Langages et Compilation

Généralisation de la notion d'automate fini

D'expression régulière vers automate fini

Rappel - Équivalence entre expression régulière et AF

Le théorème de Kleene

Un langage est reconnaissable par automate fini déterministe si et seulement si une expression régulière le représente.

$\Leftarrow$ Tout langage décrit par une expression régulière est reconnaissable par automate fini :

- Les langages $\emptyset$, $\{\varepsilon\}$ et $\{a\}$ pour toute lettre a de Σ sont reconnaissables par AF.
- La classe des langages reconnaissables par AF est close par union, concaténation et étoile.

$\Rightarrow$ On a un algorithme simple qui trouve une expression régulière à partir d'un automate fini

Ce qui nous intéresse, c'est une construction effective qui traduise une expression régulière en automate fini.

Ceci utilise des extensions des AF déterministes qui autorisent des transitions non-déterministes. Fait notable, ces extensions ne changent pas les capacités de reconnaissance (cf. déterminisation).

On a alors tous les ingrédients pour envisager la construction de Glushkov et celle de Thompson.

Automate fini non déterministe

Un **automate fini non déterministe** (**AFN**) est un quintuplet $\left(\Sigma, Q, \delta, I, F\right)$ où :

- Σ est un alphabet
- Q est un ensemble **fini** d'états
- δ est la fonction de transition de $Q \times \Sigma \to \mathcal{P}(Q)$
  ($\mathcal{P}(Q)$ l'ensemble de toutes les parties de Q)
- $I \subset Q$ est l'ensemble des états initiaux
- $F \subset Q$ est l'ensemble des états finaux

Automate fini non déterministe

Fonctionnement

Au temps initial, la machine est dans un des états initiaux $\boldsymbol{q_0} \in \boldsymbol{I}$ (non déterminisme)

Si à l'instant $\boldsymbol{t}$ la machine est dans l'état $\boldsymbol{q}$ et lit le symbole $\boldsymbol{a}$, alors à l'instant $\boldsymbol{t} + 1$

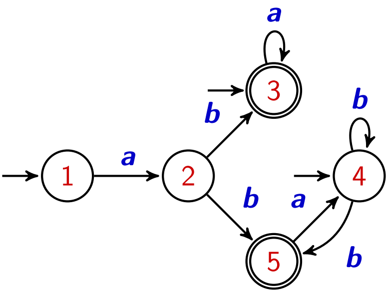
- si $\delta(\boldsymbol{q}, \boldsymbol{a}) = \emptyset$: la machine se bloque
- si $\delta(\boldsymbol{q}, \boldsymbol{a}) \neq \emptyset$: la machine entre dans l'un des états de $\delta(\boldsymbol{q}, \boldsymbol{a})$ (non déterminisme) et consomme $\boldsymbol{a}$.

Automate fini non déterministe

Exemple

$(\{a, b\}, \{1, 2, 3, 4, 5\}, \delta, \{1, 3, 4\}, \{3, 5\})$ où δ est définie par la table :

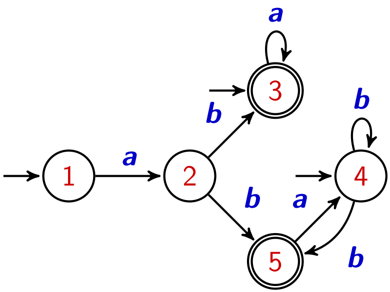
	a	b
$\rightarrow$ 1	$\{2\}$	$\emptyset$
2	$\emptyset$	$\{3, 5\}$
$\rightarrow$ 3	$\{3\}$	$\emptyset$
$\rightarrow$ 4	$\emptyset$	$\{4, 5\}$
5	$\{4\}$	$\emptyset$



Un mot u est **accepté** s'il existe, à partir d'un état initial $q_0 \in I$, un chemin étiqueté par u qui aboutit à un état d'acceptation $q_f \in F$.

