Théorie des langages

Langages réguliers

Jérôme Delobelle
jerome.delobelle@u-paris.fr

LIPADE - Université de Paris

Langages réguliers

1. Définitions
2. D'un automate fini vers une expression régulière
3. D'une expression régulière vers un automate fini
4. Grammaires régulières et automates finis
5. Caractérisation des langages réguliers
6. Au delà des langages réguliers

Définitions

Langages réguliers

Les langages réguliers sont les langages obtenus à partir des « atomes » $\emptyset$, $\{\epsilon\}$ et $\{a\}, \forall a \in \Sigma$, par un nombre **fini** d'applications de l'union, la concaténation et la fermeture de Kleene.

Langages réguliers

Les langages réguliers sont les langages obtenus à partir des « atomes » $\emptyset$, $\{\epsilon\}$ et $\{a\}, \forall a \in \Sigma$, par un nombre **fini** d'applications de l'union, la concaténation et la fermeture de Kleene.

Definition

L'ensemble REG des **langages réguliers** sur un alphabet Σ est le plus petit des sous-ensembles de $\mathscr{P}(\Sigma^*)$ des langages satisfaisant les conditions :

1. $\emptyset \in REG$ et $\{\epsilon\} \in REG$
2. $\forall a \in \Sigma$, $\{a\} \in REG$
3. Si $A, B \in REG$, alors $A \cup B \in REG$, $A.B \in REG$ et $A^* \in REG$

Langages réguliers

Les langages réguliers sont les langages obtenus à partir des « atomes » $\emptyset$, $\{\epsilon\}$ et $\{a\}, \forall a \in \Sigma$, par un nombre **fini** d'applications de l'union, la concaténation et la fermeture de Kleene.

Definition

L'ensemble *REG* des **langages réguliers** sur un alphabet Σ est le plus petit des sous-ensembles de $\mathscr{P}(\Sigma^*)$ des langages satisfaisant les conditions :

1. $\emptyset \in REG$ et $\{\epsilon\} \in REG$
2. $\forall a \in \Sigma$, $\{a\} \in REG$
3. Si $A, B \in REG$, alors $A \cup B \in REG$, $A.B \in REG$ et $A^* \in REG$

- Pour tout mot $u \in \Sigma^*$, le langage $\{u\}$ est régulier
- Tout langage fini est régulier
- Σ^* est un langage régulier

Propriétés des langages réguliers

Propriétés

Soient L et M deux langages réguliers ($L, M \in REG$). Les langages suivants sont également réguliers :

- $L \cup M$ (union)
- L^* (fermeture de Kleene)
- $L.M$ (concaténation)
- $L \cap M$ (intersection)
- L^R (miroir)

Expressions régulières

Les expressions régulières permettent de décrire les langages réguliers, de façon plus simple qu'en utilisant des opérations ensemblistes.

Expressions régulières

Les **expressions régulières** sur un alphabet Σ sont les règles formées par les règles suivantes :

1. $\emptyset$ et ϵ sont des expressions régulières
2. $\forall a \in \Sigma$, a est une expression régulière
3. Si α et β sont des expressions régulières alors

$$\left.\begin{array}{l}(\alpha+\beta)\\(\alpha.\beta)\\(\alpha)^*\end{array}\right\}\text{ sont des expressions régulières}$$

Expressions régulières

Les expressions régulières permettent de décrire les langages réguliers, de façon plus simple qu'en utilisant des opérations ensemblistes.

Expressions régulières

Les **expressions régulières** sur un alphabet Σ sont les règles formées par les règles suivantes :

1. $\emptyset$ et ϵ sont des expressions régulières
2. $\forall a \in \Sigma$, a est une expression régulière
3. Si α et β sont des expressions régulières alors

$$\left.\begin{array}{l}(\alpha+\beta)\\(\alpha.\beta)\\(\alpha)^*\end{array}\right\}\text{ sont des expressions régulières}$$

Priorité dans l'ordre décroissant : *, ., $+$

Propriétés des expressions régulières

Soient r, s et t trois expressions régulières sur le même alphabet Σ.

1. $r+s=s+r$
2. $r+\emptyset=\emptyset+r=r$
3. $r+r=r$
4. $(r+s)+t=r+(s+t)=r+s+t$
5. $r.\epsilon=\epsilon.r=r$
6. $r.\emptyset=\emptyset.r=\emptyset$
7. $(r.s).t=r.(s.t)=r.s.t$
8. $r.(s+t)=r.s+r.t$

Propriétés des expressions régulières

Soient r, s et t trois expressions régulières sur le même alphabet Σ.

9. $r^* = (r^*)^* = r^* r^* = (\epsilon + r)^* = r^*(r+\epsilon) = (r+\epsilon)r^* = \epsilon + rr^* = \epsilon + r^* r$
10. $(r+s)^* = (r^* s^*)^* = (r^* s)^* r^* = (s^* r)^* s^* = r^*(sr^*)^*$
11. $r(sr)^* = (rs)^* r$
12. $(r^* s)^* = \epsilon + (r+s)^* s$
13. $(rs^*)^* = \epsilon + r(r+s)^*$
14. $rr^* = r^* r = r^+$

Langage représenté par une expression régulière

Soit r une expression régulière. $\mathscr{L}(r)$ est le **langage représenté par** r.

1. $\mathscr{L}(\emptyset) = \emptyset$, $\mathscr{L}(\epsilon) = \{\epsilon\}$
2. $\forall a \in \Sigma$, $\mathscr{L}(a) = \{a\}$
3. $\mathscr{L}(\alpha, \beta) = \mathscr{L}(\alpha) \cup \mathscr{L}(\beta) = \mathscr{L}(\alpha) + \mathscr{L}(\beta)$
4. $\mathscr{L}(\alpha.\beta) = \mathscr{L}(\alpha).\mathscr{L}(\beta)$
5. $\mathscr{L}((\alpha)^*) = (\mathscr{L}(\alpha))^*$

Expressions régulières

Théorème

Un langage est régulier si et seulement si il peut être dénoté par une expression régulière.

Expressions régulières

Théorème

Un langage est régulier si et seulement si il peut être dénoté par une expression régulière.

Cela implique que :

1. Toute expression régulière décrit un langage régulier
2. Tout langage régulier peut être décrit par une expression régulière

Expressions régulières

Egalité d'expressions régulières

Deux expressions régulières sont **égales** si elles représentent le même langage.

Expressions régulières

Egalité d'expressions régulières

Deux expressions régulières sont **égales** si elles représentent le même langage.

Exemple :

$$r^* = r^* + \epsilon \text{ car } \epsilon \in r^*$$

Passer d'une description d'un langage régulier à une autre ?

Grammaire régulière

Automate fini

Expression régulière

Passer d'une description d'un langage régulier à une autre ?

D’un automate fini vers une expression régulière

Méthodes

- Algorithme par résolution d'équations (Lemme d'Arden)
- Algorithme par élimination d'états (ou réduction d'automates)
- Algorithme de McNaughton et Yamada
- ...

Méthodes

- **Algorithme par résolution d'équations (Lemme d'Arden)**
- **Algorithme par élimination d'états (ou réduction d'automates)**
- Algorithme de McNaughton et Yamada
- ...

D’un automate fini vers une expression régulière

Théorème d’Arden

Rappel : Système d'équations définissant un langage

Equation définissant un langage généré à partir d'un état

Le langage reconnu à partir d'un état q par un automate M est défini par une équation de la forme :

$$L(q) = (\bigcup_{x \in \Sigma} x.L(\delta(q,x))) \cup d(L(q))$$

$$\text{où } d(A) = \begin{cases} \emptyset & \text{si A n'est pas un état final} \\ \{\epsilon\} & \text{si A est un état final} \end{cases}$$

On pourra également noter :

$$L_q = (\sum_{x \in \Sigma} x.L(\delta(q,x))) + d(L_q)$$

Le théorème d'Arden

Théorème d'Arden

Une équation sur les langages de la forme $X = AX + B$, où $\epsilon \notin A$, a une solution unique $X = A^* B$

Le théorème d'Arden

Théorème d'Arden

Une équation sur les langages de la forme $X = AX + B$, où $\epsilon \notin A$, a une solution unique $X = A^* B$

Si $\epsilon \in A$, $A^* B$ est une solution mais ce n'est pas une solution unique. ($A^* B$ est inclus dans toutes les solutions.)

Le théorème d'Arden

Théorème d'Arden

Une équation sur les langages de la forme $X = AX + B$, où $\epsilon \notin A$, a une solution unique $X = A^*B$

Si $\epsilon \in A$, A^*B est une solution mais ce n'est pas une solution unique. (A^*B est inclus dans toutes les solutions.)

<u>Démonstration</u> :

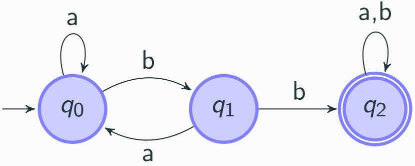
1. $X = A^*B$ est solution : $AX + B = A.A^*B + B = (A.A^* + \epsilon)B = A^*B$
2. A^*B est solution unique : si Y est solution, alors Y est de la forme A^*B.

Intérêt du théorème d'Arden

Transformation d'un automate en ER

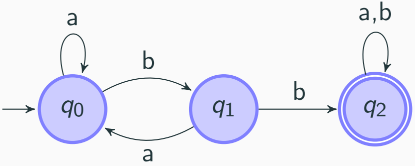
Grâce au Théorème d'Arden, il est possible de résoudre un système d'équations et d'obtenir une expression régulière qui représente le langage reconnu par l'automate.

