Chapitre 7: Le théorème de Kleene

Expressions régulières, automates finis, graphes de transitions: tous définissent les même langages.

- Méthode de démonstration

Théorème: Soient A,B,C des ensembles tels que $A \subseteq B$, $B \subseteq C$, $C \subseteq A$. Alors A=B=C.

Remarque: Les expressions régulières, les automates finis, ainsi que les graphes de transitions définissent des ensembles de langages.

Théorème de Kleene:

Tout langage défini par une expression régulière, ou un automate fini, ou un graphe de transition, peut être défini par toutes les trois méthodes.

Preuve du Théorème de Kleene: Une conséquence des trois lemmes suivants: (qu'on doit prouver)

Lemme 1: Tout langage reconnu par un automate fini est reconnu par un graphe de transition.

Lemme 2: Tout langage reconnu par un graphe de transition peut être défini par une expression régulière.

Lemme 3: Tout langage défini par une expression régulière est reconnu par un automate fini.

$$AF \subseteq GT \subseteq ER \subseteq AF$$

Lemme 1: Tout langage reconnu par un automate fini est reconnu par un graphe de transition.

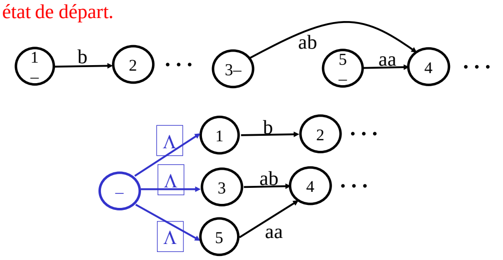
Preuve: Par définition, un automate fini est un graphe de transition.

Lemme 2: Tout langage reconnu par un graphe de transition peut être défini par une expression régulière.

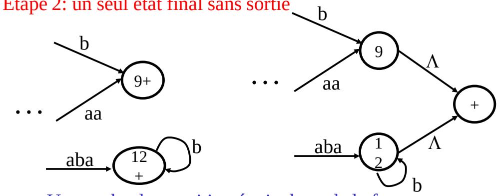
Preuve: Par un algorithme constructif, et qui fonctionne

1. pour tous les graphes de transitions
2. En un nombre fini d'étapes

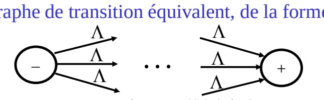
Étape 1: Transformer en un graphe de transition avec un seul état de départ.



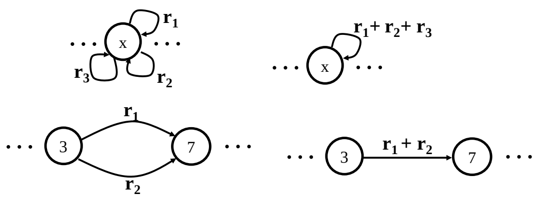
Étape 2: un seul état final sans sortie



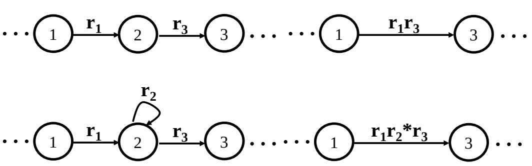
Un graphe de transition équivalent, de la forme:



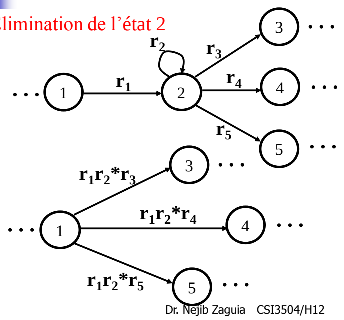
Étape 3a. Fusionner les arêtes qui ont les mêmes prédécesseurs et mêmes successeurs.