Automate fini non déterministe

Exemple



abab est accepté : étiquète le chemin 1 2 5 4 5

autres calculs possibles 1 2 5 4 4 ($4 \notin F$), 1 2 3 3 blocage, 3 3 blocage, 4 blocage

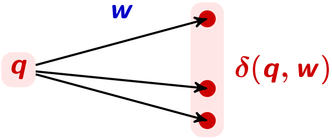
ba n'est pas accepté :

1 blocage, 3 blocage, 4 4 blocage, 4 5 4 ($4 \notin F$)

Automate fini non déterministe

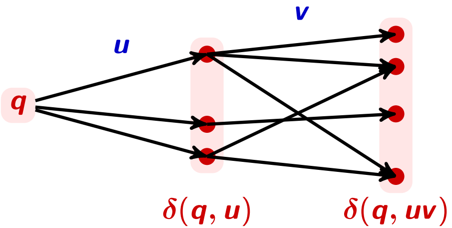
Prolongement de la fonction de transition δ en une fonction de $Q \times \Sigma^*$ dans $\mathcal{P}(Q)$:

$\delta(q, w)$ = l'ensemble des états accessibles en partant de q et après lecture du mot w



En toute généralité, on a pour tous mots u, v :

$$\delta(q, uv) = \bigcup_{r \in \delta(q,u)} \delta(r, v)$$



Automate fini non déterministe

Le langage reconnu par un AFN $\mathcal{A} = \left(\Sigma, Q, \delta, I, F\right)$ est l'ensemble des mots acceptés par $\mathcal{A}$

$$L(\mathcal{A}) = \left\{ w \in \Sigma^* \colon \bigcup_{i \in I} \delta(i, w) \cap F \neq \emptyset \right\}$$

Automate fini non déterministe

Les automates finis déterministes et les automates finis non déterministes ont la même capacité de calcul :

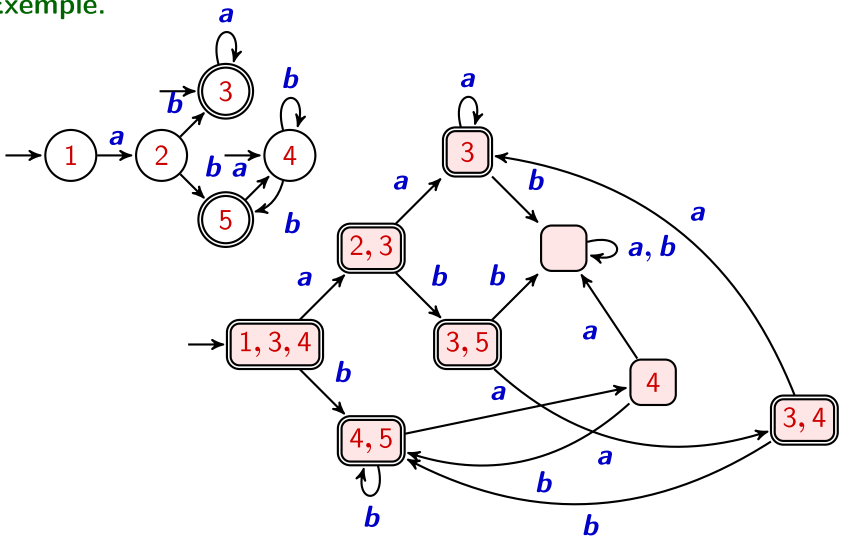
Le théorème de Rabin-Scott

Tout langage reconnu par un AFN l'est aussi par un AFD.

Preuve constructive : il existe un algorithme qui étant donné un AFN construit un AFD équivalent.

L'AFD simule en parallèle tous les calculs possibles de l'AFN. Son état initial est l'ensemble des états initiaux de l'AFN.
L'état atteint sur l'AFD après lecture du mot w correspond à l'ensemble des états accessibles sur l'AFN après lecture du mot w.
Les états de l'AFD sont ainsi des ensembles d'états de l'AFN.

