

# Langages, Compilation, Automates.

## Partie 3: expressions régulières, conversion en AFD et Lemme de l'étoile

Florian Bridoux

Polytech Nice Sophia

2022-2023

- 1 Expressions régulières
- 2 Conversion en AFD
- 3 Langages non réguliers et lemme de l'étoile

1 Expressions régulières

2 Conversion en AFD

3 Langages non réguliers et lemme de l'étoile

## Definition (Expression régulière)

Les **expressions régulières** sur l'alphabet  $\Sigma$  sont définies récursivement :

- ① ( $\emptyset$  est une expression régulière qui décrit l'ensemble vide;)
- ② ( $\epsilon$  est une expression régulière qui décrit l'ensemble réduit au mot vide  $\{\epsilon\}$  ;)
- ③ pour toute lettre  $a \in \Sigma$ ,  $a$  est une expression régulière qui décrit l'ensemble  $\{a\}$ .
- Si  $r$  et  $s$  sont deux expressions régulières, alors:
  - ①  $r + s$  est une expression régulière qui décrit le langage  $L(r) \cup L(s)$ ,
  - ②  $rs$  est une expression régulière qui décrit le langage  $L(r)L(s) = \{uv \mid u \in L(r) \text{ et } v \in L(s)\}$ ;
  - ③  $r^*$  est une expression régulière qui décrit  $L(r)^* = \{w_1 \dots w_n \mid w_i \in L(r) \text{ et } n \in \mathbb{N}\}$ .
  - ④  $(r)$  est une expression régulière qui décrit  $L(r)$ .

# Expression régulière

Ordre de priorité des opérateurs :

l'étoile  $>$  la concaténation  $>$  l'union.

Par exemple,  $b + ab^*$  se lit  $b + (a(b^*))$ .

description en français	expression régulière	langage
taille multiple de 2	$((a + b)(a + b))^*$	$(\{a, b\}^2)^*$
se termine par $a$	$(a + b)^*a$	$\{a, b\}^* \{a\}$
avec le facteur $bb$	$(a + b)^*bb(a + b)^*$	$\{a, b\}^* \{bb\} \{a, b\}^*$
sans le facteur $bb$	$(\epsilon + b)(a + ab)^*$	$\{\epsilon, b\} \{a, ab\}^*$

## Remarque:

Une expression régulière décrit un unique langage mais un langage peut-être décrit par plusieurs expressions régulières.

Exemple:  $(a + b)(a + b) = aa + ab + ba + bb$ .

## Théorème de Kleene

Les expressions régulières décrivent exactement la même famille de langages que les AFD et les AFI (les langages réguliers).

On va le prouver en donnant:

- un "algorithme" qui transforme n'importe quelle expression régulière  $r$  en un AFD  $A$  tel que  $L(r) = L(A)$ ;
- un "algorithme" qui transforme n'importe quel AFD  $A$  en une expression régulière  $r$  tel que  $L(A) = L(r)$ ;

# Table des matières

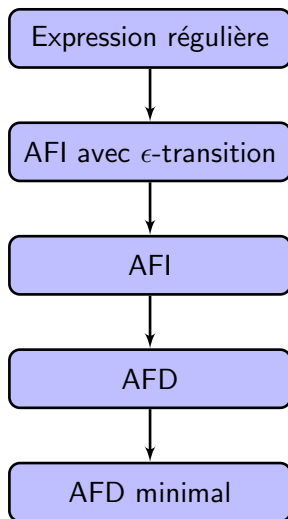
1 Expressions régulières

2 Conversion en AFD

3 Langages non réguliers et lemme de l'étoile

# Transformer une expression régulière en AFD

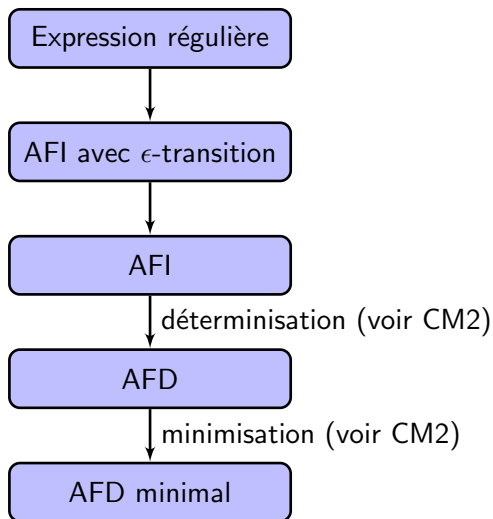
Étapes de la transformation:

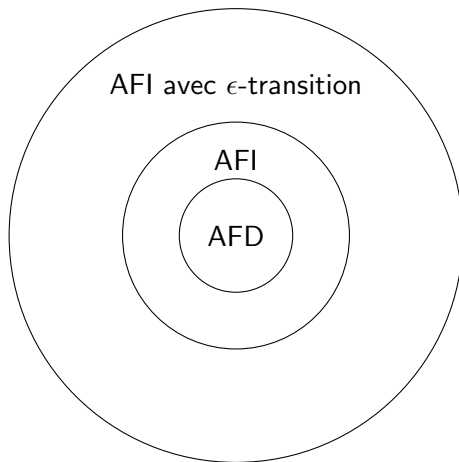




# Transformer une expression régulière en AFD

Étapes de la transformation:

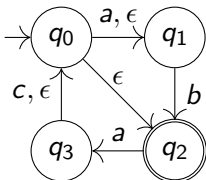




Les AFI avec  $\epsilon$ -transition sont une généralisation des AFI. On va voir qu'ils reconnaissent exactement la même famille de langages que les AFI et les AFD.

# AFI avec $\epsilon$ -transitions

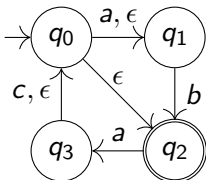
Les  $\epsilon$ -transitions permettent de passer d'un état à l'autre sans "consommer" de lettres.



Arbre des transitions possibles sur le mot *acba*:

# AFI avec $\epsilon$ -transitions

Les  $\epsilon$ -transitions permettent de passer d'un état à l'autre sans "consommer" de lettres.