Transformation d'un automate en ER : le théorème d'Arden



$$\begin{cases} L_0 & = & aL_0 + bL_1 \\ L_1 & = & aL_0 + bL_2 \\ L_2 & = & aL_2 + bL_2 + \epsilon \end{cases}$$

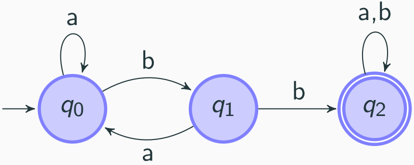
Transformation d'un automate en ER : le théorème d'Arden



$$\begin{cases} L_0 & = & aL_0 + bL_1 \\ L_1 & = & aL_0 + bL_2 \\ L_2 & = & aL_2 + bL_2 + \epsilon \end{cases}$$

$$\begin{cases} L_0 & = & aL_0 + bL_1 \\ L_1 & = & aL_0 + bL_2 \\ L_2 & = & (a+b)L_2 + \epsilon \end{cases}$$

Transformation d'un automate en ER : le théorème d'Arden

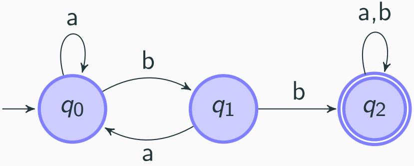


$$\begin{cases} L_0 & = & aL_0 + bL_1 \\ L_1 & = & aL_0 + bL_2 \\ L_2 & = & aL_2 + bL_2 + \epsilon \end{cases}$$

$$\begin{cases} L_0 & = & aL_0 + bL_1 \\ L_1 & = & aL_0 + bL_2 \\ L_2 & = & (a+b)^*\epsilon = (a+b)^* \end{cases}$$

Application du théorème d'Arden sur L_2 avec $A = (a+b)$ et $B = \epsilon$

Transformation d'un automate en ER : le théorème d'Arden

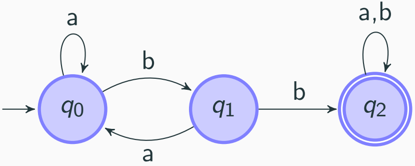


$$\begin{cases} L_0 & = & aL_0 + bL_1 \\ L_1 & = & aL_0 + bL_2 \\ L_2 & = & aL_2 + bL_2 + \epsilon \end{cases}$$

$$\begin{cases} L_0 & = & aL_0 + bL_1 \\ L_1 & = & aL_0 + b(a+b)^* \\ L_2 & = & (a+b)^* \end{cases}$$

Impossible d'appliquer le théorème d'Arden sur L_1

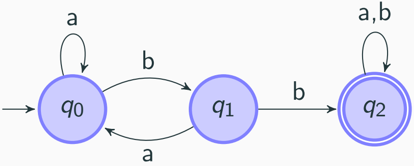
Transformation d'un automate en ER : le théorème d'Arden



$$\begin{cases} L_0 &= aL_0 + bL_1 \\ L_1 &= aL_0 + bL_2 \\ L_2 &= aL_2 + bL_2 + \epsilon \end{cases}$$

$$\begin{cases} L_0 &= aL_0 + baL_0 + bb(a+b)^* \\ L_1 &= aL_0 + b(a+b)^* \\ L_2 &= (a+b)^* \end{cases}$$

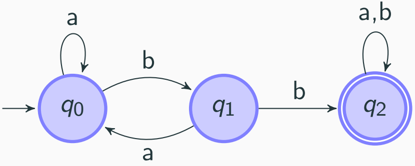
Transformation d'un automate en ER : le théorème d'Arden



$$\left\{\begin{array}{lcl} L_0 & = & aL_0 + bL_1 \\ L_1 & = & aL_0 + bL_2 \\ L_2 & = & aL_2 + bL_2 + \epsilon \end{array}\right.$$

$$\left\{\begin{array}{lcl} L_0 & = & (a+ba)L_0 + bb(a+b)^* \\ L_1 & = & aL_0 + b(a+b)^* \\ L_2 & = & (a+b)^* \end{array}\right.$$

Transformation d'un automate en ER : le théorème d'Arden

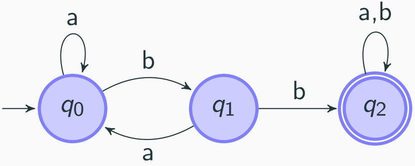


$$\begin{cases} L_0 &= aL_0 + bL_1 \\ L_1 &= aL_0 + bL_2 \\ L_2 &= aL_2 + bL_2 + \epsilon \end{cases}$$

$$\begin{cases} L_0 &= (a+ba)^* bb(a+b)^* \\ L_1 &= aL_0 + b(a+b)^* \\ L_2 &= (a+b)^* \end{cases}$$

Application du théorème d'Arden sur L_0 avec
$A = a + ba$ et $B = bb(a+b)^*$

Transformation d'un automate en ER : le théorème d'Arden



$$\begin{cases} L_0 & = & aL_0 + bL_1 \\ L_1 & = & aL_0 + bL_2 \\ L_2 & = & aL_2 + bL_2 + \epsilon \end{cases}$$

$$\begin{cases} L_0 & = & (a+ba)^* bb(a+b)^* \\ L_1 & = & aL_0 + b(a+b)^* \\ L_2 & = & (a+b)^* \end{cases}$$

$$L_0 \quad = \quad (a+ba)^* bb(a+b)^*$$

D’un automate fini vers une expression régulière

Elimination d’états

Transformation d'un automate en ER : élimination d'états

Méthode d'élimination d'état (algorithme BMC)

On cherche une expression régulière dénotant le langage reconnu par un automate M. On procède par suppression successive de transitions et d'états :

1. Ajouter à M deux nouveaux états, notés α et ω, et les transitions (α,ϵ,q_0) pour q_0 l'état initial ; et (q_n,ϵ,ω) pour $q_n \in F$.

Transformation d'un automate en ER : élimination d'états

Méthode d'élimination d'état (algorithme BMC)

On cherche une expression régulière dénotant le langage reconnu par un automate M. On procède par suppression successive de transitions et d'états :

1. Ajouter à M deux nouveaux états, notés α et ω, et les transitions (α,ϵ,q_0) pour q_0 l'état initial ; et (q_n,ϵ,ω) pour $q_n \in F$.
2. Itérer les réductions suivantes tant que possible :

Transformation d'un automate en ER : élimination d'états

Méthode d'élimination d'état (algorithme BMC)

On cherche une expression régulière dénotant le langage reconnu par un automate M. On procède par suppression successive de transitions et d'états :

1. Ajouter à M deux nouveaux états, notés α et ω, et les transitions (α,ϵ,q_0) pour q_0 l'état initial ; et (q_n,ϵ,ω) pour $q_n \in F$.
2. Itérer les réductions suivantes tant que possible :
   - s'il existe deux transitions (q_i,x,q_j) et (q_i,y,q_j), les remplacer par la transition $(q_i,x+y,q_j)$

Transformation d'un automate en ER : élimination d'états

Méthode d'élimination d'état (algorithme BMC)

On cherche une expression régulière dénotant le langage reconnu par un automate M. On procède par suppression successive de transitions et d'états :

1. Ajouter à M deux nouveaux états, notés α et ω, et les transitions (α,ϵ,q_0) pour q_0 l'état initial ; et (q_n,ϵ,ω) pour $q_n \in F$.
2. Itérer les réductions suivantes tant que possible :
   - s'il existe deux transitions (q_i,x,q_j) et (q_i,y,q_j), les remplacer par la transition $(q_i,x+y,q_j)$
   - supprimer un état q (autre que α et ω) et remplacer, pour tous les états $p,r \neq q$, les transitions (p,x,q), (q,y,q), (q,z,r), par la transition (p,xy^*z,r).

Transformation d'un automate en ER : élimination d'états

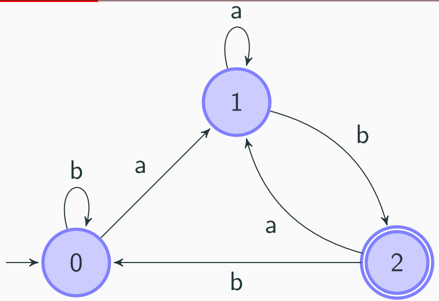
Méthode d'élimination d'état (algorithme BMC)

On cherche une expression régulière dénotant le langage reconnu par un automate M. On procède par suppression successive de transitions et d'états :

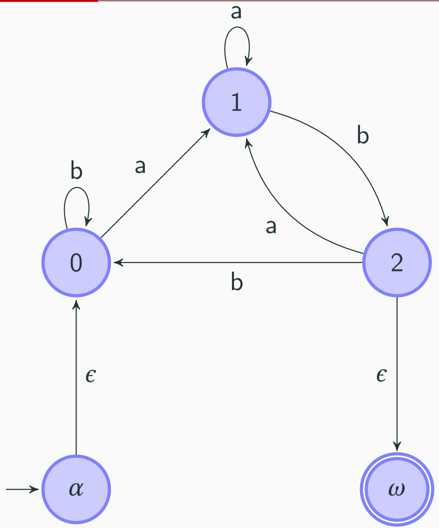
1. Ajouter à M deux nouveaux états, notés α et ω, et les transitions (α, ϵ, q_0) pour q_0 l'état initial ; et (q_n, ϵ, ω) pour $q_n \in F$.
2. Itérer les réductions suivantes tant que possible :
   - s'il existe deux transitions (q_i, x, q_j) et (q_i, y, q_j), les remplacer par la transition $(q_i, x + y, q_j)$
   - supprimer un état q (autre que α et ω) et remplacer, pour tous les états $p, r \neq q$, les transitions (p, x, q), (q, y, q), (q, z, r), par la transition (p, xy^*z, r).