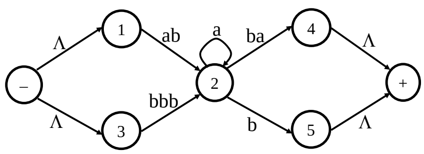


Étape 3.b: Opérations d'élimination d'états un par un



Élimination de l'état 2

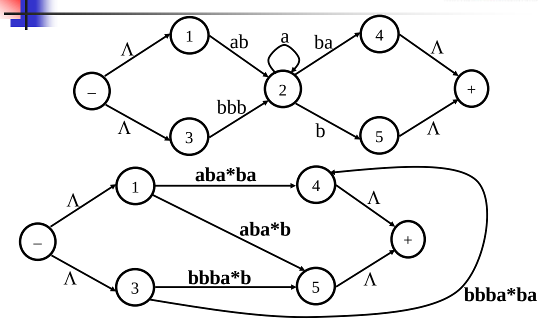


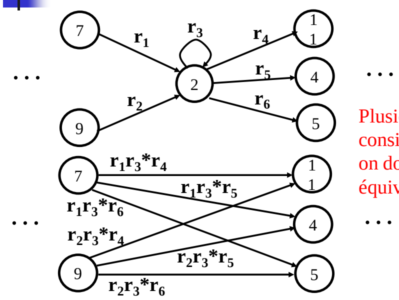


élimination de l'état 2,

les chemins passant par 2:

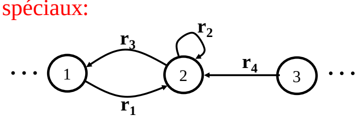
1→ 2→4, 1→2→5, 3→2→4, 3→2→5



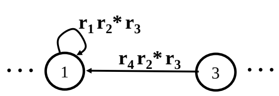


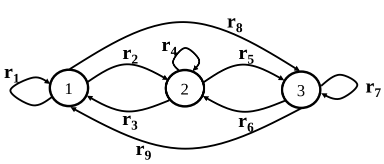
Plusieurs chemins a considérer: Toujours on doit avoir un GT équivalent.

Cas spéciaux:

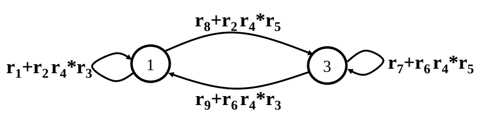


3 → 2→ 1, 1→2→1

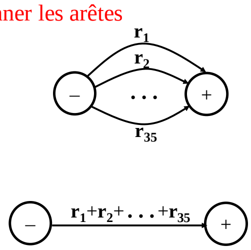




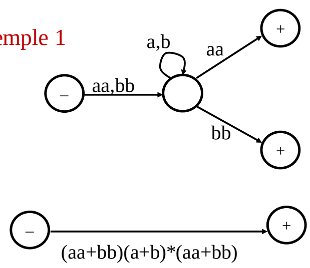
3 →2→1,
3→2→3,
1 →2→1,
1→2→3



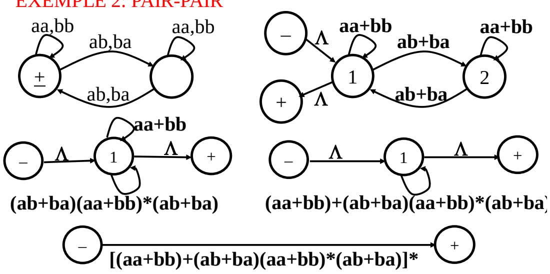
3a. Fusionner les arêtes



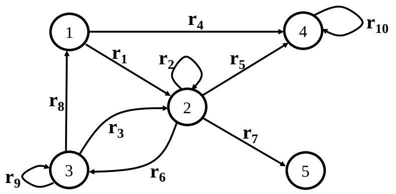
Exemple 1



EXEMPLE 2: PAIR-PAIR



Exemple 3



Graphe de transition → Expression régulière

Algorithme (et preuve)

1. Ajouter (si nécessaire) un état de départ unique sans entrée et un état final unique sans sortie.

Pour chaque état qui n'est ni état de départ ni état final, répéter 2 et 3.