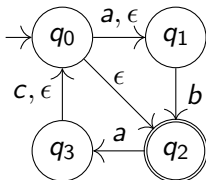


Arbre des transitions possibles sur le mot *acba*:

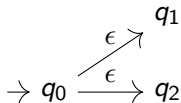
$\rightarrow q_0$

# AFI avec $\epsilon$ -transitions

Les  $\epsilon$ -transitions permettent de passer d'un état à l'autre sans "consommer" de lettres.

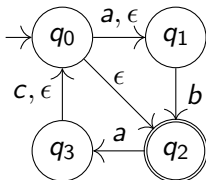


Arbre des transitions possibles sur le mot *acba*:

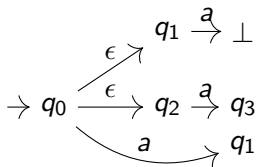


# AFI avec $\epsilon$ -transitions

Les  $\epsilon$ -transitions permettent de passer d'un état à l'autre sans "consommer" de lettres.

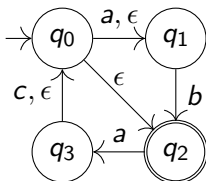


Arbre des transitions possibles sur le mot *acba*:

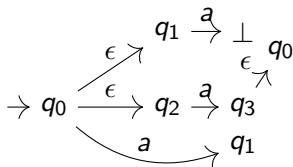


# AFI avec $\epsilon$ -transitions

Les  $\epsilon$ -transitions permettent de passer d'un état à l'autre sans "consommer" de lettres.

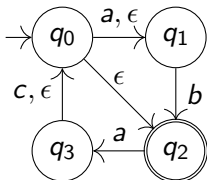


Arbre des transitions possibles sur le mot *acba*:

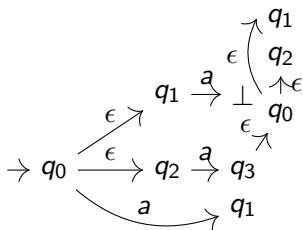


# AFI avec $\epsilon$ -transitions

Les  $\epsilon$ -transitions permettent de passer d'un état à l'autre sans "consommer" de lettres.



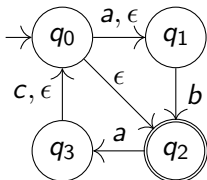
Arbre des transitions possibles sur le mot *acba*:



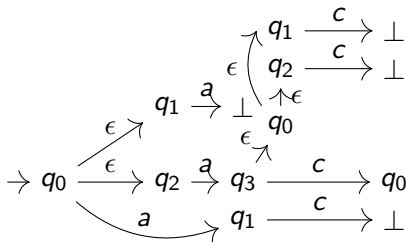


# AFI avec $\epsilon$ -transitions

Les  $\epsilon$ -transitions permettent de passer d'un état à l'autre sans "consommer" de lettres.

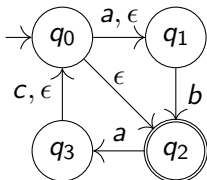


Arbre des transitions possibles sur le mot *acba*:

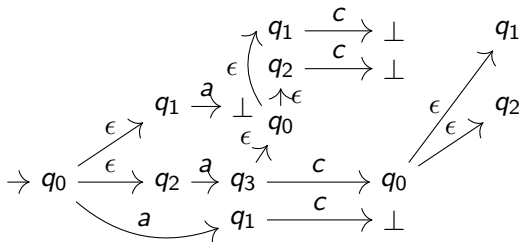


## AFI avec $\epsilon$ -transitions

Les  $\epsilon$ -transitions permettent de passer d'un état à l'autre sans "consommer" de lettres.

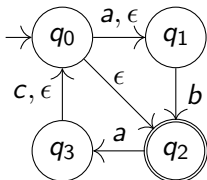


Arbre des transitions possibles sur le mot *acba*:

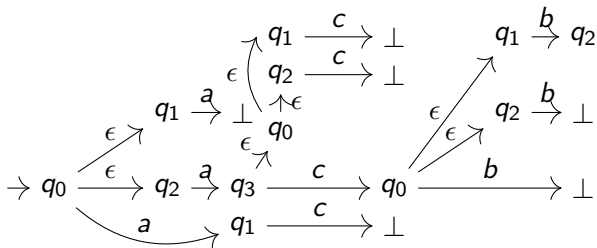


# AFI avec $\epsilon$ -transitions

Les  $\epsilon$ -transitions permettent de passer d'un état à l'autre sans "consommer" de lettres.

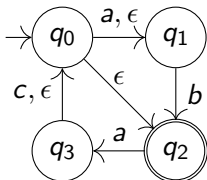


Arbre des transitions possibles sur le mot *acba*:

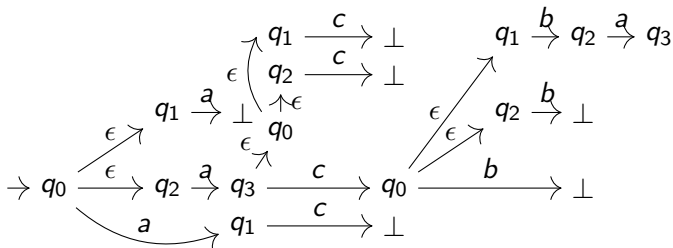


## AFI avec $\epsilon$ -transitions

Les  $\epsilon$ -transitions permettent de passer d'un état à l'autre sans "consommer" de lettres.

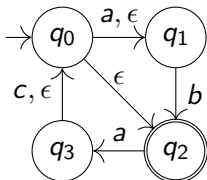


Arbre des transitions possibles sur le mot *acba*:

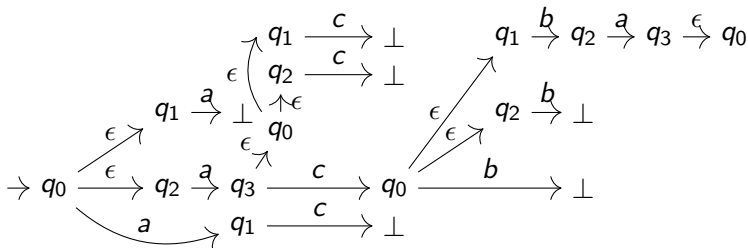


## AFI avec $\epsilon$ -transitions

Les  $\epsilon$ -transitions permettent de passer d'un état à l'autre sans "consommer" de lettres.