Cet algorithme termine car on diminue le nombre de transitions et d'états, jusqu'à obtenir une seule transition (α, e, ω). e est alors une expression régulière pour le langage $\mathscr{L}(M)$.

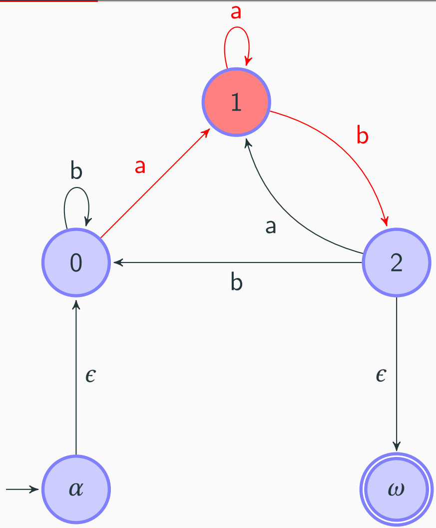
Transformation d’un automate en ER : élimination d’états



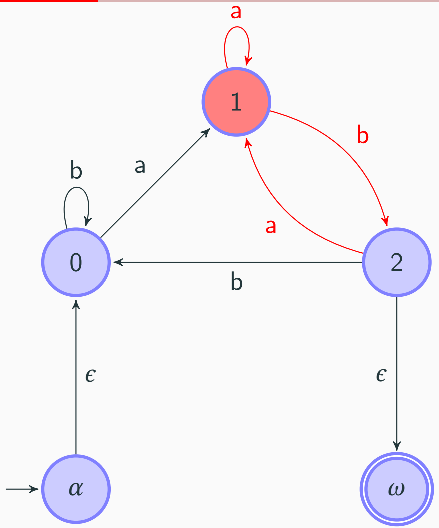
Transformation d'un automate en ER : élimination d'états



Transformation d'un automate en ER : élimination d'états

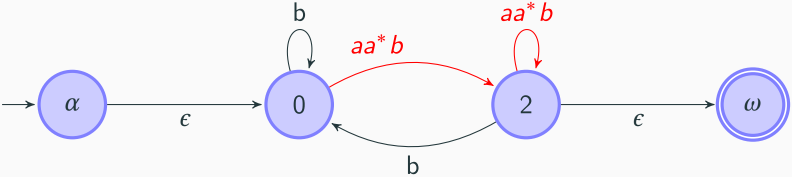


$$(0, a, 1), (1, a, 1), (1, b, 2) \Rightarrow (0, aa^* b, 2)$$

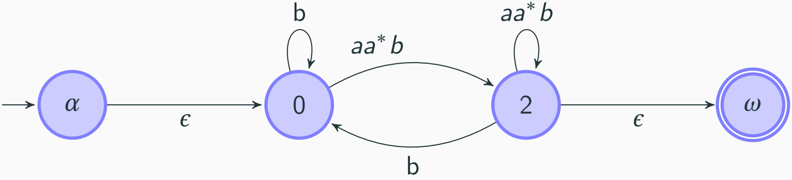


$$(2, a, 1), (1, a, 1), (1, b, 2) \Rightarrow (2, aa^* b, 2)$$

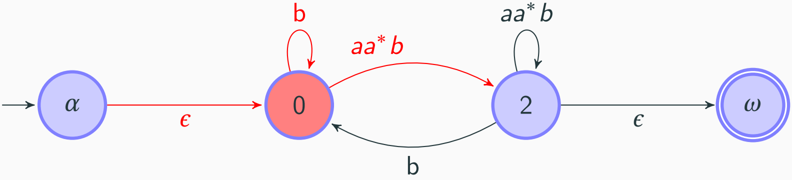
Transformation d'un automate en ER : élimination d'états



Transformation d'un automate en ER : élimination d'états

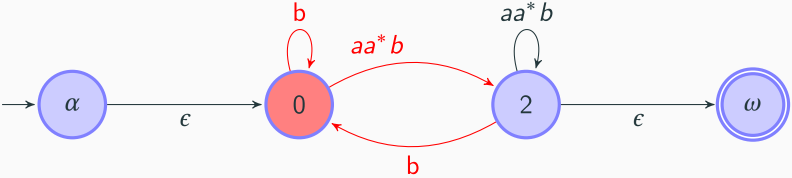


Transformation d'un automate en ER : élimination d'états



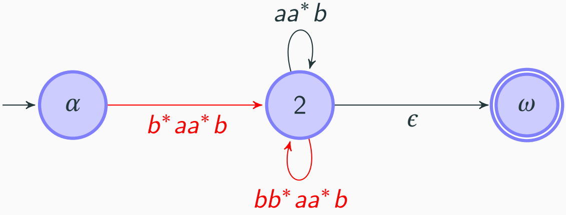
$$(\alpha, \epsilon, 0), (0, b, 0), (0, aa^* b, 2) \Rightarrow (\alpha, b^* aa^* b, 2)$$

Transformation d'un automate en ER : élimination d'états

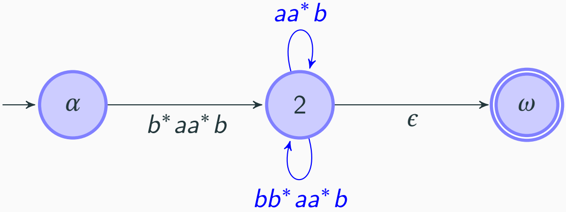


$$(2,b,0),(0,b,0),(0,aa^*b,2) \Rightarrow (2,bb^*aa^*b,2)$$

Transformation d'un automate en ER : élimination d'états

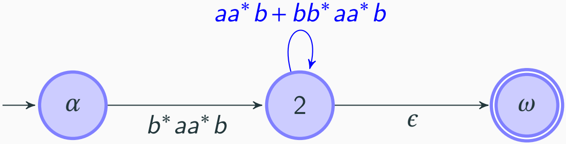


Transformation d'un automate en ER : élimination d'états

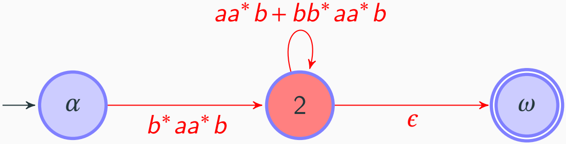


$$(2, aa^* b, 2), (2, bb^* aa^* b, 2) \Rightarrow (2, aa^* b + bb^* aa^* b, 2)$$

Transformation d'un automate en ER : élimination d'états

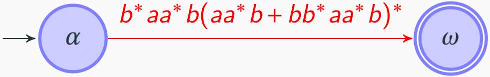


Transformation d'un automate en ER : élimination d'états

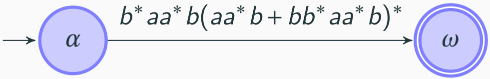


$$(\alpha, b^*aa^*b, 2), (2, aa^*b + bb^*aa^*b, 2), (2, \epsilon, \omega)$$
$$\Downarrow$$
$$(\alpha, b^*aa^*b(aa^*b + bb^*aa^*b)^*, \omega)$$

Transformation d'un automate en ER : élimination d'états

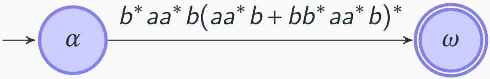


Transformation d'un automate en ER : élimination d'états



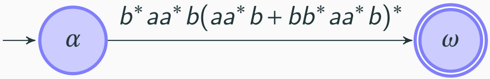
$$b^* aa^* b(aa^* b + bb^* aa^* b)^* \quad = \quad b^* a^+ b(a^+ b + b^+ a^+ b)^* \quad \text{R. 14. : } rr^* = r^+$$

Transformation d'un automate en ER : élimination d'états

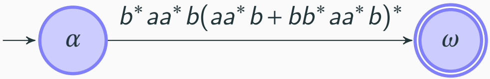


$$\begin{aligned} b^* aa^* b(aa^* b + bb^* aa^* b)^* \quad &= \quad b^* a^+ b(a^+ b + b^+ a^+ b)^* \quad \text{R. 14. : } rr^* = r^+ \\ &= \quad b^* a^+ b((\epsilon + b^+) a^+ b)^* \end{aligned}$$

Transformation d'un automate en ER : élimination d'états

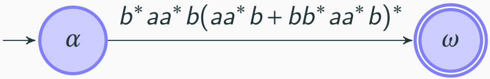


$$
\begin{aligned}
b^* aa^* b(aa^* b + bb^* aa^* b)^* \quad &= \quad b^* a^+ b(a^+ b + b^+ a^+ b)^* \quad \text{R. 14. : } rr^* = r^+ \\
&= \quad b^* a^+ b((\epsilon + b^+)a^+ b)^* \\
&= \quad b^* a^+ b(b^* a^+ b)^*
\end{aligned}
$$



$$
\begin{aligned}
b^*aa^*b(aa^*b+bb^*aa^*b)^* &= b^*a^+b(a^+b+b^+a^+b)^* \quad \text{R. 14. : } rr^*=r^+\\
&= b^*a^+b((\epsilon+b^+)a^+b)^*\\
&= {\color{red}b^*a^+b}{\color{blue}(b^*a^+b)^*}\\
&= {\color{blue}(b^*a^+b)^*}{\color{red}b^*a^+b} \qquad\qquad \text{R. 14. : } rr^*=r^*r
\end{aligned}
$$

