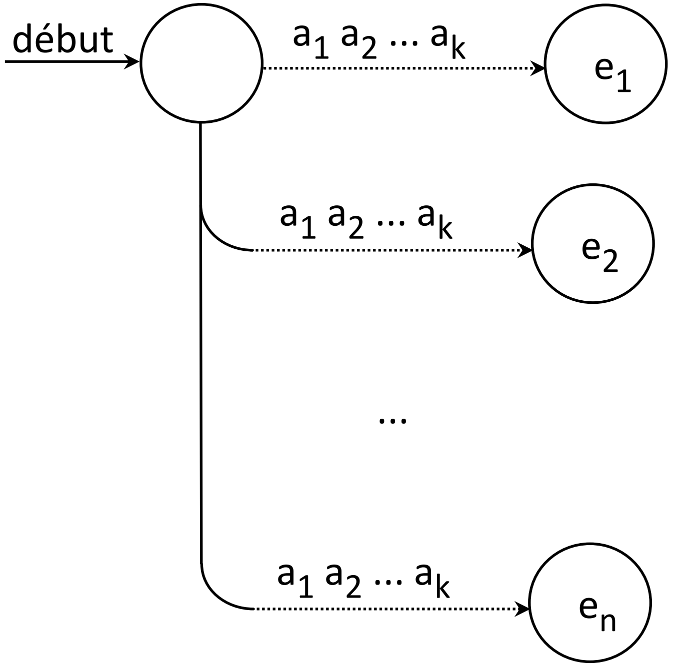


- Un automate fini est déterministe, intuitivement s'il ne peut se comporter que d'une façon **unique** pour un mot donné :

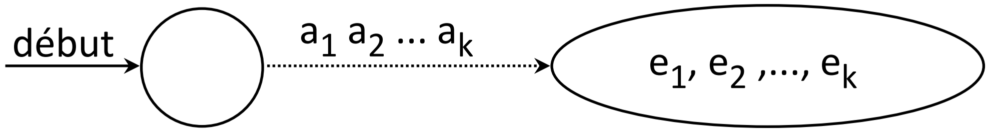
- Il ne contient pas de ε-transitions
- Etant donné un état de départ et un caractère lu, il y a au plus 1 état d'arrivée.
  On parle alors de **fonction de transition**.

- Il est en plus complet si, étant donné un état de départ et un caractère lu, il y a au moins un état d'arrivée

