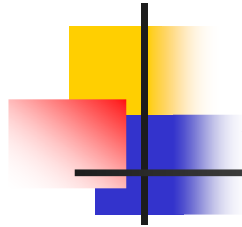




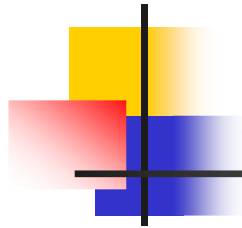
ANALYSE LEXICALE

*ESI – École nationale Supérieure en
Informatique*



Analyse lexicale en bref I

- Lire le texte source d'entrée caractère par caractère
- Identifier les entités lexicales (mots-clés, identificateurs, opérateurs, ...).
- Ces entités lexicales seront codifiées pour une utilisation ultérieure efficace.



Analyse lexicale en bref II

- Les caractères superflus (blancs, tabulations, commentaires) seront ignorés.
- En cas d'erreur, l'analyseur lexical devra situer l'erreur et donner un message significatif à l'utilisateur (e.g. "mot trop long").



Analyse lexicale en bref III

- L'outil formel sous-jacent à l'analyse lexicale est l'automate d'états finis déterministe (désigné par l'abréviation AFD)
- L'obtention directe de l'AFD peut être parfois une tâche ardue.



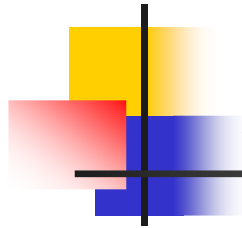
Analyse lexicale en bref IV

- L'outil formel sous-jacent à l'analyse lexicale est l'automate d'états finis déterministe (Abréviation AFD)
- L'obtention directe de l'AFD peut être parfois une tâche ardue.
- C'est pour cela que l'on donne souvent en premier lieu l'automate d'états finis non déterministe (désigné par AFN).



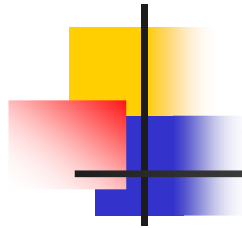
Analyse lexicale en bref V

- L'automate AFN est "facile" à obtenir et la transformation de l'AFN en AFD est automatique (par algorithme).
- Pour des spécifications lexicales nous pouvons avoir plusieurs AFD corrects.
- La minimisation du nombre d'états de l'AFD est une étape fortement recommandée.



Analyse lexicale en bref VI

- Les spécifications lexicales peuvent être décrites par des expressions régulières (ou expressions rationnelles).
- Les expressions régulières permettent de définir d'une manière concise et compacte un modèle de chaînes de caractères.



Analyse lexicale en bref VII

- Les expressions régulières sont par exemple utilisées par les générateurs d'analyseurs lexicaux comme l'outil LEX.
- Les expressions régulières doivent être transformées en automates d'états finis
- (le théorème de Kleene prouve qu'un langage est décrit par une expression régulière ssi il est reconnu par un AF).



Expressions Régulières

Chaînes et langages

- Un alphabet (dénnoté généralement par le symbole Σ) est un ensemble fini de symboles.
- Une chaîne est séquence finie de symboles (le terme mot est également utilisé pour désigner une chaîne)



Expressions Régulières

Chaînes et langages

- La longueur d'une chaîne ω est notée $|\omega|$ est désigne le nombre de symbole de la chaîne ω .
- La chaîne vide est notée ε et est de longueur 0.



Expressions Régulières

Concaténation

- Soient ϖ et ψ deux chaînes.
- La **concaténation** de ϖ et de ψ notée $\varpi.\psi$ ou $\varpi\psi$ est formée des symboles de ϖ suivis des symboles de ψ .
 - $abc.de = abcde$
 - $\varepsilon.\varpi = \varpi.\varepsilon = \varpi$
 - $\varpi^1 = \varpi$ et $\varpi^2 = \varpi.\varpi$
 - ϖ^i : concaténation i fois de la chaîne ϖ
 - $\varpi^0 = \varepsilon$



Expressions Régulières

Langage

- On utilise le terme langage pour désigner un ensemble de chaînes formées à partir d'un alphabet.
- La concaténation de deux langages L et M notée L.M ou LM est définie comme suit : $L.M = \{\varpi\psi \mid \varpi \in L \text{ et } \psi \in M\}$.



Expressions Régulières

Langage

- L'union de 2 langages L et M est définie par $L \cup M = \{\varpi \mid \varpi \in L \text{ ou } \varpi \in M\}$
- Les opérations de fermeture et de fermeture positive d'un langage L sont notées L^* et L^+ et sont définies comme suit :
 - $L^* = L^i$
 - $L^+ = L.L^i$



Expressions Régulières

Définition de Kleene

- Une expression régulière sur un alphabet Σ est construite récursivement à partir de :
 - Une lettre de l'alphabet a désigne le langage $\{a\}$.
 - Epsilon: ε désigne le langage $\{\varepsilon\}$.



Expressions Régulières

Définition de Kleene (suite)

- Si M et N sont 2 expressions régulières décrivant les langages L_M et L_N alors :
- $M.N$: Concaténation, désigne le langage $L_M \cdot L_N$
- $M \mid N$ (ou $M+N$) : Alternative, désigne le langage $L_M \cup L_N$
- M^* : Itération ou Fermeture de Kleene, désigne le langage L_M^*



Expressions Régulières

Remarques

- Si R est une expression régulière alors (R) désigne la même expression.
- Les parenthèses peuvent être utilisées pour définir des éléments composés d'une expression régulière (e.g. $(a|b)^*ab$).



Expressions Régulières

Exemples

- $a^* = \{a^i\}$ i.e. l'ensemble des chaînes de zéro ou plusieurs a.
- L'expression régulière **a^+ ($a.a^*$)** désigne l'ensemble des chaînes de 0 ou plusieurs a.
- $(a|b)^* = \{a,b\}^i$
- `identificateur` = $\{\text{lettre}\} . (\{\text{lettre}\} \mid \{\text{chiffre}\})^*$



Expressions Régulières

Convention d'écriture

- L'opérateur de concaténation $.$ est souvent omis dans l'écriture des expressions régulières.
- Le fait de faire suivre deux expressions sans séparateur signifie leur concaténation.



Automate d'états Finis

Formalisme – Modèle de Moore

- Un automate fini M est un quintuple
 - $(Q, \Sigma, \delta, q_0, F)$ où :
- Σ est un alphabet;
- Q est un ensemble fini d'états;
- $\delta: Q \times \Sigma \rightarrow Q$ est la "fonction" de transition;
- q_0 est l'état initial;
- F est un ensemble d'états finaux.



Automate d'états Finis

Langage reconnu

- Le langage $L(M)$ reconnu par l'automate M est l'ensemble $\{ w \mid \delta(q_0, w) \in F \}$ des mots permettant d'atteindre un état final à partir de l'état initial de l'automate



Automate d'états Finis Déterministe

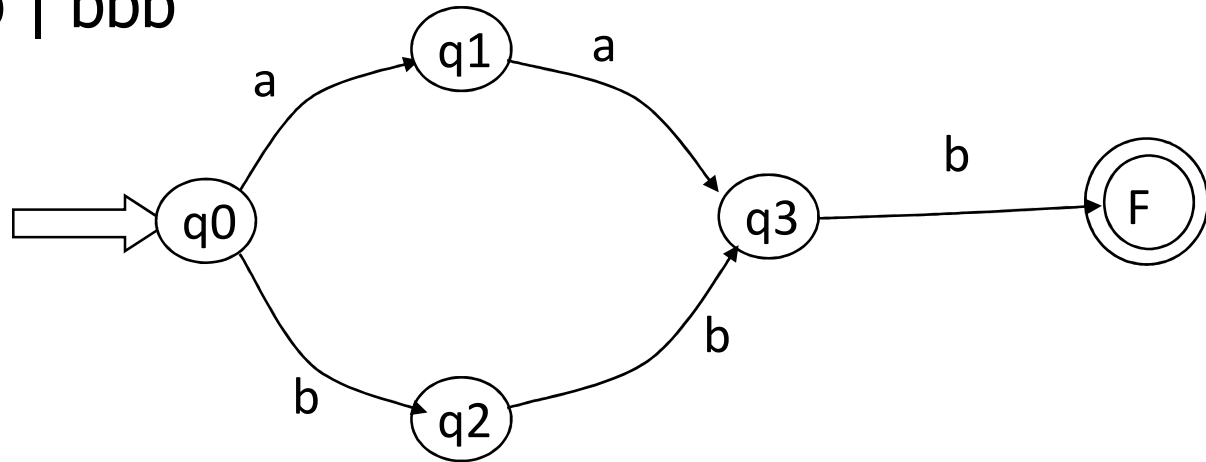
AFD

- Un automate d'états finis est déterministe si :
 - Il n'existe pas de ε -transition
 - Pour chaque état e et un symbole d'entrée a , alors il existe au plus un arc étiqueté a sortant de l'état e .