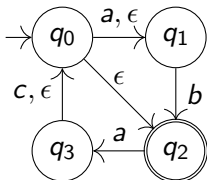


Arbre des transitions possibles sur le mot *acba*:

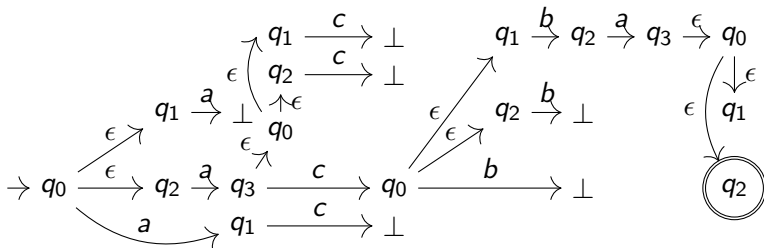


# AFI avec $\epsilon$ -transitions

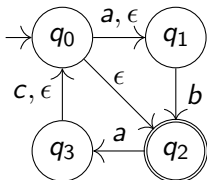
Les  $\epsilon$ -transitions permettent de passer d'un état à l'autre sans "consommer" de lettres.



Arbre des transitions possibles sur le mot *acba*:

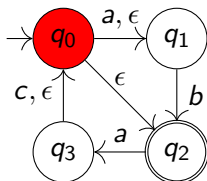


# Algorithme pour retirer les $\epsilon$ -transitions



Tant que j'ai des  $\epsilon$ -transitions:

# Algorithme pour retirer les $\epsilon$ -transitions

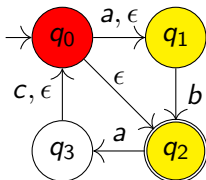


Tant que j'ai des  $\epsilon$ -transitions:

- Je choisis  $q_i$  qui a des  $\epsilon$ -transitions sortantes



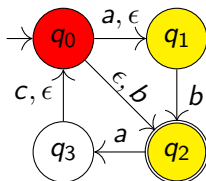
# Algorithme pour retirer les $\epsilon$ -transitions



Tant que j'ai des  $\epsilon$ -transitions:

- Je choisis  $q_i$  qui a des  $\epsilon$ -transitions sortantes
- Si, avec des  $\epsilon$ -transitions je peux atteindre  $q_j$ , alors:
  - J'ajoute toutes les (non  $\epsilon$ ) transitions sortantes de  $q_j$  sur  $q_i$ .

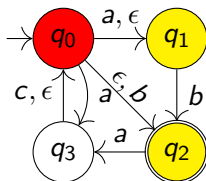
# Algorithme pour retirer les $\epsilon$ -transitions



Tant que j'ai des  $\epsilon$ -transitions:

- Je choisis  $q_i$  qui a des  $\epsilon$ -transitions sortantes
- Si, avec des  $\epsilon$ -transitions je peux atteindre  $q_j$ , alors:
  - J'ajoute toutes les (non  $\epsilon$ ) transitions sortantes de  $q_j$  sur  $q_i$ .

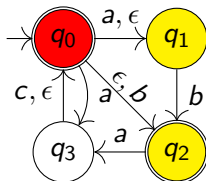
# Algorithme pour retirer les $\epsilon$ -transitions



Tant que j'ai des  $\epsilon$ -transitions:

- Je choisis  $q_i$  qui a des  $\epsilon$ -transitions sortantes
- Si, avec des  $\epsilon$ -transitions je peux atteindre  $q_j$ , alors:
  - J'ajoute toutes les (non  $\epsilon$ ) transitions sortantes de  $q_j$  sur  $q_i$ .

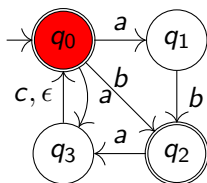
# Algorithme pour retirer les $\epsilon$ -transitions



Tant que j'ai des  $\epsilon$ -transitions:

- Je choisis  $q_i$  qui a des  $\epsilon$ -transitions sortantes
- Si, avec des  $\epsilon$ -transitions je peux atteindre  $q_j$ , alors:
  - J'ajoute toutes les (non  $\epsilon$ ) transitions sortantes de  $q_j$  sur  $q_i$ .
  - Si  $q_j$  est acceptant alors je rend  $q_i$  acceptant également.

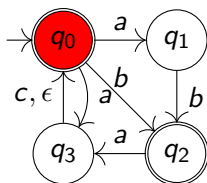
# Algorithme pour retirer les $\epsilon$ -transitions



Tant que j'ai des  $\epsilon$ -transitions:

- Je choisis  $q_i$  qui a des  $\epsilon$ -transitions sortantes
- Si, avec des  $\epsilon$ -transitions je peux atteindre  $q_j$ , alors:
  - J'ajoute toutes les (non  $\epsilon$ ) transitions sortantes de  $q_j$  sur  $q_i$ .
  - Si  $q_j$  est acceptant alors je rend  $q_i$  acceptant également.
- Je supprime les  $\epsilon$ -transitions sortantes de  $q_i$ .
- (Je supprime les transitions en doublons)

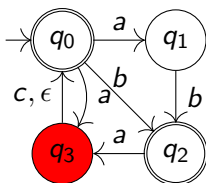
# Algorithme pour retirer les $\epsilon$ -transitions



Tant que j'ai des  $\epsilon$ -transitions:

- Je choisis  $q_i$  qui a des  $\epsilon$ -transitions sortantes
- Si, avec des  $\epsilon$ -transitions je peux atteindre  $q_j$ , alors:
  - J'ajoute toutes les (non  $\epsilon$ ) transitions sortantes de  $q_j$  sur  $q_i$ .
  - Si  $q_j$  est acceptant alors je rend  $q_i$  acceptant également.
- Je supprime les  $\epsilon$ -transitions sortantes de  $q_i$ .
- (Je supprime les transitions en doublons)

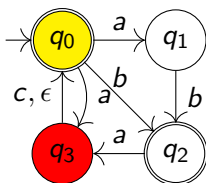
# Algorithme pour retirer les $\epsilon$ -transitions



Tant que j'ai des  $\epsilon$ -transitions:

- Je choisis  $q_i$  qui a des  $\epsilon$ -transitions sortantes
- Si, avec des  $\epsilon$ -transitions je peux atteindre  $q_j$ , alors:
  - J'ajoute toutes les (non  $\epsilon$ ) transitions sortantes de  $q_j$  sur  $q_i$ .
  - Si  $q_j$  est acceptant alors je rend  $q_i$  acceptant également.
- Je supprime les  $\epsilon$ -transitions sortantes de  $q_i$ .
- (Je supprime les transitions en doublons)