Exemple.



Si l'AFN $\mathcal{A} = (\Sigma, Q, \delta, I, F)$ reconnaît L alors

l'AFD $\mathcal{A}_{det} = (\Sigma, Q_{det}, \delta_{det}, i_{det}, F_{det})$ où :

$Q_{det} = \mathcal{P}(Q)$ l'ensemble des parties de Q

δ_{det} la fonction de transition de $\mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q)$ définie par $\delta_{det}(X, a) = \bigcup_{q \in X} \delta(q, a)$

$i_{det} = I$

$F_{det} = \{X \subset Q : X \cap F \neq \emptyset\}$ l'ensemble des parties comprenant un état d'acceptation

reconnaît également L.

$$w \in L(\mathcal{A}) \Leftrightarrow \bigcup_{i \in I} \delta(i, w) \cap F \neq \emptyset \Leftrightarrow \delta_{det}(I, w) \cap F \neq \emptyset \Leftrightarrow \delta_{det}(I, w) \in F_{det}$$
$$\Leftrightarrow w \in L(\mathcal{A}_{det})$$

Automate fini non déterministe

Algorithme de déterminisation

Concrètement l'ensemble des états de l'AFD comprend uniquement les parties accessibles de l'état initial.
On les caractérise via un parcours du graphe de l'AFD qui visite tous les sommets accessibles à partir de son état initial.

Entrée : l'AFN $\mathcal{A} = (\Sigma, Q, \delta, I, F)$

Sortie : l'AFD $\mathcal{A}_{det} = (\Sigma, Q_{det}, \delta, I, F_{det})$ $\quad F_{det} = \{ X \subset Q_{det} : X \cap F \neq \emptyset \}$

Initialisation

$\quad Q_{det} = \{I\}$

$\quad$ VoirSucc = {I}

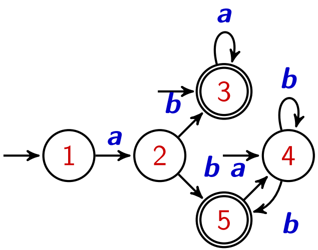
Traitement

```
tant que VoirSucc n'est pas vide faire
    extraire un élément X de VoirSucc
    pour toute lettre a de Σ
        T = ⋃_{q ∈ X} δ(q,a)
        δ_det(X,a) = T
        si T n'est pas dans Q_det alors
            ajouter T à Q_det
            ajouter T à VoirSucc
```

Exemple.

	a	b
$\rightarrow$ 1	$\{2\}$	$\emptyset$
2	$\emptyset$	$\{3,5\}$
$\rightarrow$ 3	$\{3\}$	$\emptyset$
$\rightarrow$ 4	$\emptyset$	$\{4,5\}$
5	$\{4\}$	$\emptyset$

	a	b
$\rightarrow I = \{1,3,4\}$	$\{2,3\}$	$\{4,5\}$
$\boxed{\{2,3\}}$	$\{3\}$	$\{3,5\}$
$\boxed{\{4,5\}}$	$\{4\}$	$\{4,5\}$
$\boxed{\{3\}}$	$\{3\}$	$\emptyset$
$\boxed{\{3,5\}}$	$\{3,4\}$	$\emptyset$
$\{4\}$	$\emptyset$	$\{4,5\}$
$\emptyset$	$\emptyset$	$\emptyset$
$\boxed{\{3,4\}}$	$\{3\}$	$\{4,5\}$



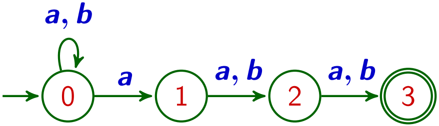
Quel est son coût ?

En $\boldsymbol{O}(|\Sigma| \times |\boldsymbol{Q_{det}}|)$: le coût du parcours du graphe de l'AFD qui a $|\boldsymbol{Q_{det}}|$ sommets et $|\boldsymbol{Q_{det}}| \times |\Sigma|$ arcs.