Automate d'états Finis Déterministe

AFD

- Automate déterministe correspondant à l'expression régulière $aab \mid bbb$



- $\Sigma = \{a, b\}$
- $Q = \{q_0, q_1, q_2, q_3, F\}$
- $\delta = \{(q_0, a) \rightarrow q_1, (q_0, b) \rightarrow q_2, (q_1, a) \rightarrow q_3, (q_2, b) \rightarrow q_3, (q_3, b) \rightarrow F\}$
- État initial q_0 et un seul état final F .



Automate d'états Finis Déterministe

AFD

- Il est facile de simuler un automate d'états finis déterministe par un programme simple.
- On désire analyser une chaîne d'entrée S terminée par un caractère spécial '#' .
- On dispose des fonctions suivantes :
 - **Transiter (e, c)** : donne l'état de l'automate vers lequel il existe une transition depuis l'état e sur le caractère d'entrée c.
 - **CarSuiv ()** : retourne le prochain caractère à analyser de la chaîne S.



Automate d'états Finis Déterministe

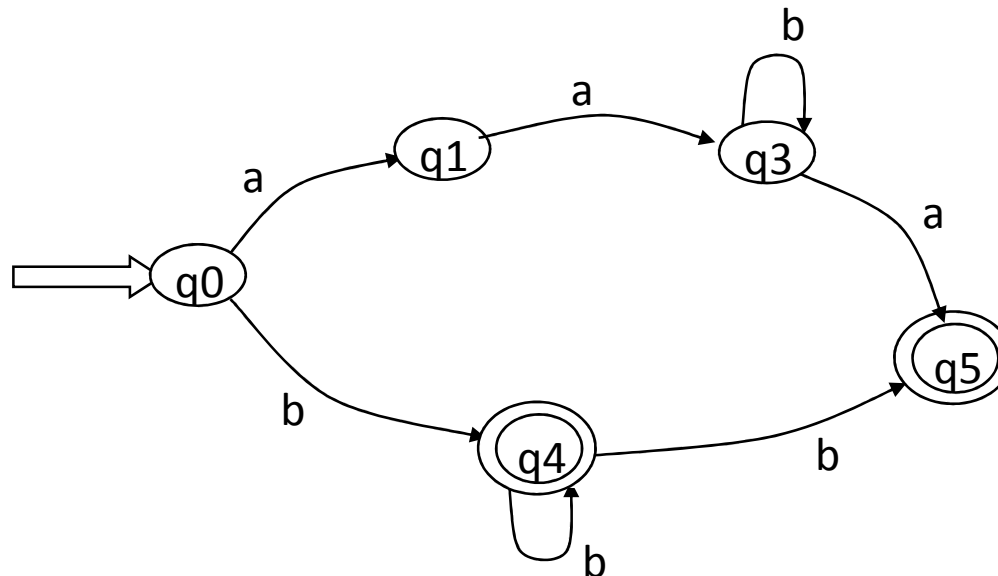
Algorithme

- $e := e_0 ;$
- $c := \text{CarSuiv} ;$
- **Tant que** $(c \neq \text{'\#'} \text{ et } e \neq \emptyset)$
- **Faire** $e := \text{Transiter}(e, c);$
- $c := \text{CarSuiv}();$
- **Fait;**
- **Si** $e \in F$ **Alors** "Chaîne acceptée"
- **Sinon** "Chaîne refusée"
- **Fsi**

Automate d'états Finis Non Déterministe

Définition

- Un automate d'états finis est non déterministe si :
 - Il peut exister des ε -transitions
 - Un symbole d'entrée a peut étiqueter plusieurs arcs sortants d'un même état e .





Automate d'états Finis Non Déterministe

Remarques

- Les automates d'états finis non déterministes sont plus faciles à obtenir que les automates déterministes.
- Mais il est "difficile" de simuler un automate d'états finis non déterministe par un programme simple.



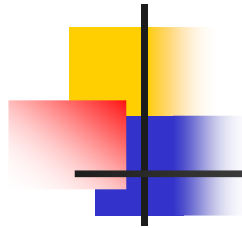
Représentation d'un automate

- La fonction δ de domaine fini $Q \times \Sigma$ peut être représentée par une matrice de dimension 2
- Les éléments sont les états (pour un automate déterministe) ou ensembles d'états (pour un automate non-déterministe) définissant δ .



Représentation d'un automate

- Représentation pleine (tableau de tableaux)
- ou creuse (tableau de listes) selon la situation.
- La première est bien sûr plus efficace en temps mais plus gourmande en espace



Représentation d'un automate

- Exemple :

	a	b
$\{q_0\}$	$\{q_1\}$	$\{q_2\}$
$\{q_1\}$	$\{q_3\}$	-
$\{q_2\}$	-	$\{q_2, q_4\}$
$\{q_3\}$	$\{q_4\}$	$\{q_3\}$
$\{q_4\}$	-	-



Transformation d'un AFN en AFD

Algorithme

- e_0 : état initial de l'AFN ;
- T : un ensemble d'états de l'AFN ;
- e : un état de l'AFN ;



Transformation d'un AFN en AFD

Algorithme

- **ϵ -fermeture(e)** : ensemble des états de l'AFN accessibles depuis l'état e de l'AFN par ϵ -transitions (état e inclus).
- **ϵ -fermeture(T)** : ensemble des états de l'AFN accessibles depuis un état e appartenant à T par des ϵ -transitions (l'ensemble T inclus).
- **Transiter (T, a)** : ensemble des états de l'AFN vers lesquels il existe une transition dans l'AFN sur le symbole a à partir d'un état e appartenant à T .



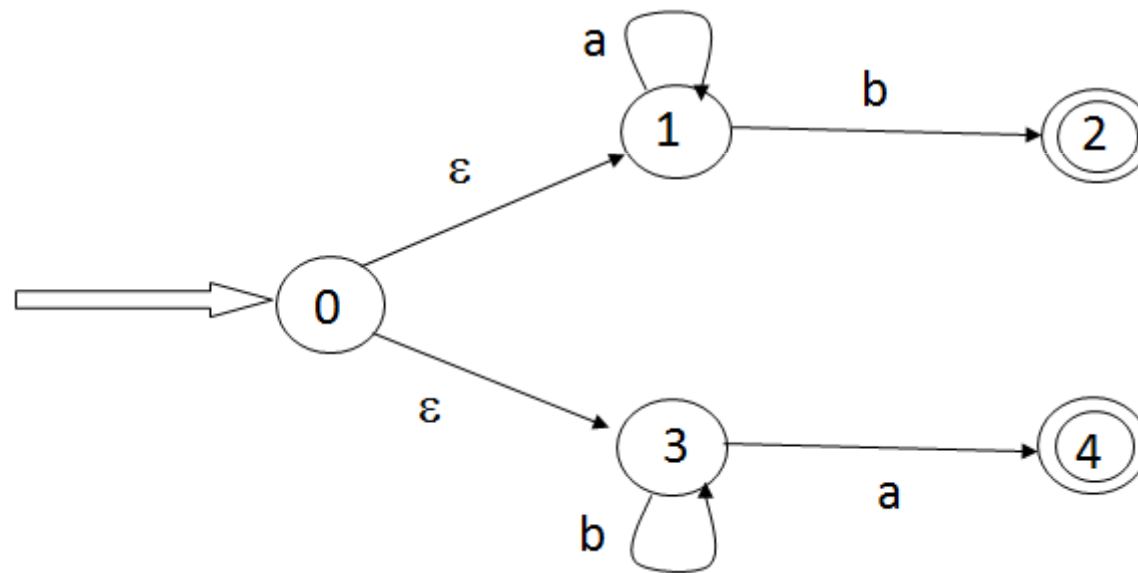
Transformation d'un AFN en AFD

Algorithme

D-états : ensemble d'états de l'AFD à construire ;
D-Trans : la table de transition de l'AFD ;
Initialement ε -fermeture(e_0) unique état de D-états et il est non marqué ;
Tant que \exists un état non marqué T dans D-états
 Faire marquer T ;
 Pour chaque symbole d'entrée a
 Faire U := ε -fermeture (Transiter (T, a));
 Si U \notin D-états
 Alors ajouter U à D-états et U est non marqué
 FinSi;
 D-Trans [T, a] := U;
 Fait;
Fait;
L'état initial de l'AFD est ε -fermeture(e_0);
Un état f est un état final de l'AFD si $\exists g \in f$ et g état final de l'AFN.