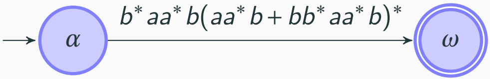
Transformation d'un automate en ER : élimination d'états



$$b^* aa^* b(aa^* b + bb^* aa^* b)^* \quad = \quad b^* a^+ b(a^+ b + b^+ a^+ b)^* \qquad \text{R. 14. : } rr^* = r^+$$

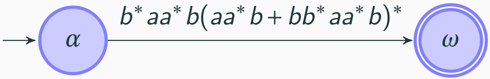
$$\begin{aligned}
&= \quad b^* a^+ b((\epsilon + b^+) a^+ b)^* \\
&= \quad b^* a^+ b(b^* a^+ b)^* \\
&= \quad (b^* a^+ b)^* b^* a^+ b && \text{R. 14. : } rr^* = r^* r \\
&= \quad (b + a^+ b)^* a^+ b && \text{R. 10. : } (r^* s)^* r^* = (r + s)^*
\end{aligned}$$

Transformation d'un automate en ER : élimination d'états



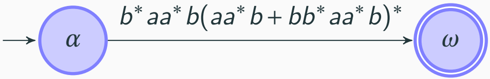
$$
\begin{aligned}
b^* aa^* b(aa^* b + bb^* aa^* b)^* \quad &= \quad b^* a^+ b(a^+ b + b^+ a^+ b)^* && \text{R. 14. : } rr^* = r^+ \\
&= \quad b^* a^+ b((\epsilon + b^+) a^+ b)^* && \\
&= \quad b^* a^+ b(b^* a^+ b)^* && \\
&= \quad (b^* a^+ b)^* b^* a^+ b && \text{R. 14. : } rr^* = r^* r \\
&= \quad ({\color{red}b} + {\color{blue}a^+} {\color{red}b})^* a^+ b && \text{R. 10. : } (r^* s)^* r^* = (r+s)^* \\
&= \quad (a^* b)^* a^+ b && r + s^+ r = s^+ r + r = (s^+ + \epsilon) r = s^* r
\end{aligned}
$$

Transformation d'un automate en ER : élimination d'états



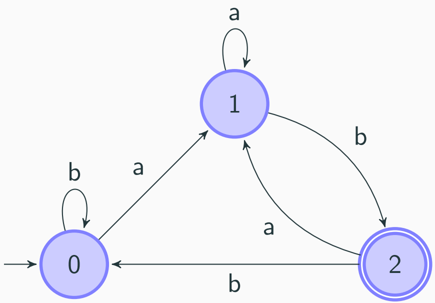
$$
\begin{aligned}
b^*aa^*b(aa^*b+bb^*aa^*b)^* &= b^*a^+b(a^+b+b^+a^+b)^* && \text{R. 14. : } rr^* = r^+ \\
&= b^*a^+b((\epsilon+b^+)a^+b)^* \\
&= b^*a^+b(b^*a^+b)^* \\
&= (b^*a^+b)^*b^*a^+b && \text{R. 14. : } rr^* = r^*r \\
&= (b+a^+b)^*a^+b && \text{R. 10. : } (r^*s)^*r^* = (r+s)^* \\
&= (a^*b)^*a^+b && r+s^+r = (s^++\epsilon)r = s^*r \\
&= (a^*b)^*a^*ab && \text{R. 14. : } r^*r = r^+
\end{aligned}
$$

Transformation d'un automate en ER : élimination d'états



$$
\begin{aligned}
b^*aa^*b(aa^*b+bb^*aa^*b)^* &= b^*a^+b(a^+b+b^+a^+b)^* && \text{R. 14. : } rr^* = r^+ \\
&= b^*a^+b((\epsilon+b^+)a^+b)^* && \\
&= b^*a^+b(b^*a^+b)^* && \\
&= (b^*a^+b)^*b^*a^+b && \text{R. 14. : } rr^* = r^*r \\
&= (b+a^+b)^*a^+b && \text{R. 10. : } (r^*s)^*r^* = (r+s)^* \\
&= (a^*b)^*a^+b && r+s^+r = (s^++\epsilon)r = s^*r \\
&= (a^*b)^*a^*ab && \text{R. 14. : } r^*r = r^+ \\
&= (a+b)^*ab && \text{R. 10. : } (r^*s)^*r^* = (r+s)^*
\end{aligned}
$$

Transformation d'un automate en ER : élimination d'états



$$\mathscr{L}(M) = (a+b)^* ab$$

D'une expression régulière vers un automate fini

Transformation d’une ER en automate

Théorème

Pour chaque expression régulière, il existe un automate fini qui reconnaît cette expression

Plusieurs méthodes pour y parvenir :

- Automate de Thompson
- Algorithme de Gloushkov
- Automate des résiduels
- ...

Transformation d'une ER en automate

Théorème

Pour chaque expression régulière, il existe un automate fini qui reconnaît cette expression

Plusieurs méthodes pour y parvenir :

- **Automate de Thompson**
- Algorithme de Gloushkov
- Automate des résiduels
- ...

D’une expression régulière vers un automate fini

Construction de Thompson

Automate fini asynchrone (AFA)

Automate fini asynchrone

Un **automate fini asynchrone (AFA)** est un quintuplet $M = \langle Q, \Sigma, \Delta, S, F \rangle$ où

- Q est un ensemble fini d'*états*
- Σ est un ensemble fini de symboles (un *alphabet*)
- $\Delta \subseteq (Q \times \Sigma \cup \{\epsilon\} \times Q)$ est une *relation de transitions*
- $S \subseteq Q$ est l'ensemble (fini) des *état initiaux*
- $F \subseteq Q$ est l'ensemble (fini) des *états finaux*

Automate fini asynchrone (AFA)

Automate fini asynchrone

Un **automate fini asynchrone (AFA)** est un quintuplet $M = \langle Q, \Sigma, \Delta, S, F \rangle$ où

- Q est un ensemble fini d'*états*
- Σ est un ensemble fini de symboles (un *alphabet*)
- $\Delta \subseteq (Q \times \Sigma \cup \{\epsilon\} \times Q)$ est une *relation de transitions*
- $S \subseteq Q$ est l'ensemble (fini) des *état initiaux*
- $F \subseteq Q$ est l'ensemble (fini) des *états finaux*

Un AFA est dit normalisé si :

- il ne possède qu'un état initial ($|S| = 1$)
- il ne possède qu'un état final ($|F| = 1$)
- Aucune transition ne va vers l'état initial
- Aucune transition ne part de l'état final

Algorithme de Thompson

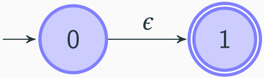
Algorithme de Thompson

Décrit l'assemblage des automates correspondant aux opérations sur les expressions régulières

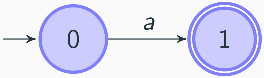
⇒ Construction d'un AFA normalisé associé à une ER

- Automate de Thompson pour $\emptyset$

- Automate de Thompson pour $\{\epsilon\}$



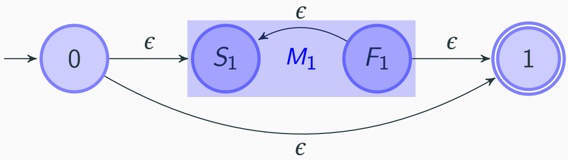
- Automate de Thompson pour $\{a\}$



Algorithme de Thompson

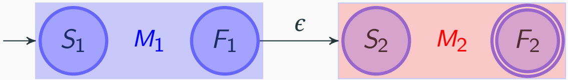
- Soit M_1 l'automate de l'ER e_1

$\boxed{e_1^*}$



- Soit M_1 l'automate de l'ER e_1 et M_2 l'automate de l'ER e_2

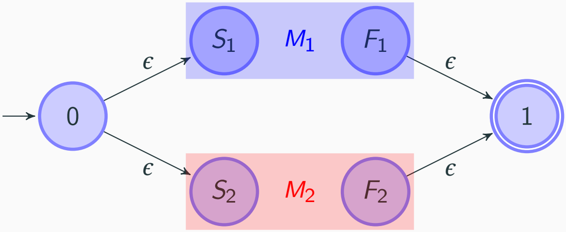
$\boxed{e_1.e_2}$



Algorithme de Thompson

Soit M_1 l'automate de l'ER e_1 et M_2 l'automate de l'ER e_2.

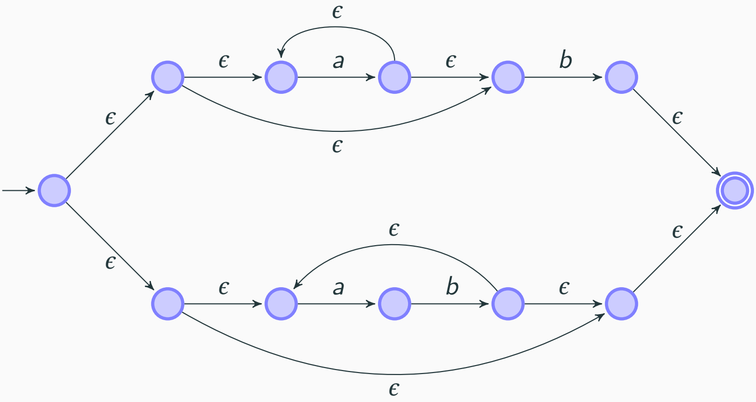
$\boxed{e_1 + e_2}$



Exemple

Automate de Thompson (AFA normalisé) associé à l'ER suivante :

$$a^* b + (ab)^*$$



Elimination des ϵ-transitions

AFA normalisé $\longrightarrow$ **AFD**

Elimination des ϵ-transitions

AFA normalisé $\longrightarrow$ **AFD**

Comment determiniser un AFA normalisé ?