# Algorithme pour retirer les $\epsilon$ -transitions

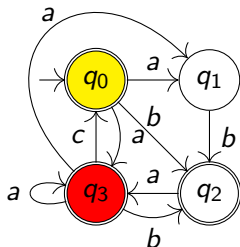


Tant que j'ai des  $\epsilon$ -transitions:

- Je choisis  $q_i$  qui a des  $\epsilon$ -transitions sortantes
- Si, avec des  $\epsilon$ -transitions je peux atteindre  $q_j$ , alors:
  - J'ajoute toutes les (non  $\epsilon$ ) transitions sortantes de  $q_j$  sur  $q_i$ .
  - Si  $q_j$  est acceptant alors je rend  $q_i$  acceptant également.
- Je supprime les  $\epsilon$ -transitions sortantes de  $q_i$ .
- (Je supprime les transitions en doublons)



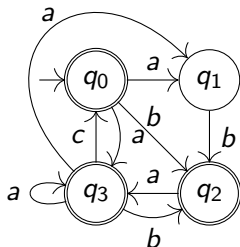
# Algorithme pour retirer les $\epsilon$ -transitions



Tant que j'ai des  $\epsilon$ -transitions:

- Je choisis  $q_i$  qui a des  $\epsilon$ -transitions sortantes
- Si, avec des  $\epsilon$ -transitions je peux atteindre  $q_j$ , alors:
  - J'ajoute toutes les (non  $\epsilon$ ) transitions sortantes de  $q_j$  sur  $q_i$ .
  - Si  $q_j$  est acceptant alors je rend  $q_i$  acceptant également.
- Je supprime les  $\epsilon$ -transitions sortantes de  $q_i$ .
- (Je supprime les transitions en doublons)

# Algorithme pour retirer les $\epsilon$ -transitions

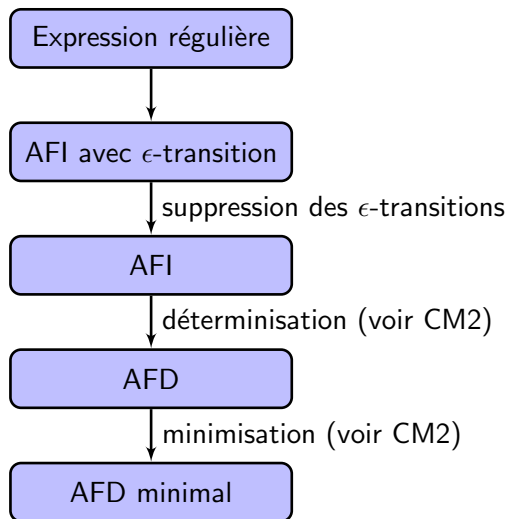


Tant que j'ai des  $\epsilon$ -transitions:



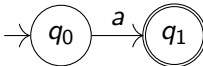
- Je choisis  $q_i$  qui a des  $\epsilon$ -transitions sortantes
- Si, avec des  $\epsilon$ -transitions je peux atteindre  $q_j$ , alors:
  - J'ajoute toutes les (non  $\epsilon$ ) transitions sortantes de  $q_j$  sur  $q_i$ .
  - Si  $q_j$  est acceptant alors je rend  $q_i$  acceptant également.
- Je supprime les  $\epsilon$ -transitions sortantes de  $q_i$ .
- (Je supprime les transitions en doublons)

# Transformer une expression régulière en AFD

Étapes de la transformation:

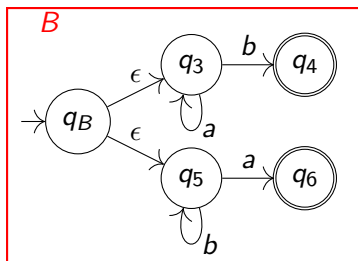
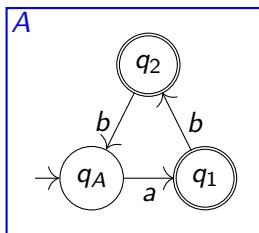


# Transformer une expression régulière: brique de base

- Langage vide: 
- Langage  $\{\epsilon\}$ : 
- Langage  $\{a\}$ : 

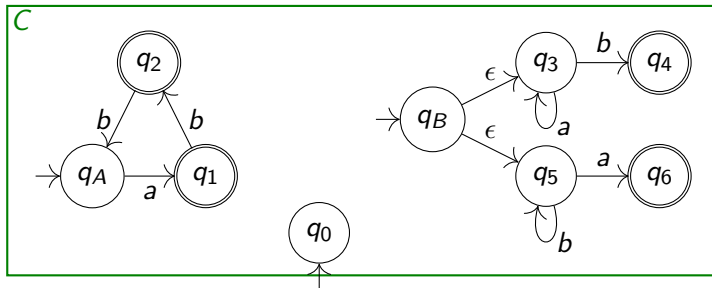
# Transformer une expression régulière: union

Supposons que  $s$  et  $r$  sont deux expressions régulières équivalentes à  $A$  et  $B$  (avec  $q_A$  et  $q_B$  leurs deux états initiaux). Comment former  $C$  équivalent à  $s + r$ ?



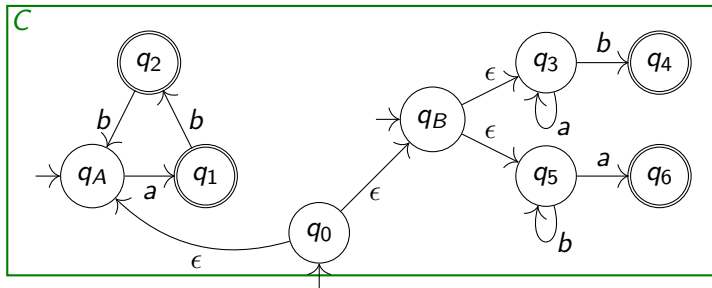
# Transformer une expression régulière: union

Supposons que  $s$  et  $r$  sont deux expressions régulières équivalentes à  $A$  et  $B$  (avec  $q_A$  et  $q_B$  leurs deux états initiaux). Comment former  $C$  équivalent à  $s + r$ ?