Transformation d'un AFN en AFD

Exemple



Transformation d'un AFN en AFD

Exemple

	a	b
{0, 1, 3}	{1, 4}	{2, 3}
{1, 4}	{1}	{2}
{2, 3}	{4}	{3}
{1}	{1}	{2}
{2}	-	-
{4}	-	-
{3}	{4}	{3}



Minimisation du nombre d'états de l'AFD

- Le processus d'obtention de l'automate d'états finis déterministe AFD obtenu :
 - intuitivement ou
 - par algorithme de transformation AFN vers AFD ou
 - par algorithme de transformation Expression Régulière vers AFD)
- n'est pas forcément minimal en nombre d'états



Minimisation du nombre d'états de l'AFD

- Il existe plusieurs algorithmes permettant d'accomplir cette tâche.
- A titre non exhaustif les algorithmes de John Hopcroft et de Edward Moore.



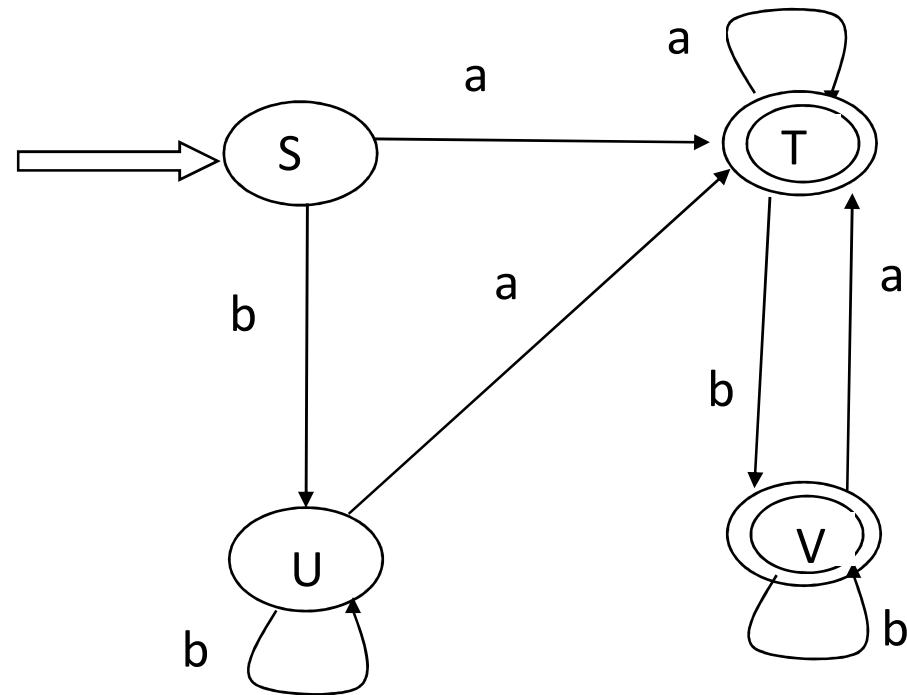
Minimisation du nombre d'états de l'AFD

Algorithme de Moore

- i) Construire une partition initiale Π des états de l'AFN avec 2 groupes; les états d'acceptation et les autres états ;
- ii) Obtenir une nouvelle partition Π' par :
Pour chaque groupe G de Π
Faire
 - Partition de G en sous-groupes de telle sorte que deux états s et t soient dans le même groupe si et seulement si, pour tout symbole a , les états s et t ont des transitions sur a vers des états du même groupe de Π ;
 - Remplacer G dans Π' par tous les sous-groupes ainsi formésFinPour;
- iii) Si $\Pi' = \Pi$ Alors aller à iv) Sinon $\Pi = \Pi'$; aller à ii) FinSi ;
- iv) L'état de départ de l'AFD est le représentant du groupe qui contient l'état de départ de l'AFN ; les états d'acceptation de l'AFD sont les représentants des états finaux de l'AFN;
- v) Supprimer de l'AFD les états non accessibles.

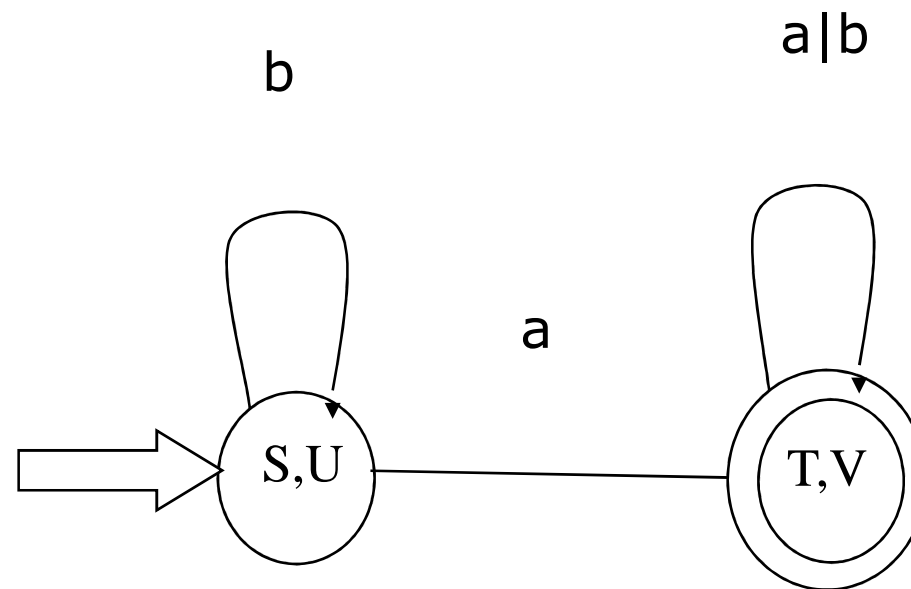
Minimisation du nombre d'états de l'AFD

Exemple 1



Minimisation du nombre d'états de l'AFD

Exemple 1



Minimisation du nombre d'états de l'AFD

Exemple 2

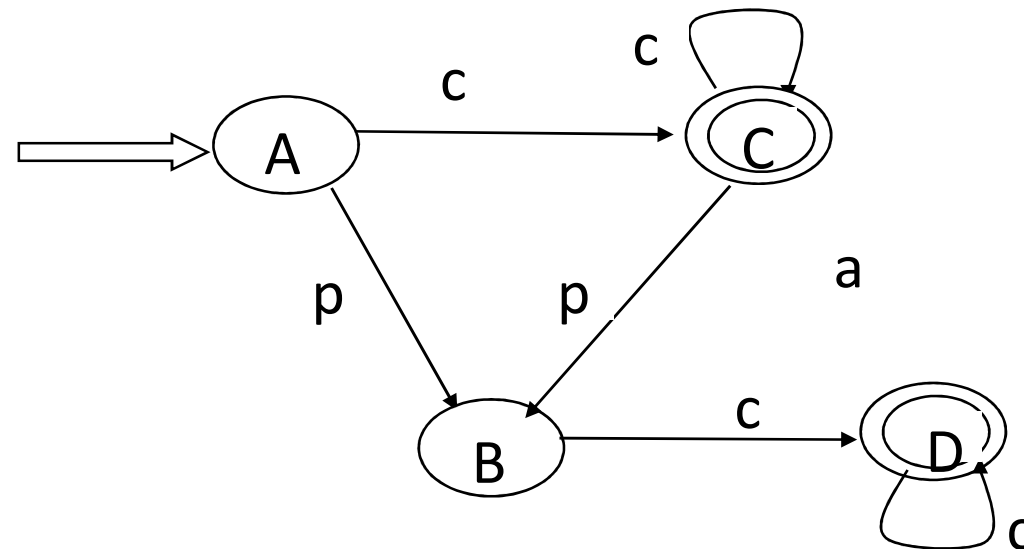
- Table de transition :

	c	p
0	1	4
2	3	-
4	5	-
1	1	2
3	3	-
5	5	-

Minimisation du nombre d'états de l'AFD

Exemple 2

- AFD minimal :





Transformation expression régulière en automate non déterministe

- Il existe plusieurs méthodes pour transformer une expression régulière en automate d'états finis non déterministe.
- A titre indicatif les méthodes de Ken Thompson et de Victor Glushkov.



Transformation ER en AFN

Construction de Thompson

- Procède d'une manière incrémentale et simple.
- Dans cette technique, on décompose l'expression régulière en composantes simples, on construit leur automate selon des règles spécifiques et on compose ensuite les automates obtenus pour atteindre l'automate final.



Transformation ER en AFN

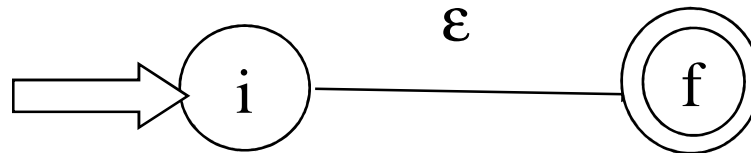
Construction de Thompson

- Cette méthode a été utilisée pour l'implémentation de la commande grep (***g**lobal / **r**egular **e**xpression / **p**rint*) du système Unix.
- Cette commande permet de chercher une chaîne définie par une expression régulière dans un fichier.