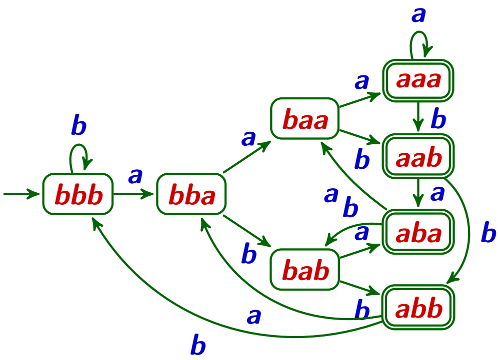
Hors le passage d'un AFN à un AFD peut coûter cher en nombre d'états.
Dans le pire cas, c'est exponentiel $|\boldsymbol{Q_{det}}| = |\mathcal{P}(\boldsymbol{Q})| = 2^{|\boldsymbol{Q}|}$.

Exemple : $(\boldsymbol{a} + \boldsymbol{b})^* \boldsymbol{a} (\boldsymbol{a} + \boldsymbol{b})^{\boldsymbol{n}-1}$

L'ensemble des mots dont la n-ème lettre en partant de la fin est un a est reconnu par un AFN avec $n+1$ mais tout AFD qui le reconnaît a au moins 2^n états.



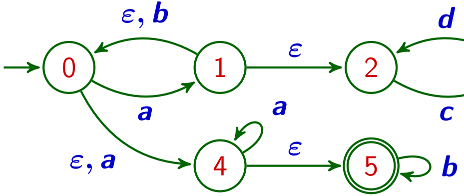
l'AFN pour $(a+b)^* a(a+b)^2$



l'AFD
coût de sa construction $\sim 2^3$

Automate fini non déterministe avec ε-transitions

Une généralisation des AFN avec des transitions étiquetées avec le mot vide ε. Lorsque l'automate choisit une telle transition, il ne consomme pas de lettre.



Un mot $\boldsymbol{w}$ est **accepté**, s'il existe un chemin étiqueté par $\boldsymbol{w}$ d'un état initial à un état final.
(chemin de longueur $|w|+$ le nombre de ε-transitions empruntées)

ac est accepté $\quad 0 \xrightarrow{a} 1 \xrightarrow{\varepsilon} 2 \xrightarrow{c} 3$

aaba est accepté $\quad 0 \xrightarrow{a} 1 \xrightarrow{\varepsilon} 0 \xrightarrow{a} 1 \xrightarrow{b} 0 \xrightarrow{a} 4 \xrightarrow{\varepsilon} 5$

abba n'est pas accepté

Automate fini non déterministe avec ε-transitions

Ajouter des ε-transitions n'augmente pas les capacités de reconnaissance de l'automate :

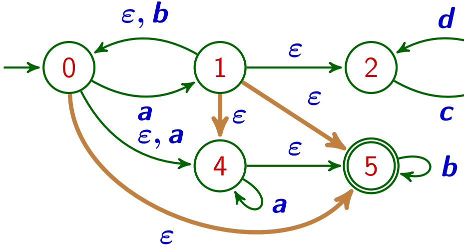
Proposition

Tout langage reconnu par un AFN avec ε-transitions l'est aussi par un AFN sans ε-transitions.

Preuve constructive : il existe un algorithme qui étant donné un AFN avec ε-transitions construit un AFN sans ε-transitions équivalent et avec le même nombre d'états.

Étape 1. Fermeture transitive sur les ε-transitions.

Pour chaque état q, on définit `Cloture`(q) l'ensemble des états accessibles à partir de q par des ε-transitions.
On ajoute à l'automate initial une ε-transition de q vers p pour tout p dans `Cloture`(q).