Elimination des ϵ-transitions

AFA normalisé $\longrightarrow$ **AFD**

Comment determiniser un AFA normalisé ?

Par rapport à l'algorithme de déterminisation d'un automate ne contenant pas de ϵ-transition, la transformation consiste simplement à remplacer chaque état créé par sa **fermeture epsilon**.

Elimination des ϵ-transitions

AFA normalisé $\longrightarrow$ **AFD**

Comment determiniser un AFA normalisé ?

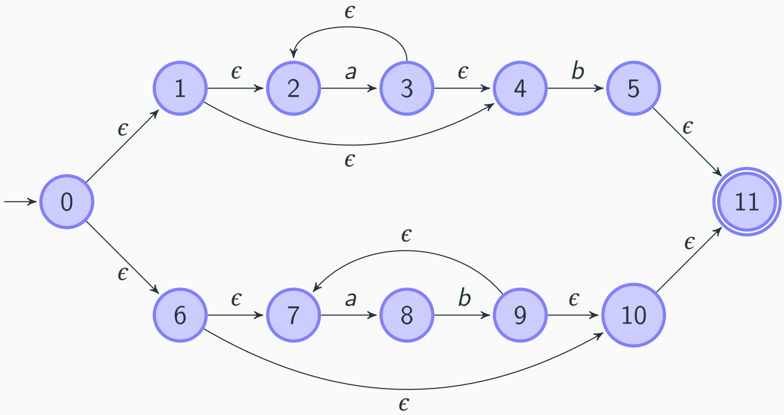
Par rapport à l'algorithme de déterminisation d'un automate ne contenant pas de ϵ-transition, la transformation consiste simplement à remplacer chaque état créé par sa **fermeture epsilon**.

Fermeture epsilon

Soit $M = \langle Q, \Sigma, \Delta, S, F \rangle$ un AFA.
La fermeture epsilon d'un état $q \in Q$, noté $\{q\}^\epsilon$, sont tous les états accessibles à partir de p par ϵ-transition (en plusieurs étapes).

$$\forall Q' \subseteq Q, \{Q'\}^\epsilon = \bigcup_{q' \in Q'} \{q'\}^\epsilon$$

Elimination des ϵ-transitions



$\{0\}^\epsilon = \{0, 1, 2, 4, 6, 7, 10, 11\}$

$\{3\}^\epsilon = \{2, 3, 4\}$

$\{0, 3\}^\epsilon = \{0\}^\epsilon \cup \{3\}^\epsilon = \{0, 1, 2, 3, 4, 6, 7, 10, 11\}$

Elimination des ϵ-transitions

Etat p	$\{p\}^{\epsilon}$	a	b
0	$\{0,1,2,4,6,7,10,11\}$	-	-
1	$\{1,2,4\}$	-	-
2	$\{2\}$	$\{3\}$	-
3	$\{2,3,4\}$	-	-
4	$\{4\}$	-	$\{5\}$
5	$\{5,11\}$	-	-
6	$\{6,7,10,11\}$	-	-
7	$\{7\}$	$\{8\}$	-
8	$\{8\}$	-	$\{9\}$
9	$\{7,9,10,11\}$	-	-
10	$\{10,11\}$	-	-
11	$\{11\}$	-	-

Déterminiser l'AFA normalisé

1) On calcule, pour chaque symbole, tous les états accessibles par la fermeture epsilon de l'état initial.

2) Idem pour tous les nouveaux états trouvés jusqu'à stabilisation.

	a	b
$\{0\}^{\epsilon} = \{0, 1, 2, 4, 6, 7, 10, 11\}$	$\{3, 8\}^{\epsilon}$	$\{5\}^{\epsilon}$

Déterminiser l'AFA normalisé

1) On calcule, pour chaque symbole, tous les états accessibles par la fermeture epsilon de l'état initial.

2) Idem pour tous les nouveaux états trouvés jusqu'à stabilisation.

	a	b
$\{0\}^\epsilon = \{0,1,2,4,6,7,10,11\}$	$\{3,8\}^\epsilon$	$\{5\}^\epsilon$
$\{3,8\}^\epsilon = \{2,3,4,8\}$	$\{3\}^\epsilon$	$\{5,9\}^\epsilon$
$\{5\}^\epsilon = \{5,11\}$	-	-
$\{3\}^\epsilon = \{2,3,4\}$	$\{3\}^\epsilon$	$\{5\}^\epsilon$
$\{5,9\}^\epsilon = \{5,7,9,10,11\}$	$\{8\}^\epsilon$	-
$\{8\}^\epsilon = \{8\}$	-	$\{9\}^\epsilon$
$\{9\}^\epsilon = \{7,9,10,11\}$	$\{8\}^\epsilon$	-

Déterminiser l'AFA normalisé

- Etat initial (unique) : Fermeture epsilon de l'état initial de l'AFA
- Etats finaux : Tous les états qui contiennent au moins un état final de l'AFA

	a	b
$\{0\}^\epsilon = \{0,1,2,4,6,7,10,11\}$	$\{3,8\}^\epsilon$	$\{5\}^\epsilon$
$\{3,8\}^\epsilon = \{2,3,4,8\}$	$\{3\}^\epsilon$	$\{5,9\}^\epsilon$
$\{5\}^\epsilon = \{5,11\}$	-	-
$\{3\}^\epsilon = \{2,3,4\}$	$\{3\}^\epsilon$	$\{5\}^\epsilon$
$\{5,9\}^\epsilon = \{5,7,9,10,11\}$	$\{8\}^\epsilon$	-
$\{8\}^\epsilon = \{8\}$	-	$\{9\}^\epsilon$
$\{9\}^\epsilon = \{7,9,10,11\}$	$\{8\}^\epsilon$	-

Déterminiser l'AFA normalisé

- Etat initial (unique) : Fermeture epsilon de l'état initial de l'AFA
- Etats finaux : Tous les états qui contiennent au moins un état final de l'AFA

	a	b
$\{0\}^{\epsilon} = \{0, 1, 2, 4, 6, 7, 10, 11\}$	$\{3,8\}^{\epsilon}$	$\{5\}^{\epsilon}$
$\{3,8\}^{\epsilon} = \{2, 3, 4, 8\}$	$\{3\}^{\epsilon}$	$\{5,9\}^{\epsilon}$
$\{5\}^{\epsilon} = \{5, 11\}$	-	-
$\{3\}^{\epsilon} = \{2, 3, 4\}$	$\{3\}^{\epsilon}$	$\{5\}^{\epsilon}$
$\{5,9\}^{\epsilon} = \{5, 7, 9, 10, 11\}$	$\{8\}^{\epsilon}$	-
$\{8\}^{\epsilon} = \{8\}$	-	$\{9\}^{\epsilon}$
$\{9\}^{\epsilon} = \{7, 9, 10, 11\}$	$\{8\}^{\epsilon}$	-

	a	b
$\Leftrightarrow$ 0	1	2
1	3	4
$\Leftarrow$ 2	-	-
3	3	2
$\Leftarrow$ 4	5	-
5	-	6
$\Leftarrow$ 6	5	-

Elimination des ϵ-transitions

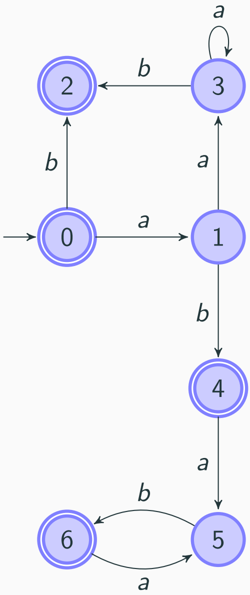


Table de transitions

	a	b
$\Leftrightarrow$ 0	1	2
1	3	4
$\Leftarrow$ 2	-	-
3	3	2
$\Leftarrow$ 4	5	-
5	-	6
$\Leftarrow$ 6	5	-

$$a^* b + (ab)^*$$

Grammaires régulières et automates finis

Transformation d'un automate fini en grammaire

Grammaire associée à un automate fini

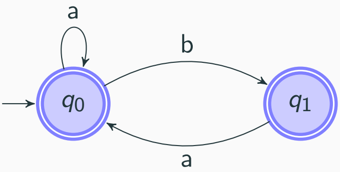
Pour tout AFD $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, il existe une grammaire linéaire à droite qui génère $\mathscr{L}(M)$.

$G = \langle V, \Sigma, P, S \rangle$, avec

- Σ est l'ensemble des symboles terminaux
- $V = Q \cup \Sigma$ est l'alphabet. Il y a donc un symbole non terminal pour chaque état de l'automate
- S est l'axiome associé à q_0
- $P = \{A \to wB \mid \delta(A, w) = B\} \cup \{A \to \epsilon \mid A \in F\}$

Exemple

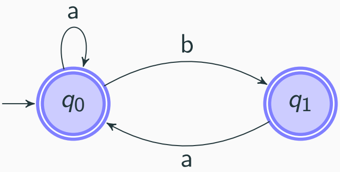
- Automate M_1.