Transformation ER en AFN

Construction de Thompson

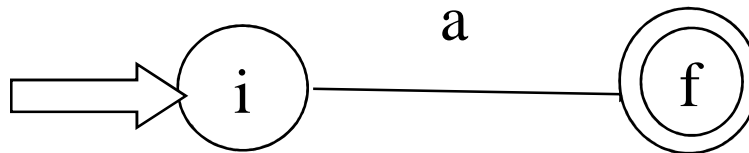
- i. Pour l'expression ε on construit l'automate suivant :



Transformation ER en AFN

Construction de Thompson

- ii. Pour l'expression a on construit l'automate suivant :

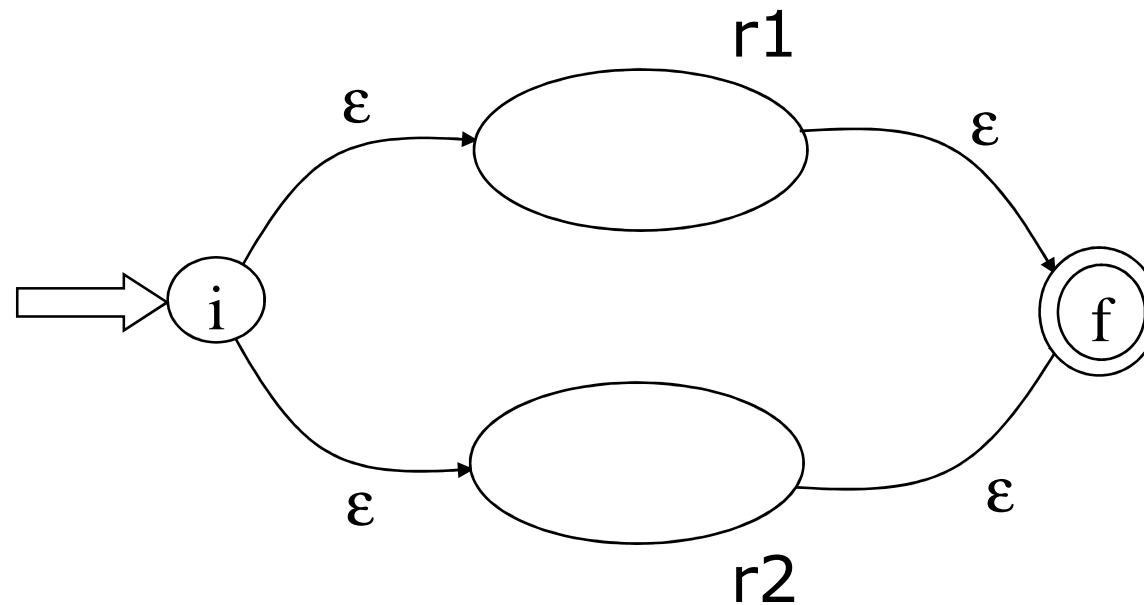


Transformation ER en AFN

Construction de Thompson

iii. Si $r1$ et $r2$ sont deux ER :

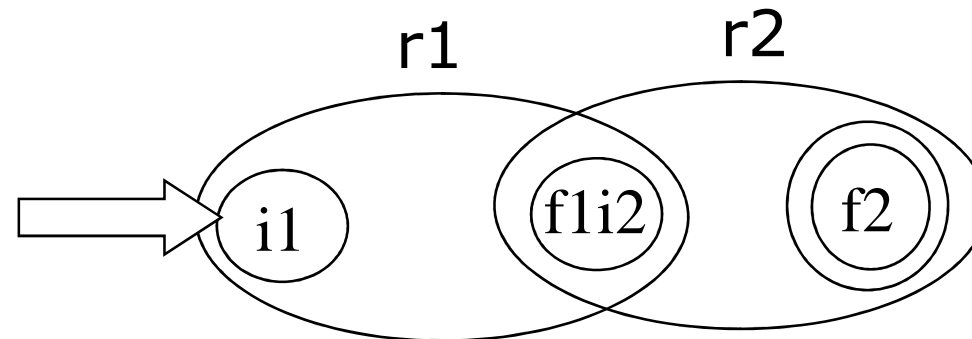
■ Pour $r1 \mid r2$ on construit :



Transformation ER en AFN

Construction de Thompson

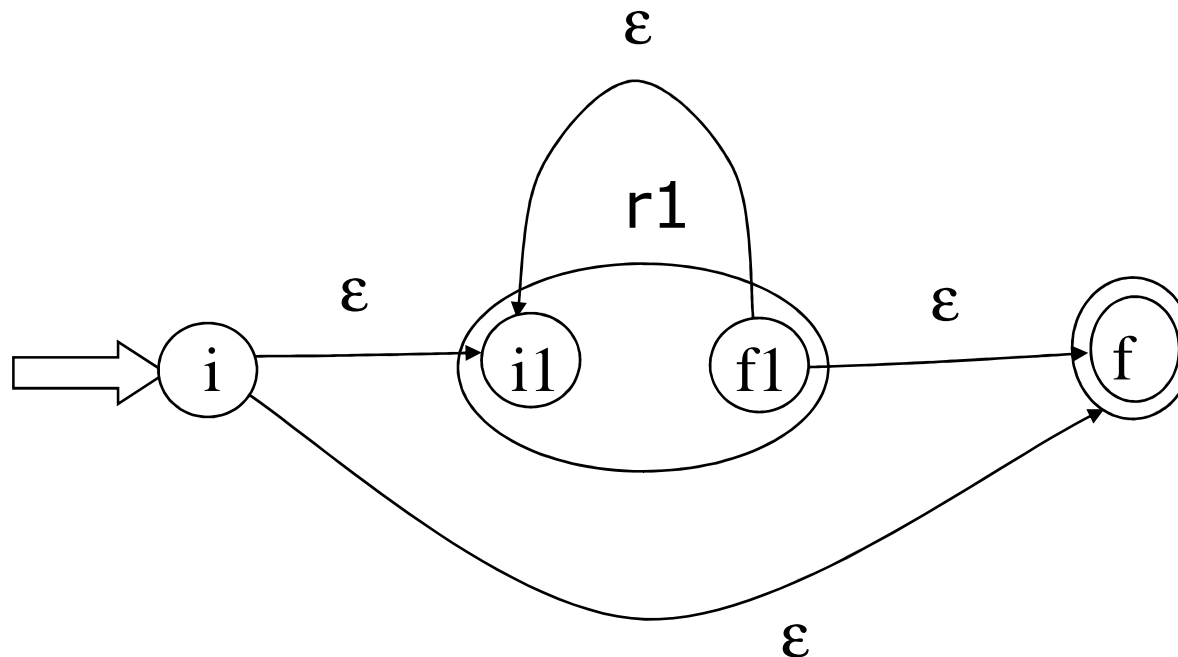
- iii. Si $r1$ et $r2$ sont deux ER :
 - Pour $r1 \cdot r2$ on construit :



Transformation ER en AFN

Construction de Thompson

- Pour r^* :





Transformation ER en AFN

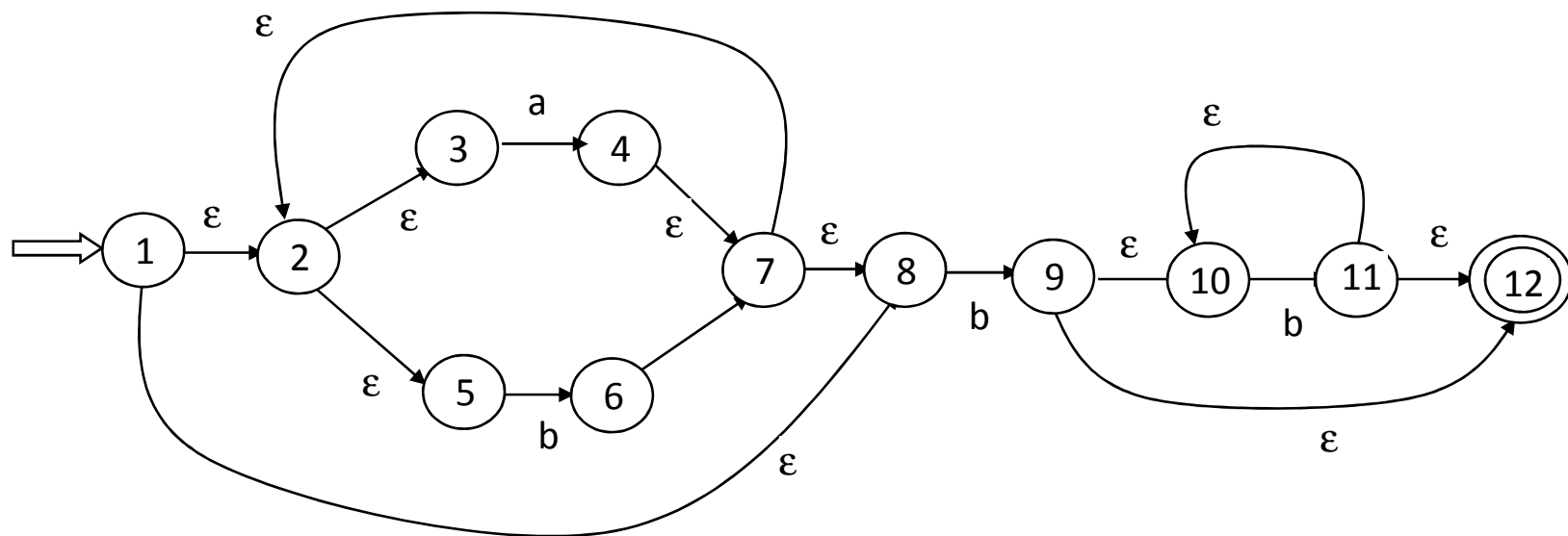
Construction de Thompson

- L'automate obtenu a un seul état final.
- Il n'y a pas de transition entrante sur l'état initial.
- Il n'y a pas de transition sortante de l'état final.