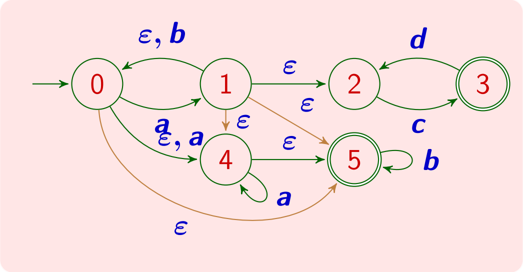


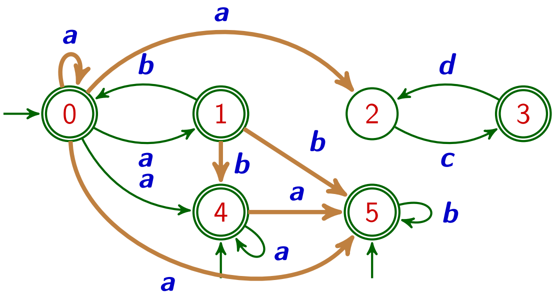
Étape 2. Construction de l'AFN sans ε-transitions par fermeture avant.

On ajoute à l'automate de départ :

- dans l'ensemble des états initiaux : les états p extrémités d'une ε-transition dont l'origine est un état initial ;
- dans l'ensemble des états finaux : les états p origines d'une ε-transition dont l'extrémité est un état final ;
- dans l'ensemble des transitions : les transitions $p \xrightarrow{a} r$ pour chaque séquence $p \xrightarrow{a} q \xrightarrow{\varepsilon} r$

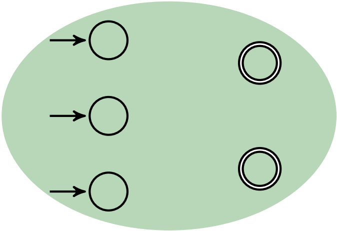
On supprime ensuite les ε-transitions.

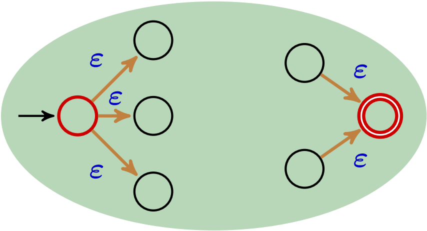

ε, b
d
0
1
ε
2
3
ε
ε
c
a
ε, a
ε
4
5
b
a
ε


a
a
b
d
0
1
2
3
b
a
b
c
a
a
4
5
b
a
a

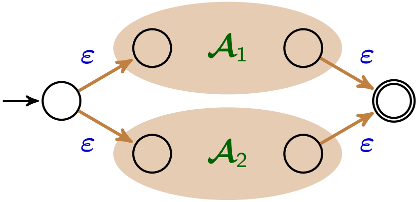
Automate fini non déterministe avec ε-transitions

Grâce aux ε-transitions, on peut toujours se ramener à des automates (avec ε-transitions) qui ont un unique état initial et un unique état final.

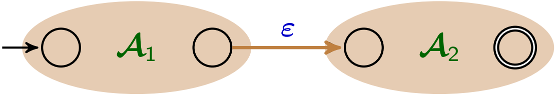




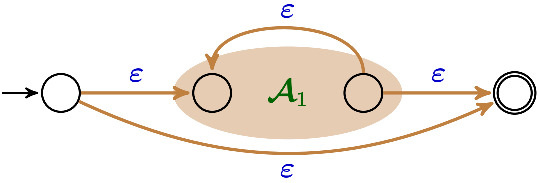
- **Union**



- **Concaténation**



- **Étoile**



État des lieux

Les trois objets introduits :

- AFD,
- AFN,
- AFN avec ε-transitions

ont la même capacité de calcul.

Autrement dit, les classes de langages reconnaissables par AFD, AFN avec ou sans ε-transitions sont identiques. Tous ces automates caractérisent l'ensemble des langages réguliers.

De plus, le théorème de Kleene assure l'équivalence entre expressions régulières et automates finis.