Idée de la déterminisation



Dans l'automate non déterministe:
Plusieurs chemins étiquetés pour un mot **a**

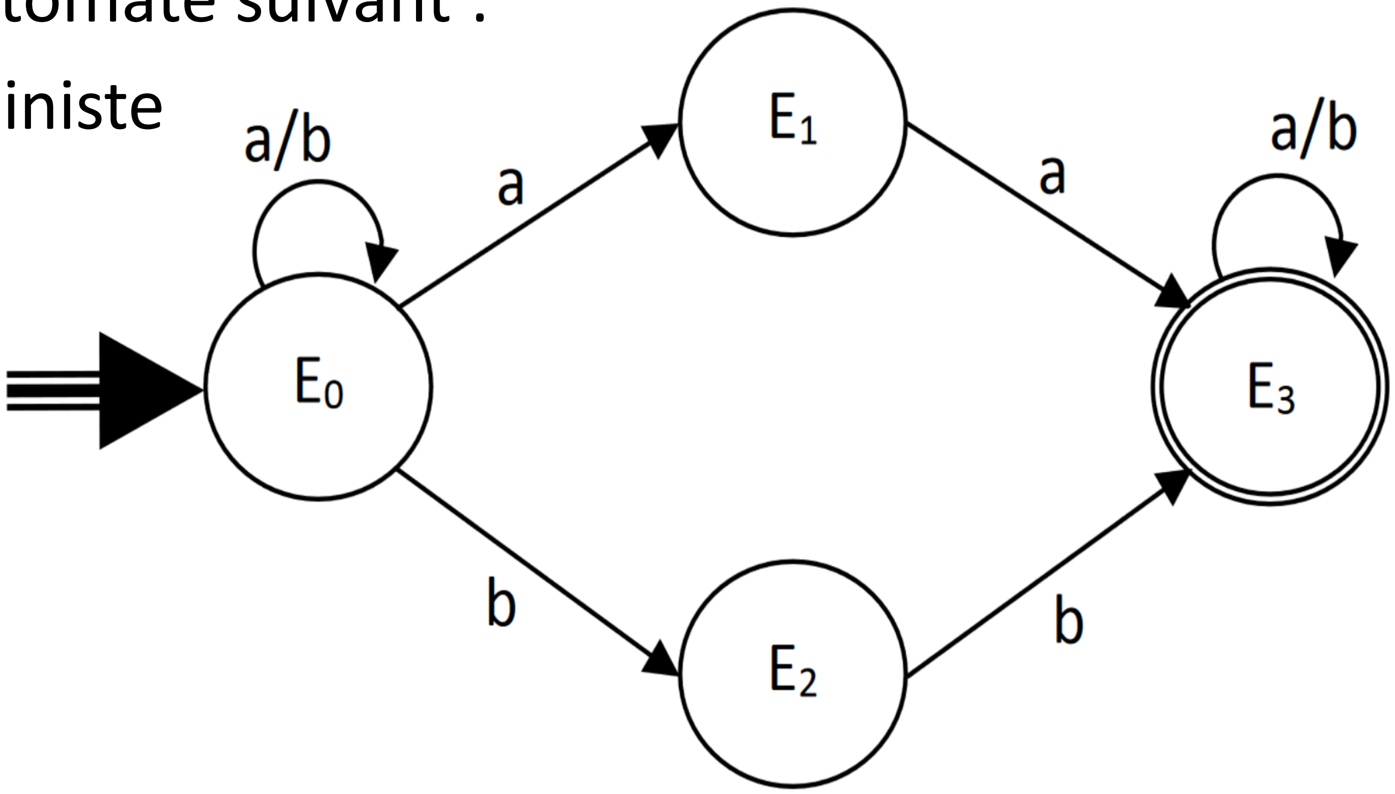


Dans l'automate déterministe:
Un seul chemin vers un état qui représente tous ceux où une copie peut être.

On rassemble les états destinations en un seul état

- On construit un automate A' :
- De même alphabet
- nouvel état = ensemble d'anciens états
- état initial = ε-successeurs(**$\{e_0\}$**) = états accessibles de **e_0** par des transitions vides
- état final = état contenant un ancien état final
- transition = ensemble des états où une copie **peut** arriver

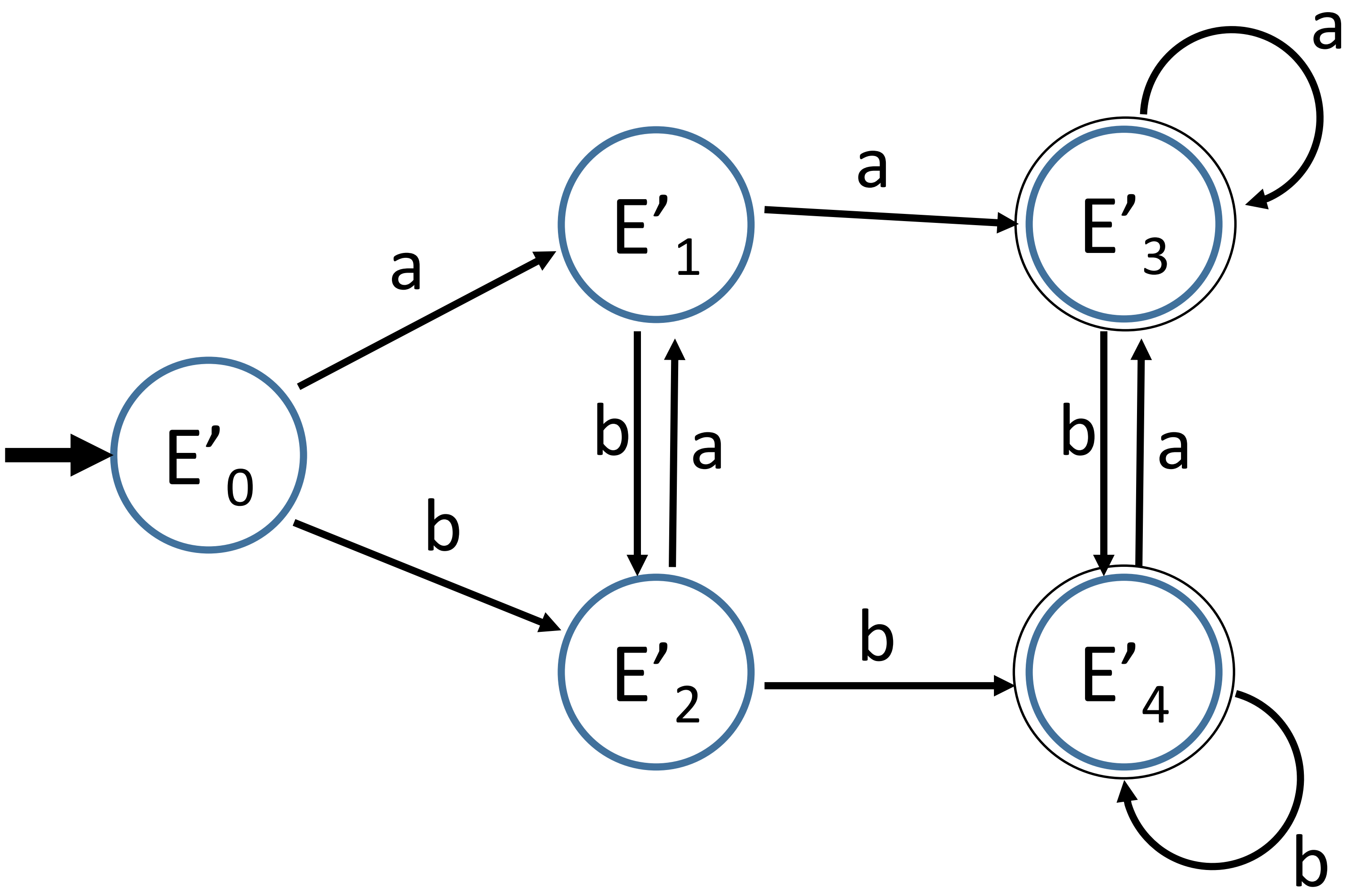
- On souhaite construire un automate pour reconnaitre sur l'alphabet **{a,b}** les séquences contenant une des chaines **aa** ou **bb**
- Naïvement on pense à l'automate suivant :
- Sauf qu'il n'est pas déterministe



