

# M1 MPRI : Automates et Applications

## Cours 1

### Rappels sur les langages réguliers

`kn@lri.fr`

30 juin 2022

- 1 Introduction
- 2 Alphabets, mots et leurs opérations
- 3 Langages et leurs opérations
- 4 Langage rationnels
- 5 Automates déterministes
- 6 Automates non-déterministes
- 7 Minimisation
- 8 Concepts avancés
- 9 Conclusions

Étudier les automates et leur connexions avec les langages et la logique. Les automates ont toujours deux aspects :

- Ce sont des objets *algébriques*, définis mathématiquement dont on peut étudier les propriétés
- Ce sont des objets *calculatoires*, que l'on peut exécuter pour obtenir un résultat

Ce cours se veut un tour d'horizon de différents types d'automates (et donc de langages et de logiques), donnant les bases pour étudier ces objets plus finement en M2.

Cette UE sera évaluée par :

- Un devoir à la maison (constituant la note de contrôle continu, 40%)
- Un examen papier (constituant la note d'examen, 60%)

Chaque séance sera composée d'un cours (1h30) suivi d'un TP ou d'un TD (1h30).

En période d'enseignement distanciel, la partie TD ou TP sera remplacée par 30 minutes d'explication de l'énoncé et des premiers exercices puis 30 minutes de correction, la séance suivante avant le cours.

Nous aborderons les concepts suivants :

- 1 Rappels sur les langages réguliers de mots et les automates de mots finis
- 2 Langages réguliers d'arbres (1)
- 3 Langages réguliers d'arbres (2)
- 4 Langages  $\omega$ -réguliers et automates de Büchi
- 5 Automates et logiques
- 6 Introductions aux chaînes de Markov en temps discret (si le temps le permet).

- 1 Introduction
- 2 Alphabets, mots et leurs opérations**
- 3 Langages et leurs opérations
- 4 Langage rationnels
- 5 Automates déterministes
- 6 Automates non-déterministes
- 7 Minimisation
- 8 Concepts avancés
- 9 Conclusions

## Définition (Alphabet)

*On appelle alphabet un ensemble d'éléments. Les éléments d'un alphabet sont appelés symboles (ou parfois des lettres).*

On utilisera  $\Sigma$  pour dénoter des alphabets et  $a, b, \dots$  pour les symboles.

On se placera toujours dans le cas d'un **alphabet fini** (même si la théorie des langages et des automates peut se généraliser au cas d'alphabets infinis).



## Définition (Mot)

*Un mot sur un alphabet  $\Sigma$  est une suite de symboles. Le mot vide (constitué de 0 symbole) est dénoté par  $\epsilon$ .*

On se place, dans le début du cours dans le cas de mots **finis**, que l'on généralisera plus tard aux mots **infinis**.

On note  $\Sigma^*$  l'ensemble des mots finis sur un alphabet  $\Sigma$  (nous justifions cette notation plus tard).

On peut voir les mots comme une version formelle des des *chaînes de caractères* utilisées en informatique. Un alphabet peut alors être assimilé à un *jeu de caractères* (ASCII, Latin-1, Unicode, Idots).

## Définition

*Longueur* Soit  $w \in \Sigma^*$  un mot composé de  $n$  symboles. On appelle  $n$  la longueur du mot et on note  $|w| = n$ .

Soit  $\Sigma = \{a, b, c\}$ . L'ensemble des mots de longueurs au plus 2 sur  $\Sigma$  est :

$$\{\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc\}$$

Soit un microprocesseur 64 bits. L'ensemble des valeurs possibles pour un registre de calcul est l'ensemble des mots sur l'alphabet  $\Sigma = \{0, 1\}$  de longueur exactement 64.