Transformation ER en AFN

Construction de Thompson

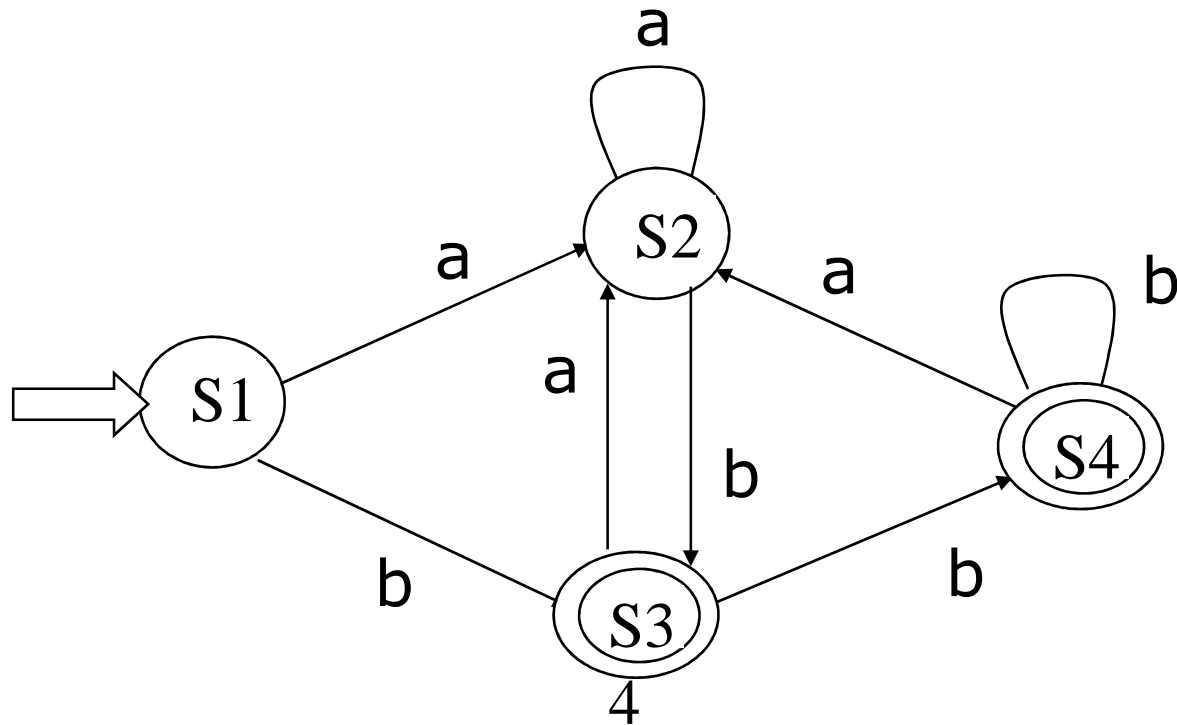
- Construire l'AFN correspondant à l'expression régulière : $(a|b)^* b b^*$



Transformation ER en AFN

Construction de Thompson

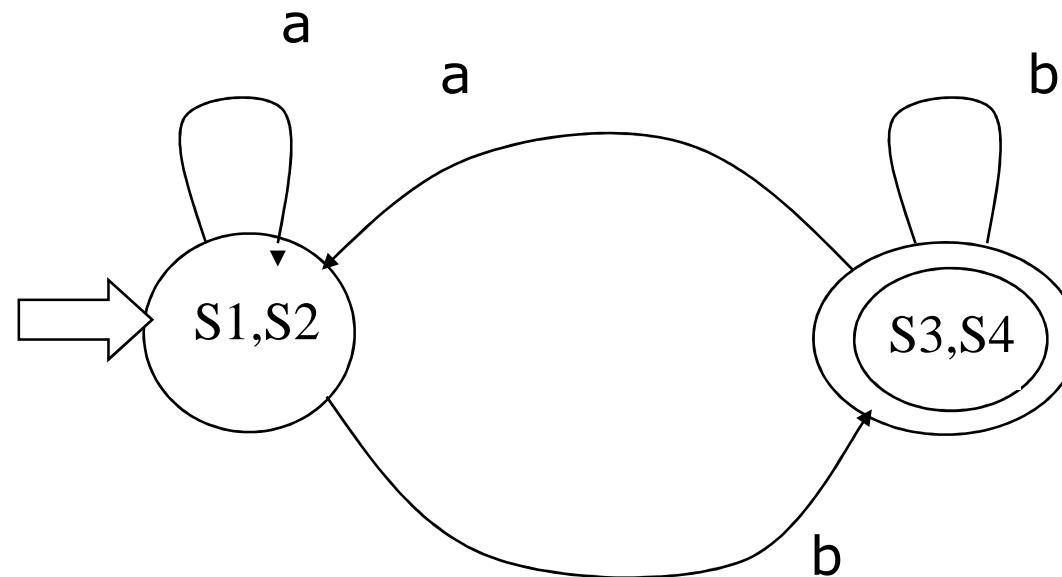
- Rendre l'automate déterministe :



Transformation ER en AFN

Construction de Thompson

- Minimisation du nombre d'états :





Transformation ER en AFN

Construction de Glushkov

- Le nombre d'états obtenu par la construction de Thompson est très grand.
- Avec la méthode de Glushkov, si n est le nombre de symboles de l'expression régulière alors le nombre d'états de l'AFN est égal à $(n+1)$.
- Cet AFN ne contiendra pas de transitions vides.



Transformation ER en AFN

Construction de Glushkov

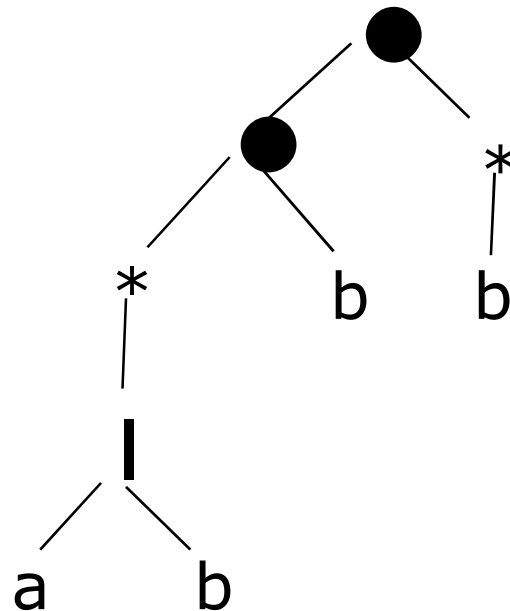
i. Arbre abstrait de l'ER :

- Les feuilles de l'arbre sont les symboles de l'expression régulière et les nœuds internes sont les opérateurs (alternative, concaténation et fermeture).

Transformation ER en AFN

Construction de Glushkov

i. Exemple Arbre abstrait de l'ER $(a|b)^* b b^*$:





Transformation ER en AFN

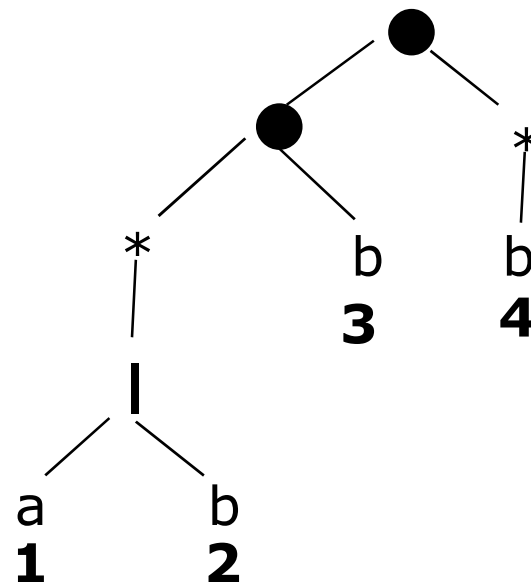
Construction de Glushkov

- ii. Positions et Expression régulière :
 - Etiqueter les feuilles de l'arbre abstrait par des numéros de position incrémentaux (parcours en profondeur).