Bilan

AFD, AFN avec ou sans ε-transitions et expressions régulières définissent la même classe de langages.

D'expression régulière vers automate fini

L'algorithme de Glushkov

Entrée : une expression régulière

$$(\boldsymbol{b} + \boldsymbol{ab})^*(\varepsilon + \boldsymbol{ab})$$

Étape 1. Linéariser l'expression régulière.
On remplace toutes les lettres de l'expression par des symboles distincts (x_i pour la lettre en i-ème position)

$$(\boldsymbol{x}_1 + \boldsymbol{x}_2\boldsymbol{x}_3)^*(\varepsilon + \boldsymbol{x}_4\boldsymbol{x}_5)$$

Étape 2. Déterminer

- **Premier** : l'ensemble des symboles pouvant commencer un mot
- **Dernier** : l'ensemble des symboles pouvant terminer un mot
- **Suivant** : pour tout symbole $\boldsymbol{x_i}$, l'ensemble des symboles pouvant suivre $\boldsymbol{x_i}$

l'expression régulière linéarisée : $(\boldsymbol{x}_1 + \boldsymbol{x}_2\boldsymbol{x}_3)^*(\varepsilon + \boldsymbol{x}_4\boldsymbol{x}_5)$

Premier $= \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_4\}$

Dernier $= \{\boldsymbol{x}_1, \boldsymbol{x}_3, \boldsymbol{x}_5\}$

	Suivant
$\boldsymbol{x}_1$	$\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_4$
$\boldsymbol{x}_2$	$\boldsymbol{x}_3$
$\boldsymbol{x}_3$	$\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_4$
$\boldsymbol{x}_4$	$\boldsymbol{x}_5$
$\boldsymbol{x}_5$	

Un mot $w = a_1 \cdots a_n$ appartient au langage décrit par l'expression linéarisée si et seulement si

- $a_1 \in$ Premier,
- $a_{i+1} \in$ Suivant(a_i) pour tout $i = 1, \cdots, n-1$ et
- $a_n \in$ Dernier.

$x_2x_3x_1x_1$ est un mot du langage : $x_2 \in$ Premier, $x_3 \in$ Suivant(x_2), $x_1 \in$ Suivant(x_3), $x_1 \in$ Suivant(x_1), $x_1 \in$ Dernier.
$x_4x_5x_1$ n'est pas un mot du langage : $x_1 \notin$ Suivant(x_5).

Étape 3. Construction de l'AFN

L'ensemble des états :

- un état initial 0
- un état $\boldsymbol{i}$ par symbole $\boldsymbol{x_i}$
  (l'automate sauvegarde le dernier symbole $\boldsymbol{x_i}$ lu via l'état $\boldsymbol{i}$)

L'ensemble des états finaux :

- l'état initial 0 **si** ε appartient au langage
- un état $\boldsymbol{i}$ pour tout $\boldsymbol{x_i}$ appartenant à **Dernier**

La fonction de transition :

- une transition de l'état initial 0 vers l'état $\boldsymbol{i}$ pour tout $\boldsymbol{x_i}$ appartenant à **Premier** et étiquetée par la lettre correspondant à $\boldsymbol{x_i}$
- une transition de l'état $\boldsymbol{i}$ vers l'état $\boldsymbol{j}$ pour tout $\boldsymbol{x_j}$ appartenant à **Suivant**($\boldsymbol{x_i}$) et étiquetée par la lettre correspondant à $\boldsymbol{x_j}$