## Définition (Concaténation)

*Soit  $u = a_1 a_2 \dots a_n$  et  $v = b_1 b_2 \dots b_m$  deux mots de longueur  $n$  et  $m$  respectivement. On appelle concaténation de  $u$  et  $v$  le mot  $w = a_1 a_2 \dots a_n b_1 b_2 \dots b_m$  de longueur  $n + m$ . On note  $w = uv$  (ou parfois  $u \cdot v$ ) l'opération de concaténation.*

Étant donné un alphabet  $\Sigma$ , l'ensemble  $\Sigma^*$  est un *monoïde* d'élément neutre  $\epsilon$  pour la concaténation :

Étant donné un alphabet  $\Sigma$ , l'ensemble  $\Sigma^*$  est un *monoïde* d'élément neutre  $\epsilon$  pour la concaténation :

**Élément neutre** pour tout mot  $v \in \Sigma^*$ , on a  $\epsilon v = v \epsilon = v$

**Associativité** pour tous mots  $u, v, w \in \Sigma^*$ , on a  $u(vw) = (uv)w$ .

On remarque aussi que la fonction longueur  $|\_| : \Sigma^* \rightarrow \mathbb{N}$  est un morphisme de monoïde entre  $\Sigma^*$  et  $\mathbb{N}$ , équipé de l'addition.

■  $|\epsilon| = 0$

■ Pour tout  $u, v \in \Sigma^*$ ,  $|u| + |v| = |uv|$

## Définition

*Puissance Soit  $v \in \Sigma^*$  et  $n \in \mathbb{N}$ , on définit  $v^n$  par :*

- $v^0 = \epsilon$
- $v^n = vv^{n-1}$ , pour  $n \geq 1$

Exemple : soit  $\Sigma = \{a, b\}$ , le mot  $a^4b^2$  est *aaaabb*.

## Définition (Préfixe, suffixe)

Soit  $w = uv$  trois mots de  $\Sigma^*$ . On dit que :

- $u$  est un préfixe de  $w$
- $v$  est un suffixe de  $w$

Question : Soit  $\Sigma = \{a, b\}$ . Donner tous les préfixes du mot *abbab*.



## Définition (Préfixe, suffixe)

Soit  $w = uv$  trois mots de  $\Sigma^*$ . On dit que :

- $u$  est un préfixe de  $w$
- $v$  est un suffixe de  $w$

Question : Soit  $\Sigma = \{a, b\}$ . Donner tous les préfixes du mot *abbab*.

$$\{\epsilon, a, ab, abb, abba, abbab\}$$

## Définition (facteur)

Soit  $v$  un mot de  $\Sigma^*$ . On dit que  $u$  est un facteur de  $v$  s'il existe  $u_p$  et  $u_s$  tels que  $v = u_p u u_s$ .

## Définition (sous-mot)

Soit  $v = x_1 \dots x_n$  un mot de  $\Sigma^*$ . Le mot  $u = x_{i_1} \dots x_{i_k}$  est un sous-mot de  $v$  si  $1 \leq i_1 < \dots i_k \leq n$ .

De façon informelle est sous-mot d'un mot  $v$  est  $v$  dans lequel on a effacé certains symboles. Par exemple, étant donné le mot  $abcabc$ ,  $aa$ ,  $abab$  et  $cac$  sont des sous-mots.

**Attention** à ne pas confondre facteur et sous-mot.

## Définition (miroir)

Soit  $v = x_1 \dots x_n$  un mot de  $\Sigma^*$ , on appelle mot miroir et note  $v^R$  le mot  $v^R = x_n \dots x_1$ .

Remarque : un mot  $v$  tel que  $v = v^R$  est appelé un *palindrome*.

- 1 Introduction
- 2 Alphabets, mots et leurs opérations
- 3 Langages et leurs opérations**
- 4 Langage rationnels
- 5 Automates déterministes
- 6 Automates non-déterministes
- 7 Minimisation
- 8 Concepts avancés
- 9 Conclusions

## Définition (langage)

*Un langage  $L$  sur un alphabet  $\Sigma$  est un sous-ensemble de  $\Sigma^*$ . On note  $\mathcal{P}(\Sigma^*)$  l'ensemble des langages sur  $\Sigma$ .*

On rappelle que  $\mathcal{P}(E)$  est l'ensemble des parties de l'ensemble  $E$ , i.e. l'ensemble des sous-ensembles de  $E$ . On ne confondra pas  $\Sigma^*$ , l'ensemble de tous les mots et  $\mathcal{P}(\Sigma^*)$  l'ensemble de tous les langages (c'est à dire l'ensemble de tous les ensembles de mots). Si  $\Sigma = \{a, b\}$ , alors :