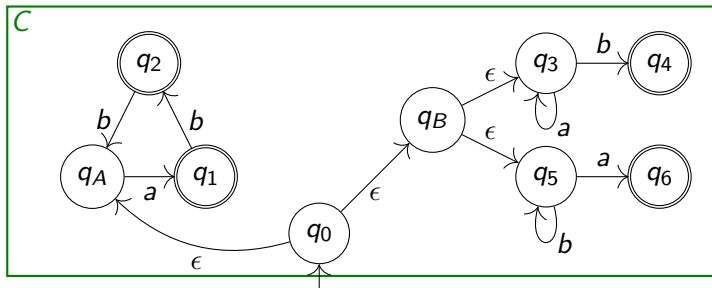
# Transformer une expression régulière: union

Supposons que  $s$  et  $r$  sont deux expressions régulières équivalentes à  $A$  et  $B$  (avec  $q_A$  et  $q_B$  leurs deux états initiaux). Comment former  $C$  équivalent à  $s + r$ ?



# Transformer une expression régulière: union

Supposons que  $s$  et  $r$  sont deux expressions régulières équivalentes à  $A$  et  $B$  (avec  $q_A$  et  $q_B$  leurs deux états initiaux). Comment former  $C$  équivalent à  $s + r$ ?

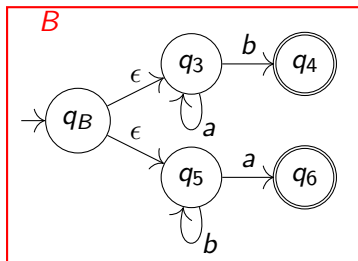
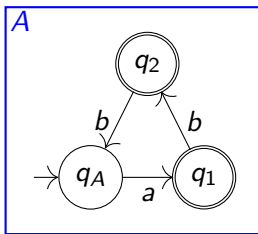


- On ajoute un nouvel état initial  $q_0$ .
- On ajoute deux transitions  $q_0 \xrightarrow{\epsilon} q_A$  et  $q_0 \xrightarrow{\epsilon} q_B$ .
- Les états  $q_A$  et  $q_B$  ne sont plus initiaux.



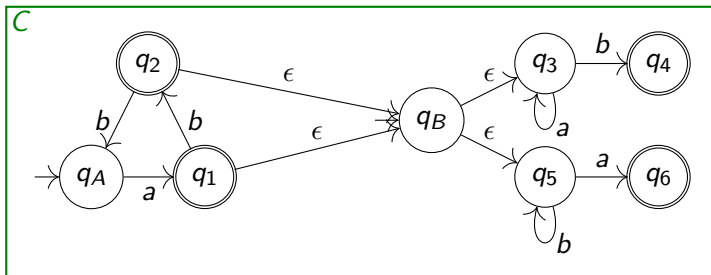
# Transformer une expression régulière: concaténation

Supposons que  $s$  et  $r$  sont deux expressions régulières équivalentes à  $A$  et  $B$  (avec  $q_A$  et  $q_B$  leurs deux états initiaux). Comment former  $C$  équivalent à  $sr$ ?



# Transformer une expression régulière: concaténation

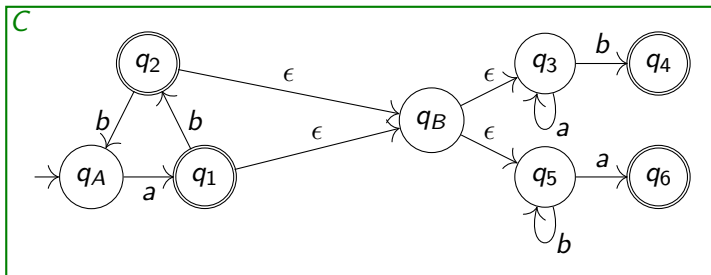
Supposons que  $s$  et  $r$  sont deux expressions régulières équivalentes à  $A$  et  $B$  (avec  $q_A$  et  $q_B$  leurs deux états initiaux). Comment former  $C$  équivalent à  $sr$ ?



- On ajoute des transitions  $q_i \xrightarrow{\epsilon} q_A$  pour tout état acceptant  $q_i$  de  $A$ .

# Transformer une expression régulière: concaténation

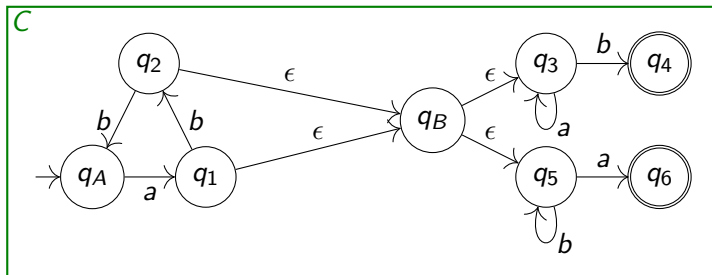
Supposons que  $s$  et  $r$  sont deux expressions régulières équivalentes à  $A$  et  $B$  (avec  $q_A$  et  $q_B$  leurs deux états initiaux). Comment former  $C$  équivalent à  $sr$ ?



- On ajoute des transitions  $q_i \xrightarrow{\epsilon} q_A$  pour tout état acceptant  $q_i$  de  $A$ .
- $q_B$  n'est plus initial.

# Transformer une expression régulière: concaténation

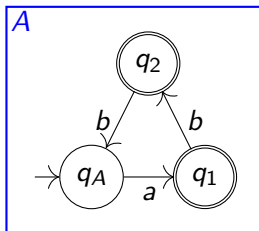
Supposons que  $s$  et  $r$  sont deux expressions régulières équivalentes à  $A$  et  $B$  (avec  $q_A$  et  $q_B$  leurs deux états initiaux). Comment former  $C$  équivalent à  $sr$ ?



- On ajoute des transitions  $q_i \xrightarrow{\epsilon} q_A$  pour tout état acceptant  $q_i$  de  $A$ .
- $q_B$  n'est plus initial.
- Les états acceptants de  $q_A$  ne sont plus acceptants.

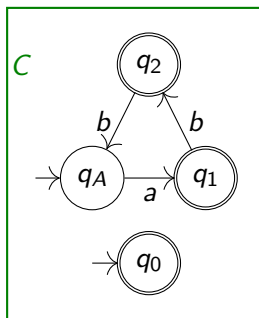
# Transformer une expression régulière: étoile

Supposons que  $s$  est une expression régulière équivalente à  $A$  (avec  $q_A$  sont état initial). Comment former  $C$  équivalent à  $s^*$ ?