- $\mathscr{L}(M_1)$: $L_0 = aL_0 + bL_1 + \epsilon$; $L_1 = aL_0 + \epsilon$

Exemple

- Automate M_1.



- $\mathscr{L}(M_1)$: $L_0 = aL_0 + bL_1 + \epsilon$; $L_1 = aL_0 + \epsilon$
- $G_{M_1} = \langle V, \Sigma, P, S \rangle$, avec $V = \{a, b, S, U\}$; $\Sigma = \{a, b\}$;

$$P = \begin{cases} S & \rightarrow & aS \\ S & \rightarrow & bU \\ S & \rightarrow & \epsilon \\ U & \rightarrow & aS \\ U & \rightarrow & \epsilon \end{cases}$$

Transformation d'une grammaire linéaire à droite en automate

Automate associé à une grammaire linéaire à droite

Pour toute grammaire linéaire à droite $G = \langle V, \Sigma, P, S \rangle$, il existe un automate $M = \langle Q, \Sigma, \Delta, S, F \rangle$ qui reconnaît $\mathscr{L}(G)$.

- Q : Un état pour chaque symbole non terminal. L'état initial est l'état correspondant à l'axiome S
- F : Les états finaux sont les états dont les non terminaux associés ont une règle du type $A \to \epsilon$
- Il est ensuite possible de construire le système d'équation correspondant à l'automate

Exemple

- $G = \langle V, \Sigma, P, S \rangle$, avec $V = \{a, b, S, U\}$; $\Sigma = \{a, b\}$;

$$P = \begin{cases} S & \rightarrow & bS \\ S & \rightarrow & aU \\ S & \rightarrow & b \\ U & \rightarrow & aS \\ U & \rightarrow & bU \end{cases}$$

Exemple

- $G = \langle V, \Sigma, P, S \rangle$, avec $V = \{a, b, S, U\}$; $\Sigma = \{a, b\}$;

$$P = \begin{cases} S & \rightarrow & bS \\ S & \rightarrow & aU \\ S & \rightarrow & b \\ U & \rightarrow & aS \\ U & \rightarrow & bU \end{cases}$$

Exemple

- $G = \langle V, \Sigma, P, S \rangle$, avec $V = \{a, b, S, U\}$; $\Sigma = \{a, b\}$;

$$P = \begin{cases} S & \rightarrow & bS \\ S & \rightarrow & aU \\ \color{red}{S} & \color{red}{\rightarrow} & \color{red}{b} \\ U & \rightarrow & aS \\ U & \rightarrow & bU \end{cases}$$

- $G' = \langle V', \Sigma, P', S \rangle$, avec $V' = \{a, b, S, U, V\}$; $\Sigma = \{a, b\}$;

$$P = \begin{cases} S & \rightarrow & bS \\ S & \rightarrow & aU \\ S & \rightarrow & bV \\ V & \rightarrow & \epsilon \\ U & \rightarrow & aS \\ U & \rightarrow & bU \end{cases}$$

Exemple

- $G' = \langle V', \Sigma, P', S \rangle$, avec $V' = \{a, b, S, U, V\}$; $\Sigma = \{a, b\}$;

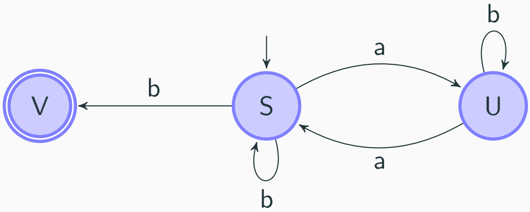
$$P = \begin{cases} S & \rightarrow & bS \\ S & \rightarrow & aU \\ S & \rightarrow & bV \\ V & \rightarrow & \epsilon \\ U & \rightarrow & aS \\ U & \rightarrow & bU \end{cases}$$

Exemple

- $G' = \langle V', \Sigma, P', S \rangle$, avec $V' = \{a, b, S, U, V\}$; $\Sigma = \{a, b\}$;

$$P = \begin{cases} S & \rightarrow & bS \\ S & \rightarrow & aU \\ S & \rightarrow & bV \\ V & \rightarrow & \epsilon \\ U & \rightarrow & aS \\ U & \rightarrow & bU \end{cases}$$

- Automate M.



Et si la grammaire est linéaire à gauche ?

- Les grammaires régulières sont les grammaires linéaires **à gauche** ou **à droite**
- On sait transformer une grammaire linéaire à droite en automate fini (et vice versa)
- Que faire si on a une grammaire linéaire à gauche ?

D'une grammaire linéaire à gauche vers une grammaire linéaire à droite

Transformation d'une grammaire linéaire à gauche en une grammaire linéaire à droite

Pour toute grammaire linéaire à gauche $G = \langle V, \Sigma, P, S \rangle$, il existe une grammaire linéaire à droite $G_D = \langle V_D, \Sigma, P_D, S_0 \rangle$ équivalente.

D'une grammaire linéaire à gauche vers une grammaire linéaire à droite

Transformation d'une grammaire linéaire à gauche en une grammaire linéaire à droite

Pour toute grammaire linéaire à gauche $G = \langle V, \Sigma, P, S\rangle$, il existe une grammaire linéaire à droite $G_D = \langle V_D, \Sigma, P_D, S_0\rangle$ équivalente.

Algorithme

Soit $A, B \in V \setminus \Sigma$, $a \in \Sigma$.

1. Si l'axiome S de la grammaire se trouve dans la partie droite d'une règle de P, ajouter les règles $S_0 \rightarrow S \in P$.
   Sinon, S est renommé S_0

D’une grammaire linéaire à gauche vers une grammaire linéaire à droite

Transformation d’une grammaire linéaire à gauche en une grammaire linéaire à droite

Pour toute grammaire linéaire à gauche $G = \langle V, \Sigma, P, S \rangle$, il existe une grammaire linéaire à droite $G_D = \langle V_D, \Sigma, P_D, S_0 \rangle$ équivalente.

Algorithme

Soit $A, B \in V \setminus \Sigma$, $a \in \Sigma$.

1. Si l’axiome S de la grammaire se trouve dans la partie droite d’une règle de P, ajouter les règles $S_0 \to S \in P$.
   Sinon, S est renommé S_0
2. Si $S_0 \to a \in P$, alors $S_0 \to a \in P_D$

D'une grammaire linéaire à gauche vers une grammaire linéaire à droite

Transformation d'une grammaire linéaire à gauche en une grammaire linéaire à droite

Pour toute grammaire linéaire à gauche $G = \langle V, \Sigma, P, S \rangle$, il existe une grammaire linéaire à droite $G_D = \langle V_D, \Sigma, P_D, S_0 \rangle$ équivalente.

Algorithme

Soit $A, B \in V \setminus \Sigma,\ a \in \Sigma$.

1. Si l'axiome S de la grammaire se trouve dans la partie droite d'une règle de P, ajouter les règles $S_0 \to S \in P$.
   Sinon, S est renommé S_0
2. Si $S_0 \to a \in P$, alors $S_0 \to a \in P_D$
3. Si $A \to a \in P$, alors $S_0 \to aA \in P_D$

D'une grammaire linéaire à gauche vers une grammaire linéaire à droite

Transformation d'une grammaire linéaire à gauche en une grammaire linéaire à droite

Pour toute grammaire linéaire à gauche $G = \langle V, \Sigma, P, S \rangle$, il existe une grammaire linéaire à droite $G_D = \langle V_D, \Sigma, P_D, S_0 \rangle$ équivalente.

Algorithme

Soit $A, B \in V \setminus \Sigma$, $a \in \Sigma$.

1. Si l'axiome S de la grammaire se trouve dans la partie droite d'une règle de P, ajouter les règles $S_0 \to S \in P$.
   Sinon, S est renommé S_0
2. Si $S_0 \to a \in P$, alors $S_0 \to a \in P_D$
3. Si $A \to a \in P$, alors $S_0 \to aA \in P_D$
4. Si $B \to Aa \in P$, alors $A \to aB \in P_D$

D'une grammaire linéaire à gauche vers une grammaire linéaire à droite

Transformation d'une grammaire linéaire à gauche en une grammaire linéaire à droite

Pour toute grammaire linéaire à gauche $G = \langle V, \Sigma, P, S \rangle$, il existe une grammaire linéaire à droite $G_D = \langle V_D, \Sigma, P_D, S_0 \rangle$ équivalente.

Algorithme

Soit $A, B \in V \setminus \Sigma$, $a \in \Sigma$.