2. Faire l'opération d'élimination d'état.
3. Fusionner les arêtes qui ont les mêmes prédécesseurs et mêmes successeurs.
4. Fusionner les arêtes entre l'état de départ et l'état final. L'étiquette de l'unique arête qui reste est l'expression régulière. S'il n'y a plus d'arête de l'état de départ vers l'état final alors l'expression régulière est ϕ.

Lemme 3: Tout langage défini par une expression régulière est reconnu par un automate fini.

Preuve: Par un algorithme constructif, en utilisant la définition récursive des expression régulières.

Rappel:

Définition: L'ensemble des expressions régulières sur un alphabet Σ est défini récursivement comme suit:

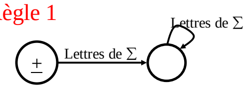
1. Chaque lettre de Σ ainsi que Λ est une expression régulière.
2. Si $\mathbf{r_1}$ et $\mathbf{r_2}$ sont des expressions régulières , alors il en est de même pour:
   1. $\mathbf{(r_1)}$
   2. $\mathbf{r_1 r_2}$
   3. $\mathbf{r_1+r_2}$
   4. $\mathbf{r_1^*}$
3. Aucune autre expression n'est régulière.

Exemple: Construire un automate fini qui accepte le langage:

(a+b)*(aa+bb)(a+b)*

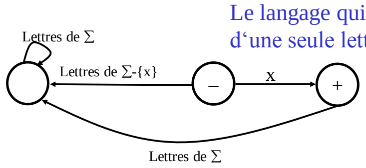
		Règle
1.	La lettre **a**	1
2.	La lettre **b**	1
3.	Le mot **aa** (en utilisant 1)	3
4.	Le mot **bb** (en utilisant 2)	3
5.	L'expression **aa+bb** (en utilisant 3 et 4)	2
6.	L'expression **a+b** (en utilisant 1 et 2)	2
7.	L'expression **(a+b)*** (en utilisant 6)	4
8.	L'expression **(a+b)*(aa+bb)** (en utilisant 7 et 5)	3
9.	L'expression **(a+b)*(aa+bb)(a+b)*** (en utilisant 8 et 7)	3

Règle 1



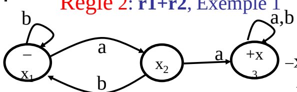
Le langage Λ

Le langage qui consiste d'une seule lettre x de Σ

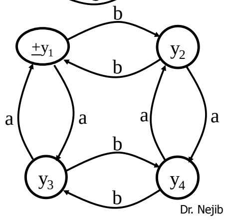


Règle 2: **r1+r2**, Exemple 1

Tous les mots contenant aa

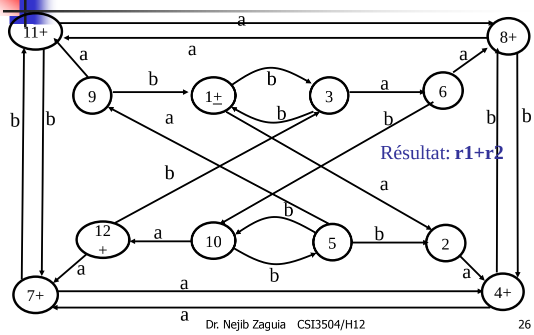


	a	b
$-x_1$	x_2	x_1
x_2	x_3	x_1
$+x_3$	x_3	x_3

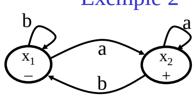


	a	b
$\pm y_1$	y_3	y_2
y_2	y_4	y_1
y_3	y_1	y_4
y_4	y_2	y_3

PAIR-PAIR

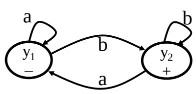


Exemple 2



	a	b
$-x_1$	x_2	x_1
$+x_2$	x_2	x_1

Mots qui se terminent par a



	a	b
$-y_1$	y_1	y_2
$+y_2$	y_1	y_2

Mots qui se terminent par b

Lemme 3: Tout langage défini par une expression régulière est reconnu par un automate fini.