Transformation ER en AFN

Construction de Glushkov

ii. Positions et Expression régulière - Exemple :





Transformation ER en AFN

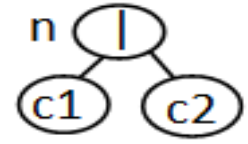
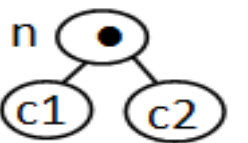
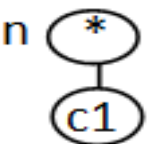
Construction de Glushkov

- ii. Positions et Expression régulière – Exemple :
l'ER précédente est notée :
- $(a_1|b_2)^* b_3 b_4^*$
 - Les positions de cette expression régulière sont :
 $\{a_1, b_2, b_3, b_4\}$

Transformation ER en AFN

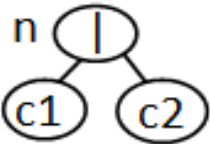
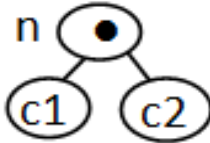
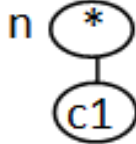
Construction de Glushkov

iii. Fonctions *vide*, *prem_pos*, *dern_pos* et *suiv_pos*

Nœud n	vide(n)
feuille étiquetée ε	true
feuille étiquetée avec la position i	false
	<u>vide(c1)</u> OR vide(c2)
	<u>vide(c1)</u> AND vide(c2)
	true

Transformation ER en AFN

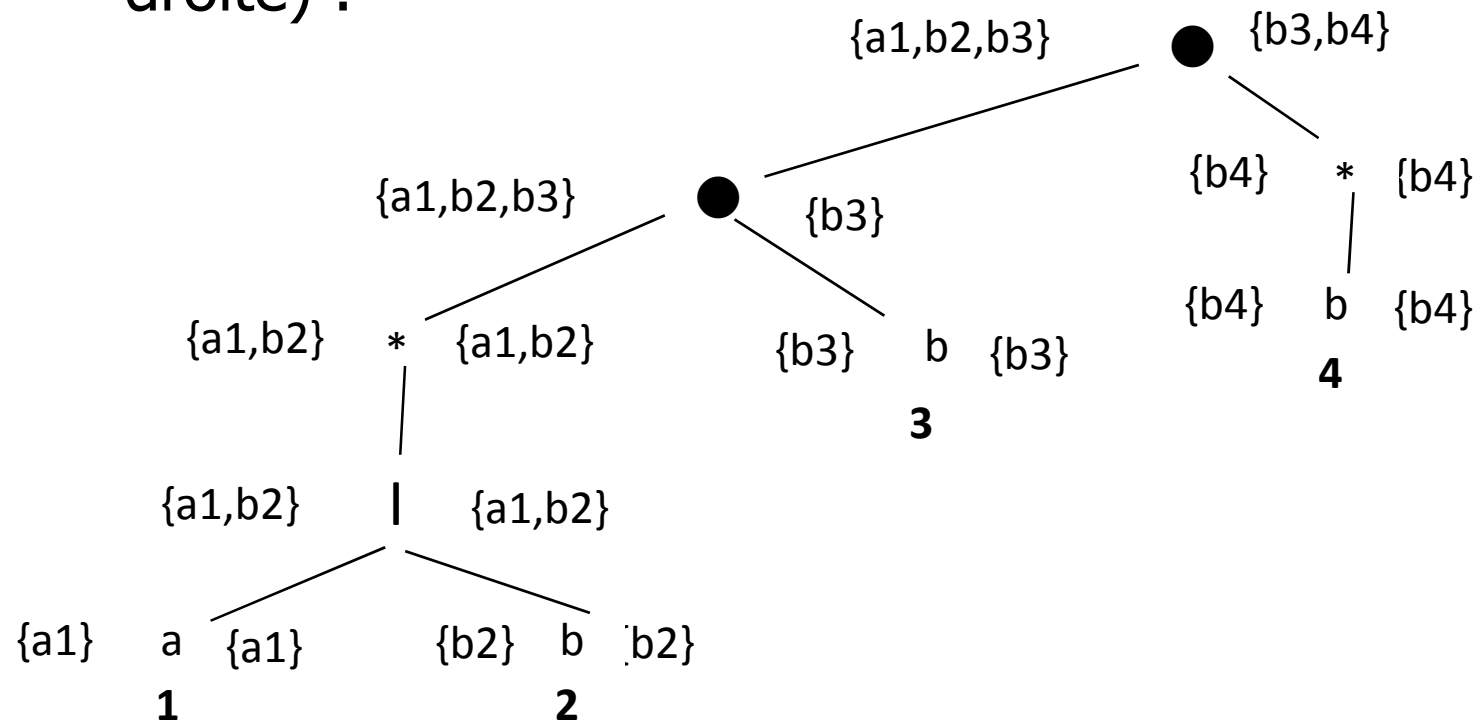
Construction de Glushkov

Nœud n	prem_pos(n)	dern_pos(n)
feuille étiquetée ε	\emptyset	\emptyset
feuille étiquetée avec la position i	$\{i\}$	$\{i\}$
	$\text{prem_pos}(c1) \cup \text{prem_pos}(c2)$	$\text{dern_pos}(c1) \cup \text{dern_pos}(c2)$
	If $\text{vide}(c1)$ then $\text{prem_pos}(c1) \cup \text{prem_pos}(c2)$ else $\text{prem_pos}(c1)$	If $\text{vide}(c2)$ then $\text{dern_pos}(c1) \cup \text{dern_pos}(c2)$ else $\text{dern_pos}(c2)$
	$\text{prem_pos}(c1)$	$\text{dern_pos}(c1)$

Transformation ER en AFN

Construction de Glushkov

iii. Exemple ***prem_pos*** (à gauche) , ***dern_pos*** (à droite) :





Transformation ER en AFN

Construction de Glushkov

- La fonction ***suiv_pos*** décrit la position qui peut suivre une position donnée dans l'arbre abstrait. Deux règles de base :
- Si n est un nœud concaténation ' \bullet ' avec comme fils gauche $c1$ et fils droit $c2$ et i est une position dans ***dern_pos***($c1$) alors toutes les positions dans ***prem_pos***($c2$) sont dans ***suiv_pos***(i).
- Si n est un nœud fermeture ' $*$ ' et i est une position dans ***dern_pos***(n) alors toutes les positions de ***prem_pos***(n) sont dans ***suiv_pos***(i).



Transformation ER en AFN

Construction de Glushkov

- Exemple suiv_pos :

Noeud	Suiv_pos
a_1	$\{a_1, b_2, b_3\}$
b_2	$\{a_1, b_2, b_3\}$
b_3	$\{b_4\}$
b_4	$\{b_4\}$



Transformation ER en AFN

Construction de Glushkov

iv.

L'automate de Glushkov est défini par $A=(Q, \Sigma, \delta, \{i\}, F)$ où :

- $Q = \{\text{positions de l'expression régulière}\} \cup \{i\}$
- Σ est l'alphabet;
- $\delta: Q \times \Sigma \rightarrow Q$ est la "fonction" de transition;
- i est l'état initial (qu'on ajoute);
- $F = \text{dern-pos}(\text{racine-arbre}) \cup \{i\}$ si $\text{vide}(\text{racine-arbre}) = \text{true}$
- $F = \text{dern-pos}(\text{racine-arbre})$ si $\text{vide}(\text{racine-arbre}) = \text{false}$.



Transformation ER en AFN

Construction de Glushkov

- Si (α, β) sont des positions de l'expression régulière, la fonction de transition $\delta: Q \times \Sigma \rightarrow Q$ est définie par :
- $\{ i \xrightarrow{s(\alpha)} \alpha / \alpha \in \text{prem-pos}(\text{racine-arbre}) \}$
 $\cup \{ \alpha \xrightarrow{s(\beta)} \beta / \beta \in \text{suiv-pos}(a) \}$
- Où la fonction $s(\alpha)$ enlève l'indice à la position α