1. Si l'axiome S de la grammaire se trouve dans la partie droite d'une règle de P, ajouter les règles $S_0 \to S \in P$.
   Sinon, S est renommé S_0
2. Si $S_0 \to a \in P$, alors $S_0 \to a \in P_D$
3. Si $A \to a \in P$, alors $S_0 \to aA \in P_D$
4. Si $B \to Aa \in P$, alors $A \to aB \in P_D$
5. Si $S_0 \to Aa \in P$, alors $A \to a \in P_D$

Exemple

- $G = \langle V, \Sigma, P, S \rangle$, avec $V = \{a, b, S, A\}$; $\Sigma = \{a, b\}$;

$$P = \left\{ \begin{array}{lll} S & \rightarrow & Aa \\ A & \rightarrow & b \end{array} \right.$$

Exemple

- $G = \langle V, \Sigma, P, S \rangle$, avec $V = \{a, b, S, A\}$; $\Sigma = \{a, b\}$;

$$P = \begin{cases} S & \rightarrow & Aa \\ A & \rightarrow & b \end{cases}$$

Algorithme

1. Si l'axiome S de la grammaire se trouve dans la partie droite d'une règle de P, ajouter la règle $S_0 \rightarrow S \in P$.
   Sinon, S est renommé S_0

Exemple

- $G = \langle V, \Sigma, P, S \rangle$, avec $V = \{a, b, S, A\}$; $\Sigma = \{a, b\}$;

$$P = \begin{cases} S_0 & \rightarrow & Aa \\ A & \rightarrow & b \end{cases}$$

- $G_D = \langle V_D, \Sigma, P_D, S_0 \rangle$, avec $V_D = \{a, b, S_0, A\}$; $\Sigma = \{a, b\}$;

Algorithme

1. Si l'axiome S de la grammaire se trouve dans la partie droite d'une règle de P, ajouter la règle $S_0 \rightarrow S \in P$.
   Sinon, S est renommé S_0

Exemple

- $G = \langle V, \Sigma, P, S \rangle$, avec $V = \{a, b, S, A\}$; $\Sigma = \{a, b\}$;

$$P = \left\{ \begin{array}{lcl} S_0 & \rightarrow & Aa \\ A & \rightarrow & b \end{array} \right.$$

- $G_D = \langle V_D, \Sigma, P_D, S_0 \rangle$, avec $V_D = \{a, b, S_0, A\}$; $\Sigma = \{a, b\}$;

$$P_D = \left\{ \begin{array}{lcl} S_0 & \rightarrow & bA \end{array} \right.$$

Algorithme

3. Si $A \rightarrow a \in P$, alors $S_0 \rightarrow aA \in P_D$

Exemple

- $G = \langle V, \Sigma, P, S \rangle$, avec $V = \{a, b, S, A\}$; $\Sigma = \{a, b\}$;

$$P = \begin{cases} S_0 & \rightarrow & Aa \\ A & \rightarrow & b \end{cases}$$

- $G_D = \langle V_D, \Sigma, P_D, S_0 \rangle$, avec $V_D = \{a, b, S_0, A\}$; $\Sigma = \{a, b\}$;

$$P_D = \begin{cases} S_0 & \rightarrow & bA \\ A & \rightarrow & a \end{cases}$$

Algorithme

5. Si $S_0 \rightarrow Aa \in P$, alors $A \rightarrow a \in P_D$

Exemple

- $G = \langle V, \Sigma, P, S \rangle$, avec $V = \{a, b, S, A\}$; $\Sigma = \{a, b\}$;

$$P = \begin{cases} S & \rightarrow & Aa \\ A & \rightarrow & b \end{cases}$$

- $G_D = \langle V_D, \Sigma, P_D, S_0 \rangle$, avec $V_D = \{a, b, S_0, A\}$; $\Sigma = \{a, b\}$;

$$P_D = \begin{cases} S_0 & \rightarrow & bA \\ A & \rightarrow & a \end{cases}$$

Les grammaires G et G_D sont équivalentes.

Exemple

- $G = \langle V, \Sigma, P, S \rangle$, avec $V = \{a, b, S, A\}$; $\Sigma = \{a, b\}$;

$$P = \left\{ \begin{array}{lcl} S & \rightarrow & Ab|Sb \\ A & \rightarrow & Aa|a \end{array} \right.$$

Exemple

- $G = \langle V, \Sigma, P, S \rangle$, avec $V = \{a, b, S, A\}$; $\Sigma = \{a, b\}$;

$$P = \begin{cases} S & \rightarrow & Ab|Sb \\ A & \rightarrow & Aa|a \\ S_0 & \rightarrow & S \end{cases}$$

- $G_D = \langle V_D, \Sigma, P_D, S_0 \rangle$, avec $V_D = \{a, b, S_0, A, S\}$; $\Sigma = \{a, b\}$;

Algorithme

1. Si l'axiome S de la grammaire se trouve dans la partie droite d'une règle de P, ajouter la règle $S_0 \rightarrow S \in P$.
   Sinon, S est renommé S_0

Exemple

- $G = \langle V, \Sigma, P, S \rangle$, avec $V = \{a, b, S, A\}$; $\Sigma = \{a, b\}$;

$$P = \begin{cases} S & \rightarrow & Ab|Sb \\ A & \rightarrow & Aa|a \\ S_0 & \rightarrow & S \end{cases}$$

- $G_D = \langle V_D, \Sigma, P_D, S_0 \rangle$, avec $V_D = \{a, b, S_0, A, S\}$; $\Sigma = \{a, b\}$;

$$P_D = \begin{cases} S_0 & \rightarrow & aA \end{cases}$$

Algorithme

3. Si $A \rightarrow a \in P$, alors $S_0 \rightarrow aA \in P_D$

Exemple

- $G = \langle V, \Sigma, P, S \rangle$, avec $V = \{a, b, S, A\}$; $\Sigma = \{a, b\}$;

$$P = \begin{cases} S & \rightarrow & Ab|Sb \\ A & \rightarrow & Aa|a \\ S_0 & \rightarrow & S \end{cases}$$

- $G_D = \langle V_D, \Sigma, P_D, S_0 \rangle$, avec $V_D = \{a, b, S_0, A, S\}$; $\Sigma = \{a, b\}$;

$$P_D = \begin{cases} S_0 & \rightarrow & aA \\ A & \rightarrow & bS \end{cases}$$

Algorithme

4. Si $B \rightarrow Aa \in P$, alors $A \rightarrow aB \in P_D$

Exemple

- $G = \langle V, \Sigma, P, S \rangle$, avec $V = \{a, b, S, A\}$; $\Sigma = \{a, b\}$;

$$P = \begin{cases} S & \rightarrow & Ab|Sb \\ A & \rightarrow & Aa|a \\ S_0 & \rightarrow & S \end{cases}$$

- $G_D = \langle V_D, \Sigma, P_D, S_0 \rangle$, avec $V_D = \{a, b, S_0, A, S\}$; $\Sigma = \{a, b\}$;

$$P_D = \begin{cases} S_0 & \rightarrow & aA \\ A & \rightarrow & bS|aA \\ S & \rightarrow & bS \end{cases}$$

Algorithme

4. Si $B \rightarrow Aa \in P$, alors $A \rightarrow aB \in P_D$

Exemple

- $G = \langle V, \Sigma, P, S \rangle$, avec $V = \{a, b, S, A\}$; $\Sigma = \{a, b\}$;

$$P = \begin{cases} S & \rightarrow & Ab|Sb \\ A & \rightarrow & Aa|a \\ S_0 & \rightarrow & S \end{cases}$$

- $G_D = \langle V_D, \Sigma, P_D, S_0 \rangle$, avec $V_D = \{a, b, S_0, A, S\}$; $\Sigma = \{a, b\}$;

$$P_D = \begin{cases} S_0 & \rightarrow & aA \\ A & \rightarrow & bS|aA \\ S & \rightarrow & bS|\epsilon \end{cases}$$

Algorithme

5. Si $S_0 \rightarrow Aa \in P$, alors $A \rightarrow a \in P_D$

Exemple

- $G = \langle V, \Sigma, P, S \rangle$, avec $V = \{a, b, S, A\}$; $\Sigma = \{a, b\}$;

$$P = \begin{cases} S & \rightarrow & Ab|Sb \\ A & \rightarrow & Aa|a \end{cases}$$

- $G_D = \langle V_D, \Sigma, P_D, S_0 \rangle$, avec $V_D = \{a, b, S_0, A, S\}$; $\Sigma = \{a, b\}$;

$$P_D = \begin{cases} S_0 & \rightarrow & aA \\ A & \rightarrow & bS|aA \\ S & \rightarrow & bS|\epsilon \end{cases}$$

Les grammaires G et G_D sont équivalentes.

Caractérisation des langages réguliers

Caractérisation des langages réguliers

Les langages réguliers peuvent être caractérisés de 4 façons. En utilisant :

1. Les expressions régulières
2. Les automates finis déterministes
3. Les automates finis non déterministes
4. Les grammaires régulières (linéaires à gauche ou à droite)

Caractérisation des langages réguliers

Les langages réguliers peuvent être caractérisés de 4 façons. En utilisant :

1. Les expressions régulières
2. Les automates finis déterministes
3. Les automates finis non déterministes
4. Les grammaires régulières (linéaires à gauche ou à droite)

⇒ Pour démontrer qu'un langage est régulier, il suffit donc de le décrire à l'aide de l'une de ces caractérisations

Caractérisation des langages réguliers

Les langages réguliers peuvent être caractérisés de 4 façons. En utilisant :

1. Les expressions régulières
2. Les automates finis déterministes
3. Les automates finis non déterministes
4. Les grammaires régulières (linéaires à gauche ou à droite)

⇒ Pour démontrer qu'un langage est régulier, il suffit donc de le décrire à l'aide de l'une de ces caractérisations

⇒ Pour démontrer des propriétés sur les langages réguliers, il est possible de choisir la caractérisation la mieux adaptée

Rappels sur les langages réguliers

Soient L, L_1 et L_2 trois langages réguliers. Les langages suivants sont réguliers :

- $L_1.L_2$
- $L_1 + L_2$
- L^*
- $\overline{L}$
- $L_1 \cap L_2$
- L^R (miroir de L)