# Transformer une expression régulière: étoile

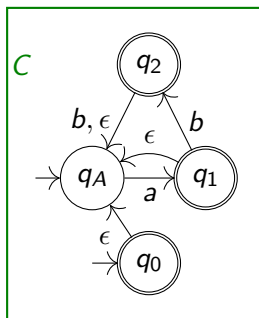
Supposons que  $s$  est une expression régulière équivalente à  $A$  (avec  $q_A$  sont état initial). Comment former  $C$  équivalent à  $s^*$ ?



- On ajoute un nouvel état initial acceptant  $q_0$ .

# Transformer une expression régulière: étoile

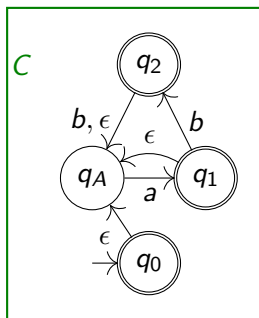
Supposons que  $s$  est une expression régulière équivalente à  $A$  (avec  $q_A$  sont état initial). Comment former  $C$  équivalent à  $s^*$ ?



- On ajoute un nouvel état initial acceptant  $q_0$ .
- On ajoute une transition  $q_0 \xrightarrow{\epsilon} q_A$  et une transition  $q_i \xrightarrow{\epsilon} q_A$  pour chaque état acceptant  $q_i$ .

# Transformer une expression régulière: étoile

Supposons que  $s$  est une expression régulière équivalente à  $A$  (avec  $q_A$  sont état initial). Comment former  $C$  équivalent à  $s^*$ ?


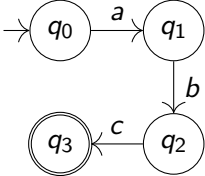
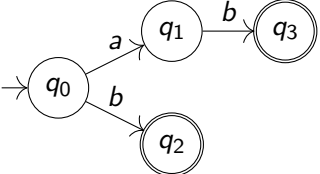


- On ajoute un nouvel état initial acceptant  $q_0$ .
- On ajoute une transition  $q_0 \xrightarrow{\epsilon} q_A$  et une transition  $q_i \xrightarrow{\epsilon} q_A$  pour chaque état acceptant  $q_i$ .
- L'état  $q_i$  n'est plus initial.



# Transformer une expression régulière: cas faciles

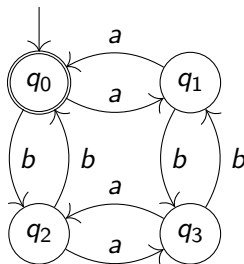
En pratique, on peut directement représenter certaines expressions basiques par un AFD (ou même un AFI sans  $\epsilon$ -transition).

Expression régulière	Langage	AFD
$a^*$	$\{a^n \mid n \in \mathbb{N}\}$	 <p>Diagram of an AFD for the regular expression <math>a^*</math>. It consists of a single state <math>q_0</math>, which is both the start state (indicated by an incoming arrow) and the final state (indicated by a double circle). There is a self-loop transition on <math>q_0</math> labeled with the symbol <math>a</math>.</p>
$abc$	$\{abc\}$	 <p>Diagram of an AFD for the regular expression <math>abc</math>. It consists of four states: <math>q_0</math> (start state), <math>q_1</math>, <math>q_2</math>, and <math>q_3</math> (final state). Transitions are: <math>q_0 \xrightarrow{a} q_1</math>, <math>q_1 \xrightarrow{b} q_2</math>, and <math>q_2 \xrightarrow{c} q_3</math>.</p>
$ab + b$	$\{ab, b\}$	 <p>Diagram of an AFD for the regular expression <math>ab + b</math>. It consists of four states: <math>q_0</math> (start state), <math>q_1</math>, <math>q_2</math> (final state), and <math>q_3</math> (final state). Transitions are: <math>q_0 \xrightarrow{a} q_1</math>, <math>q_1 \xrightarrow{b} q_3</math>, and <math>q_0 \xrightarrow{b} q_2</math>.</p>

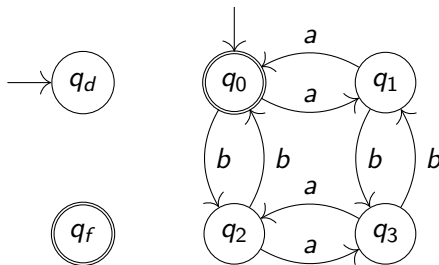
# Transformer un AFD en expression régulière

Dans l'autre sens, on peut transformer un AFD en expression régulière avec l'algorithme suivant.

# Transformer un AFD en expression régulière

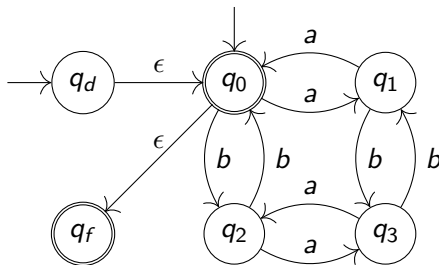


# Transformer un AFD en expression régulière



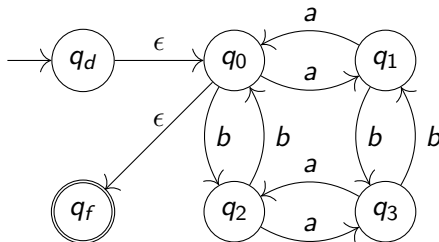
- On ajoute un nouvel initial  $q_d$  et un nouvel état acceptant  $q_f$ .

# Transformer un AFD en expression régulière



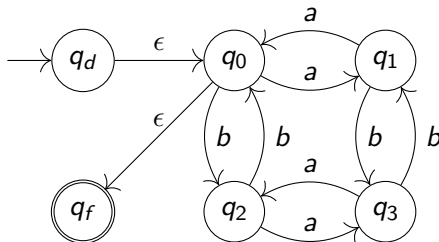
- On ajoute un nouvel initial  $q_d$  et un nouvel état acceptant  $q_f$ .
- On ajoute des  $\epsilon$ -transitions de  $q_d$  vers l'ancien état initial et depuis les anciens états acceptants vers  $q_f$ .