Transformation ER en AFN

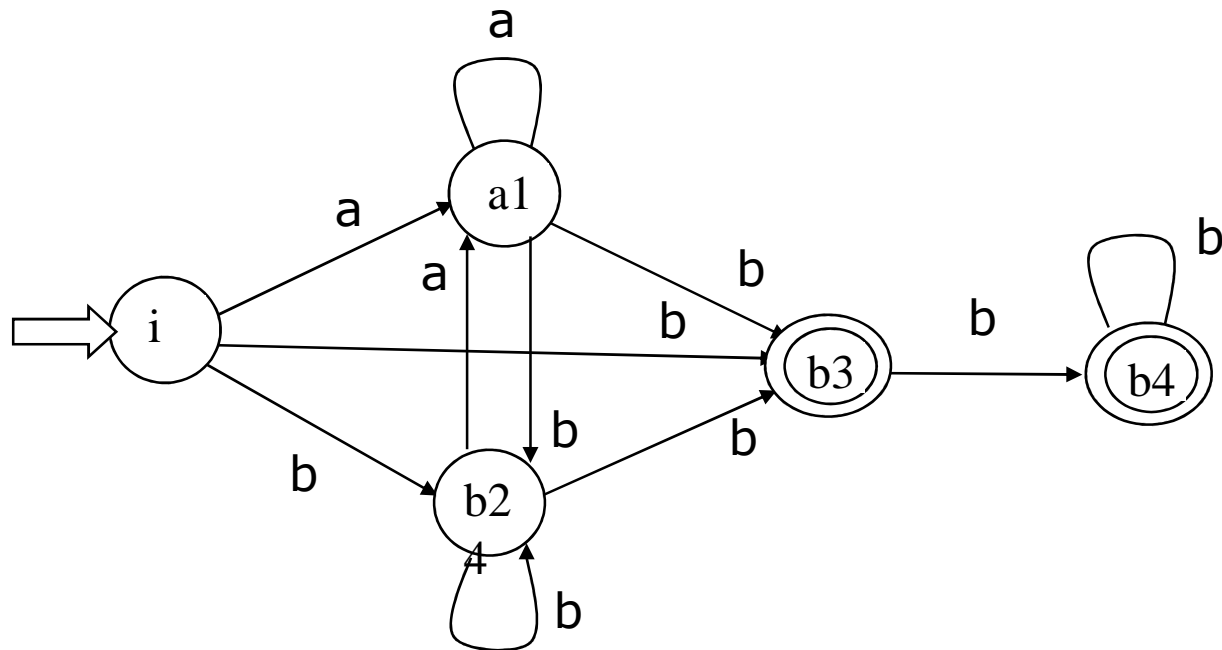
Construction de Glushkov

- L'AFN associé à l'expression régulière précédente aura 5 états :
 - les 4 positions $\{a_1, b_2, b_3, b_4\}$ et l'état initial $\{i\}$.
 - L'ensemble des états finaux $F = \{b_3, b_4\}$.

Transformation ER en AFN

Construction de Glushkov

- Application :





Transformation automate déterministe en expression régulière

- **Entrée** : Un automate d'états finis déterministe (AFD) reconnaissant les mots du langage L
- **Sortie** : Une expression régulière R décrivant le langage L



Transformation automate déterministe en expression régulière

- Cette transformation non nécessaire à un compilateur peut être utile si on veut sauvegarder un AFD sous une forme compacte (ou compressée).



Transformation automate déterministe en expression régulière

- Plusieurs techniques permettent de remonter à l'expression régulière à partir d'un AFD.
- Quelques méthodes : la fermeture transitive, la méthode de suppression d'état et la méthode algébrique de Brzozowski.



Transformation automate déterministe en expression régulière

Méthode de Brzozowski

- Cette méthode crée un système d'équations d'expressions régulières.
- Pour chaque état de l'AFD correspond une inconnue (expression régulière) dans le système.
- Une fois ce système résolu, l'expression régulière associée à l'état initial de l'AFD décrit le langage reconnu par l'AFD.



Transformation automate déterministe en expression régulière

Méthode de Brzozowski

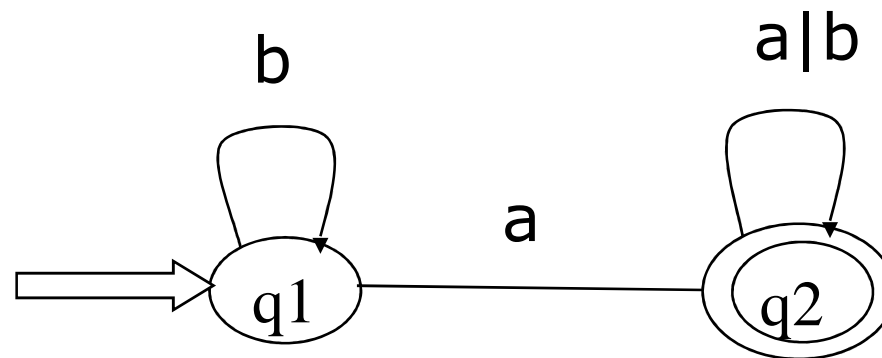
- **Génération du système d'équations :**
- Pour chaque état q_i de l'AFD est définie une équation :

$$R_i = \cup \text{termes}$$

- Les termes sont définis par les transitions sortantes de q_i .
- Pour une transition étiquetée a ($a \in \Sigma$) de l'état q_i vers un état q_j est défini le terme $a.R_j$ (a concaténé à R_j).
- Si q_i est un état final, ajouter le terme ε (mot vide) à l'équation de R_i .

Transformation automate déterministe en expression régulière

Méthode de Brzozowski - Exemple



- Le système d'équations correspondant est donné ci-après :
- $R_1 = b.R_1 \mid a.R_2$
- $R_2 = a.R_2 \mid b.R_2 \mid \varepsilon$



Transformation automate déterministe en expression régulière

Méthode de Brzozowski

- **Résolution du système d'équations :**
 - Le système d'équations est résolu par substitutions pour trouver R_1 qui est l'expression régulière décrivant le langage L .
 - Un problème est posé quand une inconnue R_k apparaît dans les parties gauche et droite d'une équation.



Transformation automate déterministe en expression régulière

Méthode de Brzozowski – Théorème d'Arden

- Etant donnée une équation de la forme :

$$R_k = A.R_k | B \text{ (où } \varepsilon \notin A),$$

- L'équation a pour solution :

$$R_k = A^*.B$$

Transformation automate déterministe en expression régulière

Méthode de Brzozowski - Exemple

- Résolution du système trouvé précédemment.
 - $R_1 = b.R_1 \mid a.R_2$
 - $R_2 = a.R_2 \mid b.R_2 \mid \varepsilon$
- Nous commençons par R_2 en utilisant le théorème d'Arden :
 - $R_2 = a.R_2 \mid b.R_2 \mid \varepsilon$
 - $= (a \mid b).R_2 \mid \varepsilon$
 - $= (a \mid b)^*. \varepsilon$
 - $= (a \mid b)^*$

Transformation automate déterministe en expression régulière

Méthode de Brzozowski - Exemple

- Substituons R_2 dans R_1 et reprenons le processus :
 - $R_1 = b.R_1 \mid a.R_2$
 - $= b.R_1 \mid a.(a \mid b)^*$
 - $= b^*.a.(a \mid b)^*$
- Donc l'expression régulière $b^*.a.(a \mid b)^*$ décrit le langage L reconnu par l'AFD précédent.



ANALYSEUR LEXICAL

Contrôle par AFD vs. Contrôle par programme

- L'automate d'états finis déterministe est l'outil pratique pour effectuer des contrôles lexicaux à partir de spécifications données.
- Cependant, contrôler par AFD certaines spécifications est prohibitif du point de vue espace mémoire occupé par la table de transition de l'AFD.
- D'autre part, on ne peut contrôler par AFD certaines contraintes imposées sur les entités lexicales.
- D'où la nécessité de recourir à un compromis entre ce qui doit être contrôlé par AFD et ce qui doit être contrôlé par programme.



ANALYSEUR LEXICAL

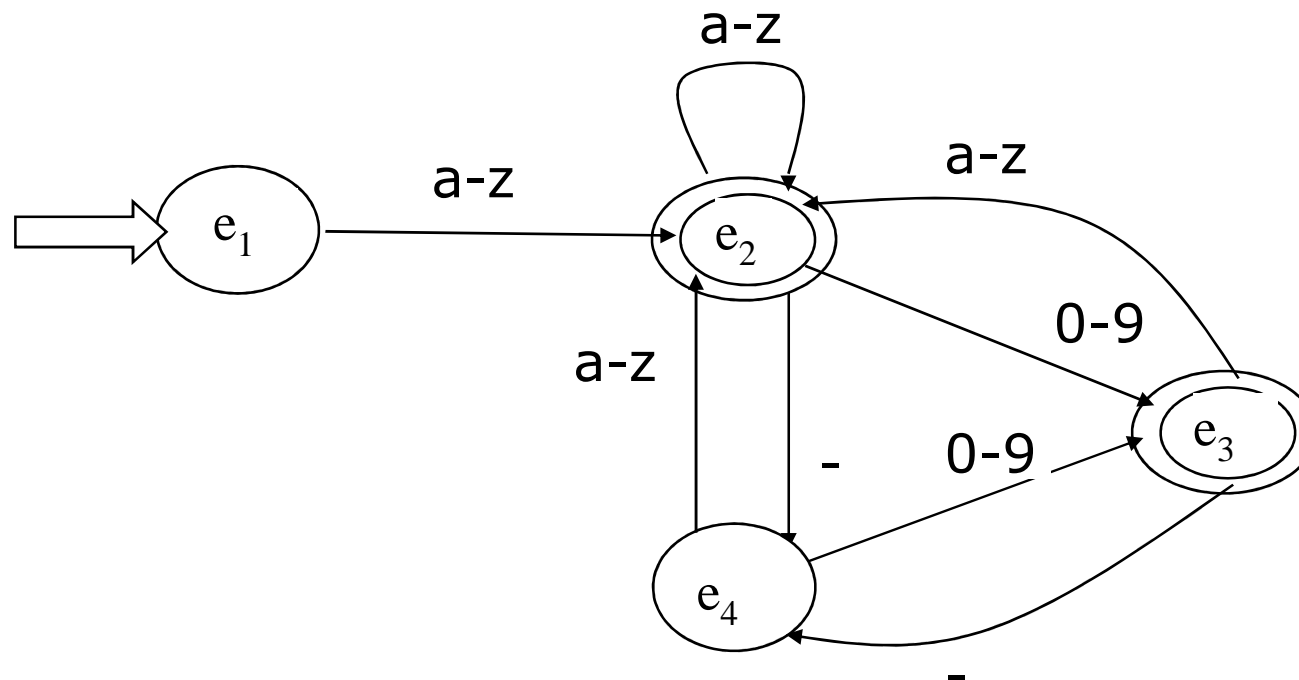
Contrôle par AFD vs. Contrôle par programme