Au delà des langages réguliers

Observations de base

1. Tous les langages finis sont réguliers

Observations de base

1. Tous les langages finis sont réguliers
2. Un langage non régulier comporte un nombre infini de mots

Observations de base

1. Tous les langages finis sont réguliers
2. Un langage non régulier comporte un nombre infini de mots
    - Attention ! La réciproque n'est pas vraie : Σ^* est un langage infini et régulier

Observations de base

1. Tous les langages finis sont réguliers
2. Un langage non régulier comporte un nombre infini de mots
    - Attention ! La réciproque n'est pas vraie : Σ^* est un langage infini et régulier
3. Si un langage comporte un nombre infini de mots, il n'y a pas de borne à la taille des mots du langage

Observations de base

1. Tous les langages finis sont réguliers
2. Un langage non régulier comporte un nombre infini de mots
    - Attention ! La réciproque n'est pas vraie : Σ^* est un langage infini et régulier
3. Si un langage comporte un nombre infini de mots, il n'y a pas de borne à la taille des mots du langage
4. Tout langage régulier est accepté par un automate fini qui comporte un nombre fini d'états

Observations de base

1. Tous les langages finis sont réguliers
2. Un langage non régulier comporte un nombre infini de mots
    - Attention ! La réciproque n'est pas vraie : Σ^* est un langage infini et régulier
3. Si un langage comporte un nombre infini de mots, il n'y a pas de borne à la taille des mots du langage
4. Tout langage régulier est accepté par un automate fini qui comporte un nombre fini d'états
5. Soit L un langage régulier infini, reconnu par un automate à m états. Soit $w \in L$ tel que $|w| \geq m$. Au cours de la reconnaissance de w par l'automate, il faut nécessairement passer au moins 2 fois par un même état.

Comment prouver qu'un langage n'est pas régulier ?

Comment prouver qu'un langage n'est pas régulier ?

Si le langage est fini $\Rightarrow$ langage régulier

Comment prouver qu'un langage n'est pas régulier ?

Si le langage est fini $\Rightarrow$ langage régulier

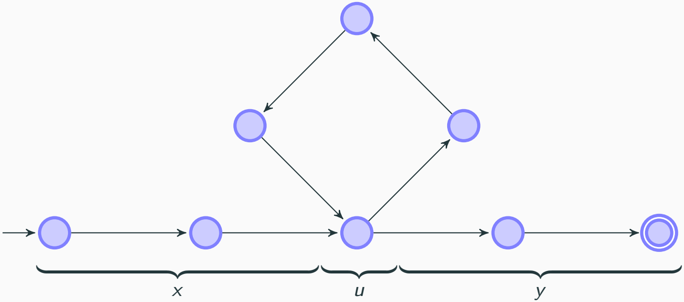
Comment prouver qu'un langage *infini* n'est pas régulier ?

Comment prouver qu'un langage n'est pas régulier ?

Si le langage est fini ⇒ langage régulier

Comment prouver qu'un langage *infini* n'est pas régulier ?

Caractéristique des langages réguliers infinis



Lemme de l'étoile (Pumping lemma)

Soit L un langage régulier infini sur l'alphabet Σ.
Alors, il existe $p \in \mathbb{N}$ tel que $\forall w \in L$ de longueur $|w| \geq p$, il existe une décomposition $w = xuy$ avec $x, u, y \in \Sigma^*$ telle que :

- $|xu| \leq p$,
- $|u| > 0$,
- $\forall n \geq 0,\ xu^n y \in L$

⚠ Ce lemme ne peut pas être utilisé pour prouver qu'un langage est régulier !

Autre formulation du lemme de l'étoile

Soit L un langage régulier infini sur l'alphabet Σ, et n le nombre d'états de l'automate fini deterministe minimal M tel que $L(M) = L$. Pour tout mot $w \in L$ tel que $|w| \geq n$, il existe une décomposition $w = xuy$ avec $x, u, y \in \Sigma^*$ telle que :

- $|xu| \leq n$,
- $|u| > 0$,
- $\forall i \geq 0,\ xu^i y \in L$

Montrer qu'un langage n'est pas régulier : exemple

Soit $\Sigma = \{a, b\}$, $L = \{a^n b^n \mid n \geq 0\}$. Supposons L régulier.

Montrer qu'un langage n'est pas régulier : exemple

Soit $\Sigma = \{a, b\}$, $L = \{a^n b^n |\ n \geq 0\}$. Supposons L régulier.

Il existe donc $p \geq 0$ tel que $w \in L$ et $|w| \geq p$, et il est possible de décomposer $w = xuy$. On sait de plus que $\forall n \geq 0$, $xu^n y \in L$.

Montrer qu'un langage n'est pas régulier : exemple

Soit $\Sigma = \{a, b\}$, $L = \{a^n b^n \mid n \geq 0\}$. Supposons L régulier.

Il existe donc $p \geq 0$ tel que $w \in L$ et $|w| \geq p$, et il est possible de décomposer $w = xuy$. On sait de plus que $\forall n \geq 0$, $xu^n y \in L$.

Soit $w = a^p b^p = xuy$. On a bien $|w| = 2p \geq p$. Il y a trois possibilités :

Montrer qu'un langage n'est pas régulier : exemple

Soit $\Sigma = \{a, b\}$, $L = \{a^n b^n \mid n \geq 0\}$. Supposons L régulier.

Il existe donc $p \geq 0$ tel que $w \in L$ et $|w| \geq p$, et il est possible de décomposer $w = xuy$. On sait de plus que $\forall n \geq 0$, $xu^n y \in L$.

Soit $w = a^p b^p = xuy$. On a bien $|w| = 2p \geq p$. Il y a trois possibilités :

1. $u \in a^*$: $w = \underbrace{a^r}_{x} \underbrace{a^s}_{u} \underbrace{a^t b^p}_{y}$, avec $r + s + t = p$ et $s > 0$.

   On a donc $\forall n \geq 0$, $xu^n y \in L$. Prenons $n = 0$. On a $a^r a^t b^p \notin L$. Contradiction.

Montrer qu'un langage n'est pas régulier : exemple

Soit $\Sigma = \{a, b\}$, $L = \{a^n b^n |\ n \geq 0\}$. Supposons L régulier.

Il existe donc $p \geq 0$ tel que $w \in L$ et $|w| \geq p$, et il est possible de décomposer $w = xuy$. On sait de plus que $\forall n \geq 0$, $xu^n y \in L$.

Soit $w = a^p b^p = xuy$. On a bien $|w| = 2p \geq p$. Il y a trois possibilités :

1. $u \in a^*$: $w = \underbrace{a^r}_{x} \underbrace{a^s}_{u} \underbrace{a^t b^p}_{y}$, avec $r + s + t = p$ et $s > 0$.
   On a donc $\forall n \geq 0$, $xu^n y \in L$. Prenons $n = 0$. On a $a^r a^t b^p \notin L$. Contradiction.
2. $u \in b^*$. Raisonnement identique.

Montrer qu'un langage n'est pas régulier : exemple

Soit $\Sigma = \{a, b\}$, $L = \{a^n b^n \mid n \geq 0\}$. Supposons L régulier.

Il existe donc $p \geq 0$ tel que $w \in L$ et $|w| \geq p$, et il est possible de décomposer $w = xuy$. On sait de plus que $\forall n \geq 0$, $xu^n y \in L$.

Soit $w = a^p b^p = xuy$. On a bien $|w| = 2p \geq p$. Il y a trois possibilités :

1. $u \in a^*$: $w = \underbrace{a^r}_{x}\underbrace{a^s}_{u}\underbrace{a^t b^p}_{y}$, avec $r + s + t = p$ et $s > 0$.
   On a donc $\forall n \geq 0$, $xu^n y \in L$. Prenons $n = 0$. On a $a^r a^t b^p \notin L$. Contradiction.
2. $u \in b^*$. Raisonnement identique.
3. $u = a^s b^t$: $w = \underbrace{a^r}_{x}\underbrace{a^s b^t}_{u}\underbrace{b^q}_{y}$, avec $r + s = t + q = p$.
   On a donc $\forall n \geq 0$, $xu^n y \in L$. Prenons $n = 2$. On a $a^r a^s b^t a^s b^t b^q \notin L$. Contradiction.

Montrer qu'un langage n'est pas régulier : exemple

Soit $\Sigma = \{a, b\}$, $L = \{a^n b^n |\ n \geq 0\}$. Supposons L régulier.

Il existe donc $p \geq 0$ tel que $w \in L$ et $|w| \geq p$, et il est possible de décomposer $w = xuy$. On sait de plus que $\forall n \geq 0$, $xu^n y \in L$.

Soit $w = a^p b^p = xuy$. On a bien $|w| = 2p \geq p$. Il y a trois possibilités :

1. $u \in a^*$: $w = \underbrace{a^r}_{x}\underbrace{a^s}_{u}\underbrace{a^t b^p}_{y}$, avec $r + s + t = p$ et $s > 0$.

   On a donc $\forall n \geq 0$, $xu^n y \in L$. Prenons $n = 0$. On a $a^r a^t b^p \notin L$. Contradiction.
2. $u \in b^*$. Raisonnement identique.
3. $u = a^s b^t$: $w = \underbrace{a^r}_{x}\underbrace{a^s b^t}_{u}\underbrace{b^q}_{y}$, avec $r + s = t + q = p$.

   On a donc $\forall n \geq 0$, $xu^n y \in L$. Prenons $n = 2$. On a $a^r a^s b^t a^s b^t b^q \notin L$. Contradiction.

Le théorème de gonflement n'est pas vérifié. Ce langage n'est donc pas un langage régulier.