# Transformer un AFD en expression régulière



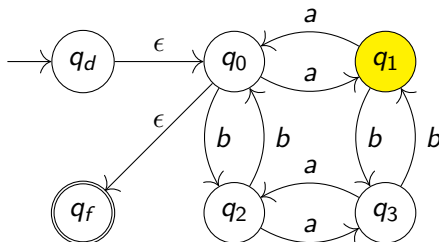
- On ajoute un nouvel initial  $q_d$  et un nouvel état acceptant  $q_f$ .
- On ajoute des  $\epsilon$ -transitions de  $q_d$  vers l'ancien état initial et depuis les anciens états acceptants vers  $q_f$ .
- L'ancien état initial n'est plus initial, les anciens états acceptants ne sont plus acceptants.

# Transformer un AFD en expression régulière



- On ajoute un nouvel initial  $q_d$  et un nouvel état acceptant  $q_f$ .
- On ajoute des  $\epsilon$ -transitions de  $q_d$  vers l'ancien état initial et depuis les anciens états acceptants vers  $q_f$ .
- L'ancien état initial n'est plus initial, les anciens états acceptants ne sont plus acceptants.
- On remplace  $q_i \xrightarrow{a,b,c,\dots} q_j$  par  $q_i \xrightarrow{a+b+c+\dots} q_j$ .

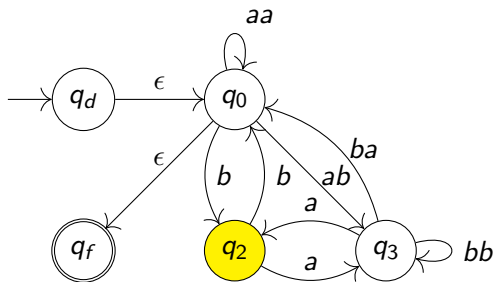
# Transformer un AFD en expression régulière



- On supprime un par un les états  $q_j$  autre que  $q_d$  et  $q_f$ .
  - Si  $q_j$  n'a pas de boucle, on remplace chaque pair de transition  $q_i \xrightarrow{r} q_j \xrightarrow{s} q_k$  par une transition  $q_i \xrightarrow{rs} q_k$ .

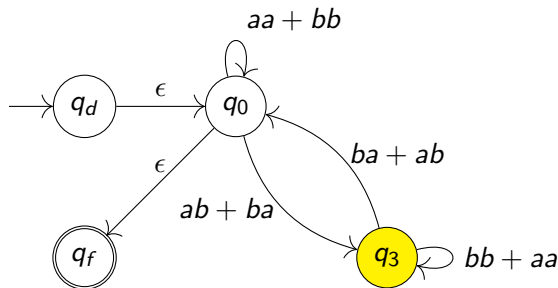


# Transformer un AFD en expression régulière



- On supprime un par un les états  $q_j$  autre que  $q_d$  et  $q_f$ .
  - Si  $q_j$  n'a pas de boucle, on remplace chaque pair de transition  $q_i \xrightarrow{r} q_j \xrightarrow{s} q_k$  par une transition  $q_i \xrightarrow{rs} q_k$ .

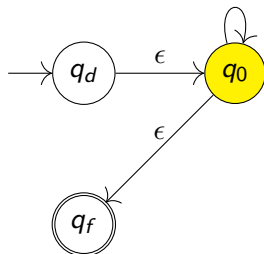
# Transformer un AFD en expression régulière



- On supprime un par un les états  $q_j$  autre que  $q_d$  et  $q_f$ .
  - Si  $q_j$  n'a pas de boucle, on remplace chaque pair de transition  $q_i \xrightarrow{r} q_j \xrightarrow{s} q_k$  par une transition  $q_i \xrightarrow{rs} q_k$ .
  - Si  $q_j$  a une boucle  $q_j \xrightarrow{t} q_j$ , on va remplacer chaque pair de transitions  $q_i \xrightarrow{r} q_j \xrightarrow{s} q_k$  par une transition  $q_i \xrightarrow{rt^*s} q_k$ .

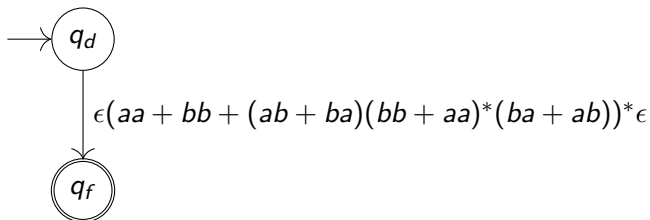
# Transformer un AFD en expression régulière

$$aa + bb + (ab + ba)(bb + aa)^*(ba + ab)$$



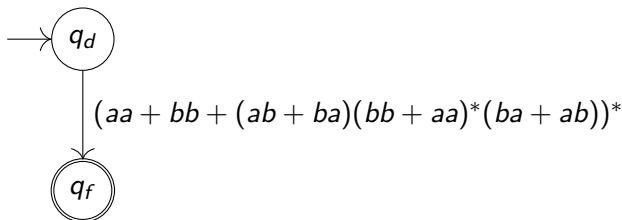
- On supprime un par un les états  $q_j$  autre que  $q_d$  et  $q_f$ .
  - Si  $q_j$  n'a pas de boucle, on remplace chaque pair de transition  $q_i \xrightarrow{r} q_j \xrightarrow{s} q_k$  par une transition  $q_i \xrightarrow{rs} q_k$ .
  - Si  $q_j$  a une boucle  $q_j \xrightarrow{t} q_j$ , on va remplacer chaque pair de transitions  $q_i \xrightarrow{r} q_j \xrightarrow{s} q_k$  par une transition  $q_i \xrightarrow{rt^*s} q_k$ .

# Transformer un AFD en expression régulière



- On supprime un par un les états  $q_j$  autre que  $q_d$  et  $q_f$ .
  - Si  $q_j$  n'a pas de boucle, on remplace chaque pair de transition  $q_i \xrightarrow{r} q_j \xrightarrow{s} q_k$  par une transition  $q_i \xrightarrow{rs} q_k$ .
  - Si  $q_j$  a une boucle  $q_j \xrightarrow{t} q_j$ , on va remplacer chaque pair de transitions  $q_i \xrightarrow{r} q_j \xrightarrow{s} q_k$  par une transition  $q_i \xrightarrow{rt^*s} q_k$ .