- **Exemple :**
- On veut reconnaître les entités d'un langage L formées des lettres, des chiffres et des tirets '-'. Un mot de L présente les caractéristiques suivantes :
 - il doit commencer obligatoirement par une lettre et ne doit pas finir par un tiret
 - il ne doit pas contenir 2 tirets consécutifs ni 2 chiffres consécutifs
 - le nombre de chiffres est inférieur strictement au nombre de lettres
 - la longueur d'un mot est comprise entre 2 et 50 caractères

ANALYSEUR LEXICAL

Contrôle par AFD vs. Contrôle par programme

- **AFD pour les spécifications 1 et 2 :**





ANALYSEUR LEXICAL

Contrôle par AFD vs. Contrôle par programme

- **Programme d'analyse :**

$e = e_1$; $nbl = nbc = nbt = 0$; $c := \text{CarSuiv}$;

Tant que ($c \neq \text{'\# '}$ et $e \neq \emptyset$)

Faire **switch** (c)

 Lettre : $nbl++$;

 Chiffre : $nbc++$;

endswitch ;

$nbt++$; $e := \text{Transiter}(e, c)$; $c := \text{CarSuiv}()$;

Fait;

Si ($e \in \{e_2, e_3\}$) et ($2 \leq nbt \leq 50$) et ($nbc < nbl$)

Alors "Chaîne acceptée"

Sinon " Chaîne refusée" **Fsi**



ANALYSEUR LEXICAL

Mots-clés vs. Identificateurs

- Les mots-clés d'un langage de programmation sont des chaînes alphabétiques ayant une signification particulière.
- Ils identifient les structures des langages (instruction de répétition, instruction de test, ...).
- A titre d'exemple nous donnons quelques mots clés du langage C : if, else, while, do, switch, for.



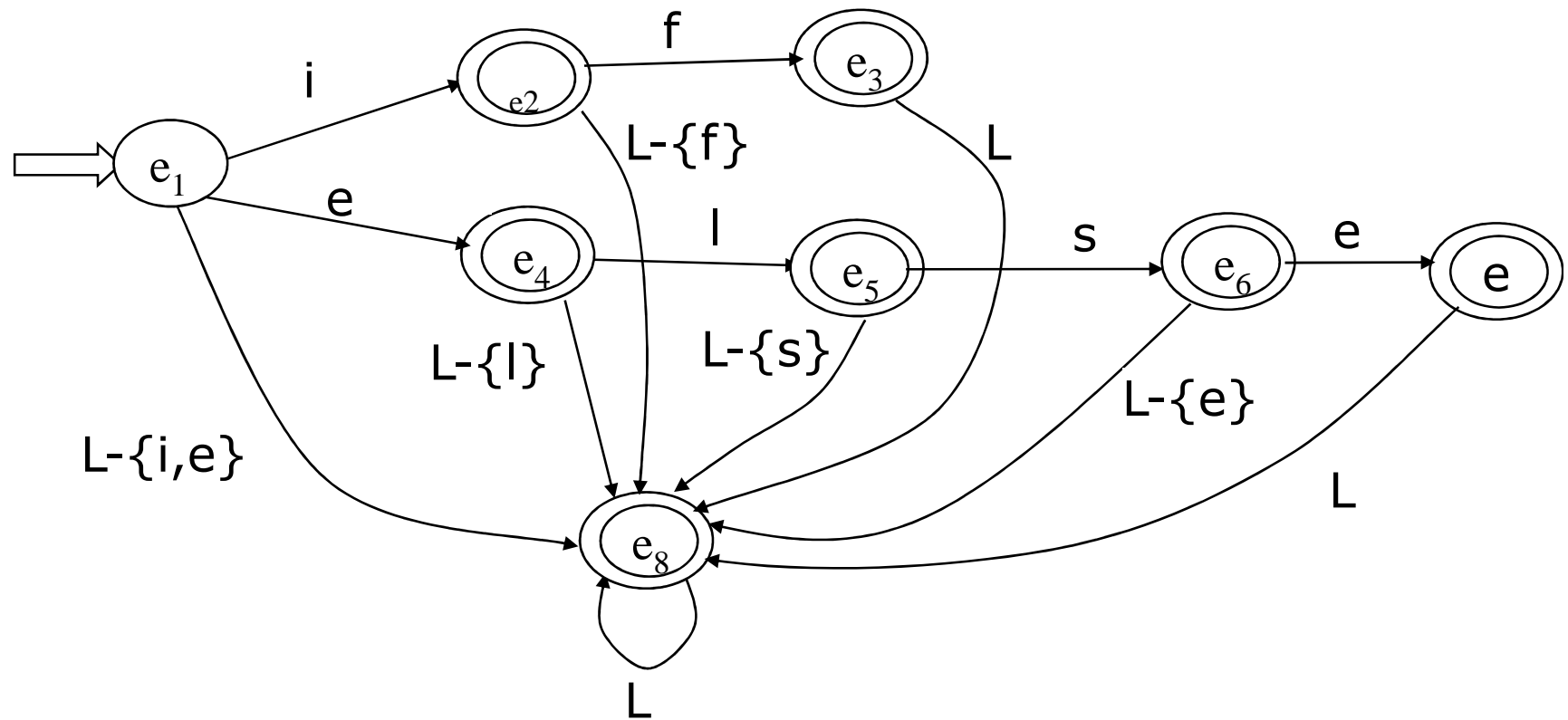
ANALYSEUR LEXICAL

Mots-clés vs. Identificateurs

- les mots-clés peuvent être contrôlés par un AFD.
- Dans l'exemple suivant, nous donnons un AFD qui permet l'identification des mots clés if, else et qui les différencie des identificateurs.

ANALYSEUR LEXICAL

Mots-clés vs. Identificateurs





ANALYSEUR LEXICAL

Mots-clés vs. Identificateurs

- Une autre approche consiste à ranger tous les mots clés dans une table des mots-clés et à reconnaître identificateurs et mots-clés par un même AFD.
- Les mots-clés sont reconnus en tant qu'identificateurs par l'AFD.
- A l'issue de cette étape, chaque chaîne est testée si elle appartient à la table des mots-clés afin de différencier les identificateurs des mots-clés.
- Le test d'appartenance peut être fait en s'aidant d'une fonction de hachage.



ANALYSEUR LEXICAL

Fonctionnement d'un analyseur lexical

- Le noyau de l'analyseur est l'automate d'états finis déterministe associé dénoté tout au long de ce chapitre par AFD.
- L'analyseur lit le programme source caractère par caractère de gauche à droite (pour les langues latines).
- Ces caractères servent de symboles de transition à l'AFD. A partir de l'état initial de l'AFD, on transite sur les caractères d'entrée jusqu'à atteindre un état final et ne plus pouvoir transiter sur le caractère courant.



ANALYSEUR LEXICAL

Fonctionnement d'un analyseur lexical

- L'état final atteint identifie l'entité lexicale reconnue. Une action est exécutée en adéquation avec l'entité reconnue.
- On réinitialise alors l'état courant de l'automate à l'état initial et on reprend le processus d'analyse sur le caractère courant jusqu'à atteindre la fin du fichier source.



ANALYSEUR LEXICAL

Fonctionnement d'un analyseur lexical

- Les caractères composant une entité lexicale doivent être sauvegardés dans un vecteur au cours du processus d'analyse car l'AFD n'a pas de "mémoire".
- S'il n'existe pas de transition d'un état non final sur le caractère courant alors une erreur à été détectée. L'analyseur pointera la position de l'erreur et donnera un message d'erreur en fonction du contexte de l'analyse.