Preuve: Par un algorithme constructif, en utilisant la définition récursive des expression régulières.

1. Il existe un automate fini que reconnaît le mot vide (Λ) et un automate fini qui reconnaît une seule lettre de Σ.
2. S'il existe un automate fini qui reconnaît $\mathbf{r_1}$ et un automate fini qui reconnaît $\mathbf{r_2}$, alors il existe un automate fini qui reconnaît le langage $\mathbf{r_1+r_2}$.
3. S'il existe un automate fini qui reconnaît $\mathbf{r_1}$ et un automate fini qui reconnaît $\mathbf{r_2}$, alors il existe un automate fini qui reconnaît leur concaténation $\mathbf{r_1r_2}$.
4. S'il existe un automate fini qui reconnaît $\mathbf{r}$ alors il existe un automate fini qui reconnaît la fermeture de ce langage $\mathbf{r^*}$.

Donc pour toute expression régulière, on peut lui construire un automate fini.

Algorithme 1: $\mathbf{r_1+r_2}$

Les données:

AF1: alphabet: Σ états: $x_1, x_2, x_3, \ldots$ état de départ: x_1

AF2: alphabet: Σ états: $y_1, y_2, y_3, \ldots$ état de départ: y_1

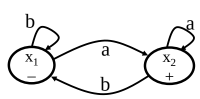
Ainsi que les états finaux, et les transitions

L'automate résultat:

alphabet: Σ états: $z_1, z_2, z_3, \ldots$ état de départ: x_1 ou y_1

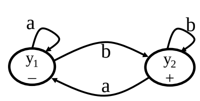
les transitions: si $z_i = x_j$ ou y_k et $x_j \rightarrow x_{nouveau}$ et $y_k \rightarrow y_{nouveau}$ (pour l'entrée p) alors $z_{nouveau} = (x_{nouveau}$ ou $y_{nouveau})$ pour l'entrée p.

Si $x_{nouveau}$ ou $y_{nouveau}$ est un état final, alors $z_{nouveau}$ est un état final.



	a	b
$-x_1$	x_2	x_1
$+x_2$	x_2	x_1

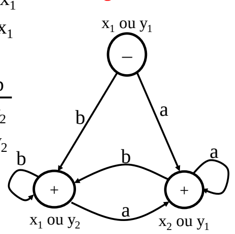
Mots qui se terminent par a



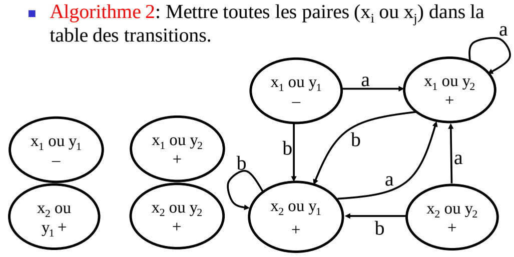
	a	b
$-y_1$	y_1	y_2
$+y_2$	y_1	y_2

Mots qui se terminent par b

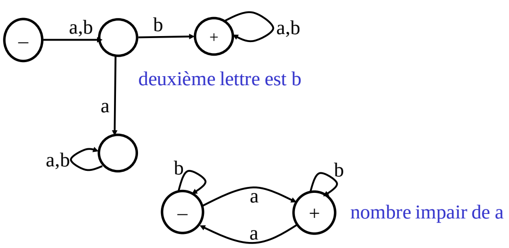
algorithme 1

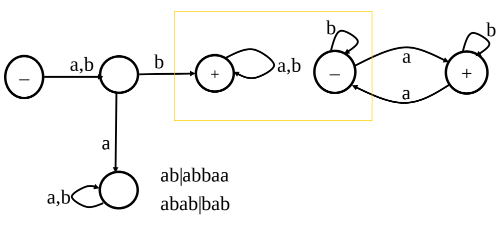


- Algorithme 2: Mettre toutes les paires (x_i ou x_j) dans la table des transitions.