$$(b + ab)^*(\varepsilon + ab)$$

$(x_1 + x_2x_3)^*(\varepsilon + x_4x_5)$

Premier $= \{x_1, x_2, x_4\}$

Dernier $= \{x_1, x_3, x_5\}$

	Suivant
x_1	x_1, x_2, x_4
x_2	x_3
x_3	x_1, x_2, x_4
x_4	x_5
x_5	

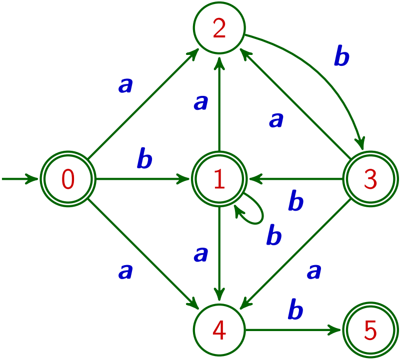
0 l'état initial

$Q = \{0, 1, 2, 3, 4, 5\}$

ε appartient au langage

$\Rightarrow$ l'état initial 0 est final

$F = \{0, 1, 3, 5\}$



La **taille** d'une expression régulière, notée $|\boldsymbol{r}|$, est son nombre de caractères $\boldsymbol{\varepsilon}$, $\boldsymbol{a}$ (lettre de l'alphabet), $+$ et $*$ (les parenthèses et la concaténation ne sont pas pris en compte).

$$|(b + ab)^*(\varepsilon + ab)| = 9$$

Le nombre d'états de l'automate est borné par $|\boldsymbol{r}| + 1$.

Le nombre de transitions est quadratique dans le pire cas, mais linéaire dans le cas moyen (pour la distribution uniforme).

$\rightsquigarrow$ Le temps de construction de l'AFN est

- en $\boldsymbol{O}(|\boldsymbol{r}|^2)$ dans le pire cas
- en $\boldsymbol{O}(|\boldsymbol{r}|)$ dans le cas moyen.

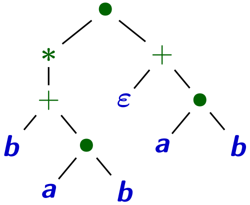
D'expression régulière vers automate fini

Une autre variante : l'algorithme de Thompson

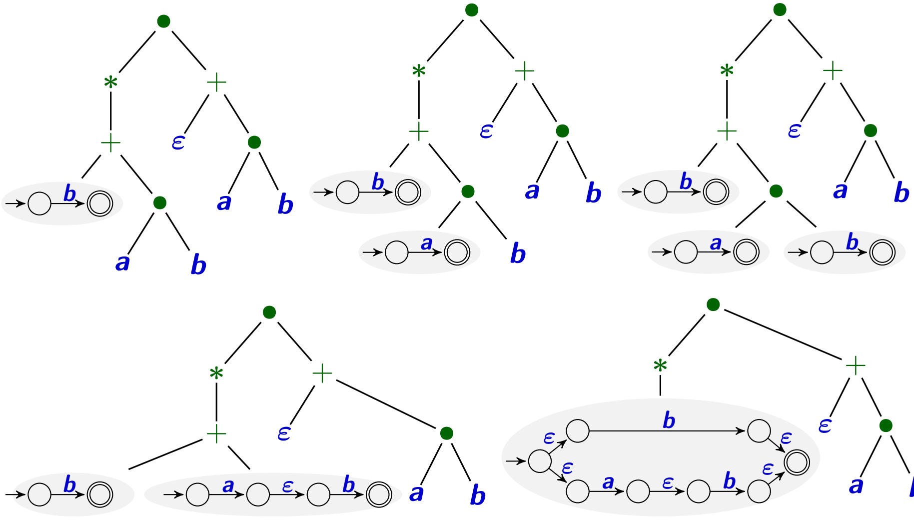
Entrée : une expression régulière

$$(\boldsymbol{b} + \boldsymbol{ab})^*(\varepsilon + \boldsymbol{ab})$$

Étape 1. Construction de l'arbre syntaxique associé à l'expression.