# Transformer un AFD en expression régulière



- On supprime un par un les états  $q_j$  autre que  $q_d$  et  $q_f$ .
  - Si  $q_j$  n'a pas de boucle, on remplace chaque pair de transition  $q_i \xrightarrow{r} q_j \xrightarrow{s} q_k$  par une transition  $q_i \xrightarrow{rs} q_k$ .
  - Si  $q_j$  a une boucle  $q_j \xrightarrow{t} q_j$ , on va remplacer chaque pair de transitions  $q_i \xrightarrow{r} q_j \xrightarrow{s} q_k$  par une transition  $q_i \xrightarrow{rt^*s} q_k$ .
- Quand on a supprimé tous les états, il ne reste qu'une transition  $q_d \xrightarrow{r} q_f$ .  $r$  est notre expression régulière.

# Les expressions régulières et les langages réguliers

## Remarque:

L'expression régulière obtenue n'est pas forcément la plus élégante ou la plus petite possible et elle change selon l'ordre de suppression.

## Remarque:

Trouver une plus petite expression régulière correspondant à un langage donné est un problème très difficile: il est PSPACE-complet.

## Remarque:

On peut vérifier que deux expressions régulières  $r$  et  $s$  sont équivalentes en les transformant en AFD minimum. Si elles sont équivalentes, à un renommage des états près, on doit obtenir le même résultat.

- 1 Expressions régulières
- 2 Conversion en AFD
- 3 Langages non réguliers et lemme de l'étoile

# Langages non réguliers

On peut voir que le langage suivant n'est pas régulier:

$$L = \{a^t b^t \mid t \in \mathbb{N}\} = \{\epsilon, ab, aabb, aaabbb, \dots\}.$$

Idée de preuve: un AFD a une mémoire finie et ne peut pas compter plus haut que son nombre d'états.

- supposons par l'absurde que l'AFD  $A$  à  $n$  états reconnaisse  $L$ .
- Considérons les mots  $a^0 = \epsilon, a^1, a^2, \dots, a^n$ . Il y a  $n+1$  mots différents et  $A$  n'a que  $n$  états. Donc il doit exister  $\alpha \neq \beta$  tel que  $q_0 \xrightarrow{a^\alpha} q_i$  et  $q_0 \xrightarrow{a^\beta} q_i$  pour un état  $q_i \in Q$ .
- Donc  $q_0 \xrightarrow{a^\alpha} q_i \xrightarrow{b^\alpha} q_j$  et  $q_0 \xrightarrow{a^\beta} q_i \xrightarrow{b^\alpha} q_j$  pour un état  $q_j \in Q$ .
- Le mot  $a^\alpha b^\alpha$  est accepté par  $A$  ssi  $a^\beta b^\alpha$  est accepté par  $A$  ce qui est absurde.



## Lemme de l'étoile

Si  $L$  est régulier, alors il existe un nombre  $n$  tel que pour tout mot  $w$  de  $L$ , si  $|w| \geq n$ , alors  $w$  peut être factorisé en  $w = xyz$  de telle sorte que

- $1 \leq |y| \leq |xy| \leq n$ .
- $\forall t \geq 0, xy^t z \in L$ .

Le lemme de l'étoile (aussi appelé lemme d'itération, lemme de pompage, ...) permet de prouver que certains langages sont non réguliers.

Idée de preuve:

- Soit  $A$  l'AFD minimum qui reconnaît  $L$  et  $n$  son nombre d'états.
- Prenons  $w \in L$ ,  $|w| \geq n$  et notons

$$q_{i_0} = q_0 \xrightarrow{w_1} q_{i_1} \xrightarrow{w_2} \dots \xrightarrow{w_{|w|}} q_{i_{|w|}}.$$

(notons que  $q_{i_{|w|}}$  est acceptant).

- Comme  $A$  n'a que  $n$  états,  $\exists \leq \alpha < \beta \leq n$  tel que  $q_{i_\alpha} = q_{i_\beta}$ .
- Prenons le plus petit tel  $\alpha$  et notons  $x = w_{[0, \alpha-1]}$  (peut être égale à  $\epsilon$ ),  $y = w_{[\alpha, \beta]}$  et  $z = w_{[\beta+1, |w|]}$ .
- On a donc:

$$q_{i_0} \xrightarrow{x} q_{i_\alpha} \xrightarrow{y} q_{i_\alpha} \xrightarrow{z} q_{i_{|w|}} \text{ et donc } q_{i_0} \xrightarrow{x} q_{i_\alpha} \xrightarrow{y^t} q_{i_\alpha} \xrightarrow{z} q_{i_{|w|}}$$

# Langages non réguliers: Exemple d'utilisation

Exemple d'utilisation: montrons que  $L = \{a^t b^t \mid t \in \mathbb{N}\}$  n'est pas régulier.

Par l'absurde, supposons que  $L$  est régulier. Donc il existe un  $n$  tel que tout mot  $w \in L$  alors  $w$  peut être factorisé en  $w = xyz$  de telle sorte que

- $1 \leq |y| \leq |xy| \leq n$ .
- $\forall t \geq 0, xy^t z \in L$ .

Prenons  $w = a^n b^n$ .

- Donc,  $xy \in \{a\}^*$ .
- Donc,  $y = a^\alpha$  avec  $\alpha \geq 1$ .
- Donc,  $xy^2 z = a^{n+\alpha} b^n \notin L$ .
- Absurde, donc  $L$  n'est pas régulier.

# Langages non réguliers: Exemple d'utilisation

Exemple d'utilisation: montrons que  $L$  le langage de Dyck n'est pas régulier.  $L$  est l'ensemble des mots sur l'alphabet  $\{(, )\}$  bien parenthésés.

$$L = \{\epsilon, (), (()), ()(), (())(), \dots\}.$$

Par l'absurde, supposons que  $L$  est régulier. Donc il existe un  $n$  tel que tout mot  $w \in L$  alors  $w$  peut être factorisé en  $w = xyz$  de telle sorte que

- $1 \leq |y| \leq |xy| \leq n$ .
- $\forall t \geq 0, xy^t z \in L$ .

Prenons  $w = ({}^n)^n$ .

- Donc,  $xy \in \{()\}^*$ .
- Donc,  $y = ({}^\alpha$  avec  $\alpha \geq 1$ .
- Donc,  $xy^2 z = ({}^{n+\alpha})^n \notin L$ .
- Absurde, donc  $L$  n'est pas régulier.