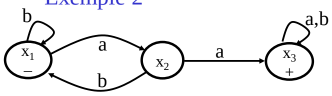
Règle 3: **r1r2**, Exemple 1



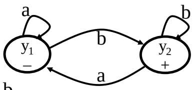


11

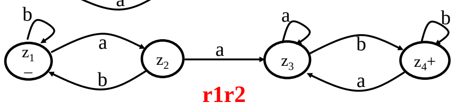
Exemple 2



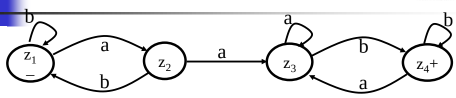
r_1: tous les mots qui contiennent aa



r_2: tous les mots qui se terminent par b



r1r2



On commence par créer les états z1=x1et z2=x2

(z2,a)= x3 "on continue sur AF" OU BIEN
y1"on passe a AF2, puisque x3 est final dans AF1"

(z2, a) = x3 ou y1= z3

(z3, a) = x3 ou y1= z3
(z3, b) = x3 ou y1 ou y2 = z4 +

(z4, a)= x3 ou y1=z3
(z4, b)=x3 ou y1 ou y2 = z4 +

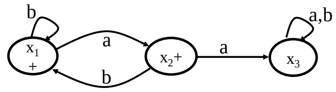
Règle 3: Concaténation: Résumé de l'algorithme

1. Créer un état z pour chaque état x du premier automate tel que x est accessible avec un chemin à partir d'un état initial sans passer par un état final.
2. Pour chaque état final x du premier automate, créer un état z = (x ou y_1), ou y_1 est l'état de départ du deuxième automate.
3. A partir des états créer à l'étape 2, on crée des états :

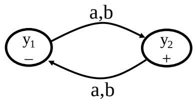
$$z=\begin{cases} x & \text{(l'état ou on continue sur le premier automate)} \\ & \text{OU} \\ y & \text{(l'état ou on passe sur le deuxième automate)} \end{cases}$$

4. Les états finaux sont les états qui contiennent un état final du deuxième automate.

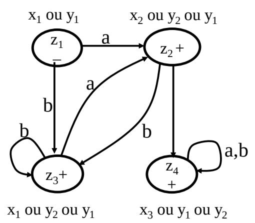
Exemple 3



$\mathbf{r_1}$: pas de aa

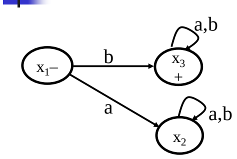


$\mathbf{r_2}$: nombre impair de lettres

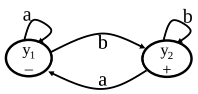


$\mathbf{r_1 r_2}$: tous les mots sauf Λ

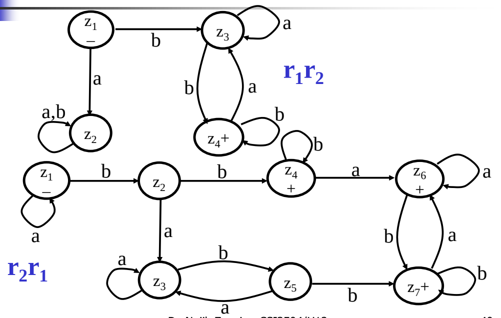
Exemple 4



$\mathbf{r_1}$: les mots qui commencent par b

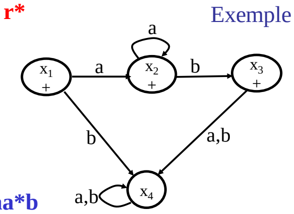


$\mathbf{r_2}$: les mots qui se terminent par b



Règle 4: **r***

Exemple 1



r: **a* + aa*b**

r*: les mots sans double b, et qui ne commencent pas par b.