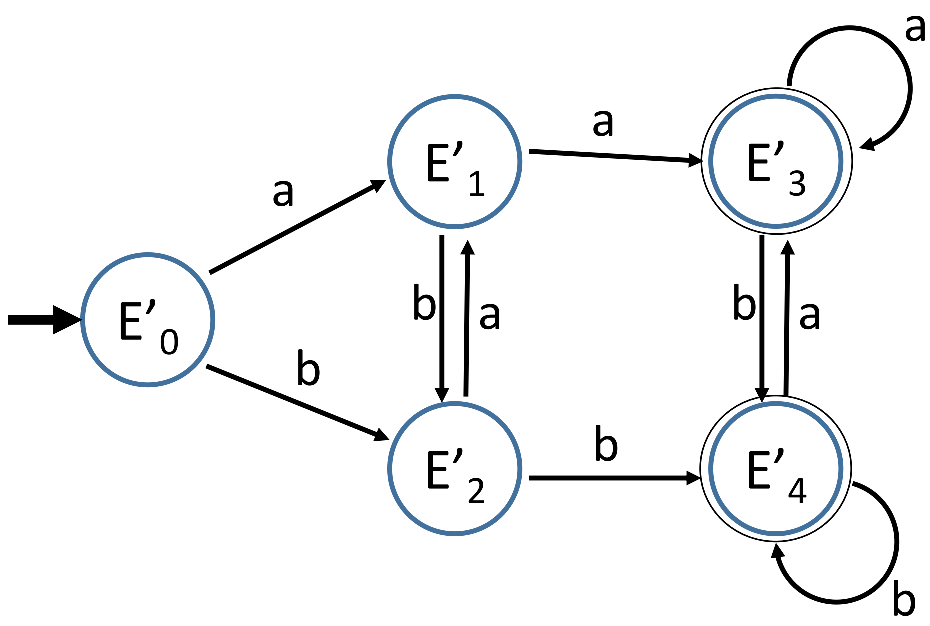
Etapes pour transformer l'automate

Nouveaux états		a	b
E_0	E'_0	E_0, E_1	E_0, E_2
E_0, E_1	E'_1	E_0, E_1, E_3	E_0, E_2
E_0, E_2	E'_2	E_0, E_1	E_0, E_2, E_3
E_0, E_1, E_3	E'_3	E_0, E_1, E_3	E_0, E_2, E_3
E_0, E_2, E_3	E'_4	E_0, E_1, E_3	E_0, E_2, E_3

Version déterministe



- Pour représenter un automate en mémoire, on utilise souvent une une matrice (Un tableau en 2D), appelée Table de transitions
- Les lignes représentent les états, les colonnes les lettres de l'alphabet.
- Les cellules contiennent les états destinations



	a	b
0	1	2
1	3	2
2	1	4
3	3	4
4	3	4

Pseudo-code générique de vérification d’un mot

```
état_courant = état_initial;
erreur = FAUX;
Répéter {
      c = Caractere_suivant(); // lecture d’un caractère
      état_courant = Transition[état_courant][c];
      Si (Erreur(état_courant)){
            erreur = VRAI;
      }
}
Tant_que (!erreur et c!=FIN);

Retourner !erreur et c==FIN et Est_Final(état_courant)
```

Outil en ligne

http://ivanzuzak.info/noam/webapps/fsm_simulator/

Construire l’automate de l’expression régulière
(00+11+(01+10)(00+11)*(01+10))*

Vérifier le mot 001010011101