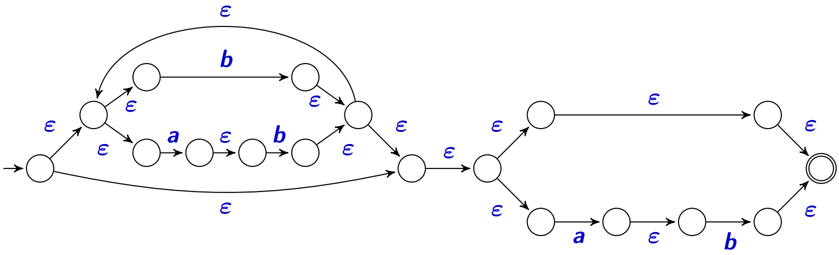
Étape 2. Construction d'un AFN avec ε-transitions via un parcours en profondeur postfixé de l'arbre. Chaque fragment a un seul état initial, un seul état final et aucun arc arrivant dans l'état initial ou partant de l'état final.



. . .

D'expression régulière vers automate fini

Complexité de l'algorithme de Thompson



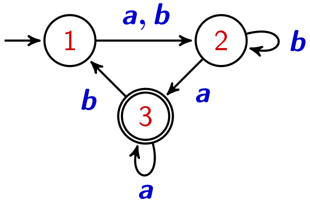
Coût de l'algorithme de Thompson :

- Construction de l'arbre de syntaxe de taille $O(|r|)$ en $O(|r|)$ étapes
- Construction de l'AFN via un parcours postfixé de l'arbre de taille $O(|r|)$ en $O(|r|)$ étapes

D'automate fini vers expression régulière

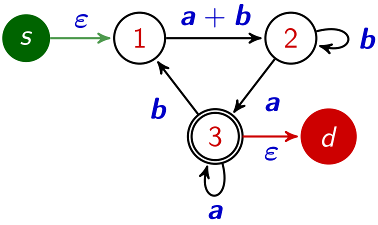
L'algo naïf

On part du graphe de transition de l'automate.

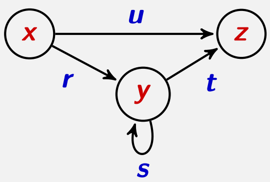


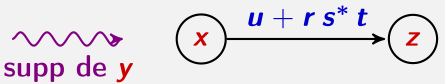
On ajoute

- une source avec une transition étiquetée par ε sur tous les états initiaux.
- une destination avec des transitions étiquetées par ε des états d'acceptation vers cette destination

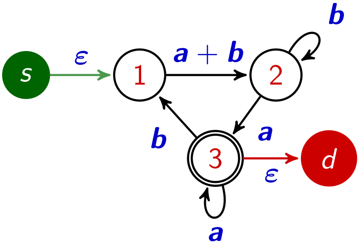


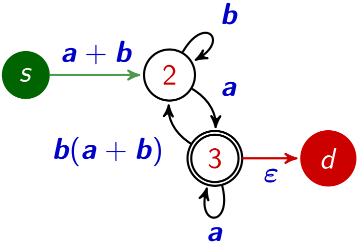
On supprime un à un les états de la façon suivante :

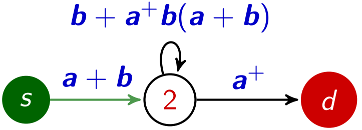


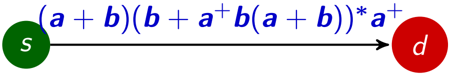


invariant : les transitions sont étiquetées par des expressions régulières



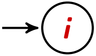


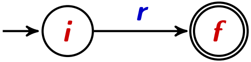




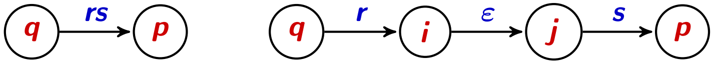
D'expression régulière vers automate fini

L'algorithme de Thompson construit par induction un AFN avec ε-transitions reconnaissant le langage décrit par une expression régulière.

Si $r = \emptyset$, l'automate correspondant est 

Si $r \neq \emptyset$, on part du graphe 

et on applique les transformations suivantes jusqu'à ce que les arcs soient étiquetés soit par ε, soit par une lettre de Σ.

- 

-