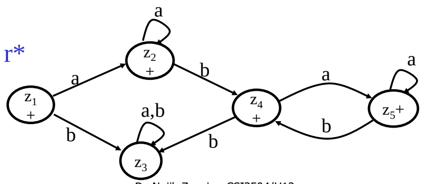
z1=x1 ± (z1, a) = x1 ou x2= z2 + (z1,b) = x4= z3

(z2,a)= x1 ou x2 =z2 (z2, b)= x1 ou x3 ou x4= z4 +

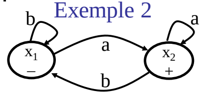
(z3, a)=z3 (z3,b) = z3

(z4,a)= x1 ou x2 ou x4= z5 + (z4, b)= x4 = z3

(z5,a)= x1 ou x2 ou x4= z5 (z5, b)= x1 ou x3 ou x4= z4

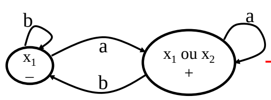


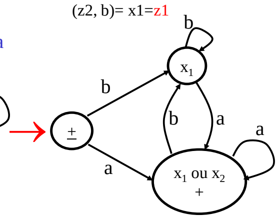
Exemple 2



$z1 = x1 -$
(z1, a) = x1 ou x2= z2 +
(z1,b) = z1
(z2,a)= x1 ou x2 =z2
(z2, b)= x1=z1

Les mots qui se terminent par a





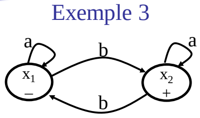
Problème avec Λ

Règle 4: r*: “Algorithme”

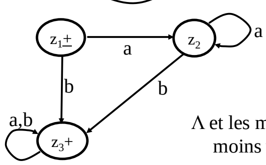
Donnée: un automate avec un ensemble d’états $\{x_1, x_2, x_3, \ldots\}$

- Pour chaque sous-ensemble d’états, on crée un état dans le nouveau automate. Il faut éliminer tout sous-ensemble qui contient un état final et ne contient pas l’état de départ.
- On fait une table de transitions pour tous les nouveaux états.
- On ajoute un état ±. Les transitions qui sortent de cet état sont les mêmes que les transitions qui sortent partir de l’état de départ original.
- Les états finaux sont les états qui contiennent au moins un état final de l’automate original.

Exemple 3



Les mots qui contiennent un nombre impair des b.

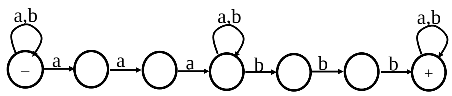


Λ et les mots qui contiennent au moins un b.

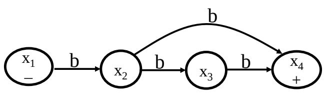
- Un automate non déterministe est défini par:
  1. Un ensemble fini non vide d'états avec un état désigné comme l'état de départ (ou initial) et quelques (peut-être aucun) états désignés comme les états finaux (ou états acceptants)
  2. Un alphabet Σ des lettres d'entrées
  3. Un ensemble fini de transitions qui fait correspondre à quelques pairs (état, lettre) un état. «étant dans un état et avec une entrée spécifique, cette fonction indique l'état (ou possiblement états) dans lequel on passe »

Remarque: Tout automate fini est un automate non déterministe.
Tout automate non déterministe est un graphe de transition.

Exemples d'automates non déterministes



(a+b)*aaa(a+b)*bbb(a+b)*



bb+bbb

Théorème: Tout langage reconnu par un automate non déterministe est également reconnu par un automate déterministe (i.e. un automate fini).

Preuve (1): Tout automate non déterministes est un graphe de transition:

Par lemme 2: graphe de transition → expression régulière

Par lemme 3: expression régulière → automate fini

Preuve (2): Par un algorithme constructif (Voir Livre page135).