- $\Sigma^* = \{\epsilon, a, b, aa, ab, ba, aaa, aab, aba, \dots\}$
- $\mathcal{P}(\Sigma^*) = \{\{\}, \{\epsilon\}, \{\epsilon, a\}, \{\epsilon, aaaaba, b\}, \dots\}$

Exemples :

- $L = \{abc, bac, acb\}$  sur  $\Sigma = \{a, b, c\}$
- $L = \{ab^n c \mid n \in \mathbb{N}\} = \{ac, abc, abbc, ab^3c, \dots\}$
- $L = \{a^n \mid n \text{ est premier}\} = \{aa, aaa, aaaaa, a^7, a^{11}, \dots\}$

Les langages étant essentiellement des ensembles, on peut définir les opérations ensemblistes traditionnelles, plus quelques opérations spécifiques aux mots.

## Définition (Union, intersection)

Soit  $L_1$  et  $L_2$  deux langages **sur un même alphabet**  $\Sigma$ . On peut définir le langage union  $L_1 \cup L_2$  et le langage intersection  $L_1 \cap L_2$  avec les opérations ensemblistes usuelles.

## Définition (Complémentaire)

Soit  $L$  un langage sur un alphabet  $\Sigma$ . On appelle complémentaire de  $L$  et on note  $\bar{L}$  l'ensemble  $\Sigma^* \setminus L$ .

## Définition (concaténation de deux langages)

Soit  $L_1$  et  $L_2$  deux langages sur un alphabet  $\Sigma$ . Le langage concaténation de  $L_1$  et  $L_2$  l'ensemble  $L_1L_2$  défini par :

$$L_1L_2 = \{w \mid \exists u \in L_1, \exists v \in L_2, w = uv\}$$

## Définition (puissance d'un langage)

Soit  $L$  un langage sur un alphabet  $\Sigma$  et  $n \in \mathbb{N}$ . Le langage  $L^n$  est défini par :

$$L^n = \{v_1 \dots v_n \mid v_i \in L \text{ pour } 1 \leq i \leq n\}$$

Attention :  $L^n \neq \{v^n \mid v \in L\}$  (exercice).



## Définition (étoile de Kleene)

Soit  $L$  un langage sur un alphabet  $\Sigma$ . L'étoile de Kleene de  $L$  est le langage  $L^*$  défini par :

$$L^* = \bigcup_{n \in \mathbb{N}} L^n = \{v_1 \dots v_n \mid n \in \mathbb{N}, v_i \in L \text{ pour } 1 \leq i \leq n\}$$

Exemple si  $L = \{aa, bb, ab\}$ , alors :

$$L^* = \{\epsilon, aa, bb, ab, aaaa, aabb, aaab, bbaa, bbbb, bbab, \dots\}$$

Stephen C. Kleene : mathématicien Américain, élève d'Alonzo Church (comme Alan Turing). Pionier de la théorie de la récursion et de l'étude des fonctions calculables.

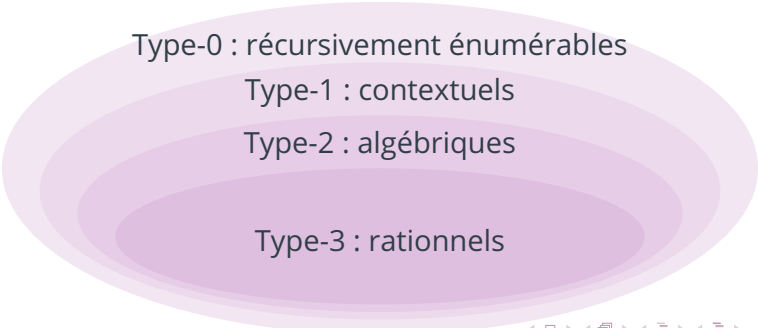
## Définition (Langage miroir)

Soit  $L$  un langage sur  $\Sigma$ . On appelle langage miroir de  $L$  et note  $L^R$  le langage :

$$L^R = \{v^R \mid v \in L\}$$

- 1 Introduction
- 2 Alphabets, mots et leurs opérations
- 3 Langages et leurs opérations
- 4 Langage rationnels**
- 5 Automates déterministes
- 6 Automates non-déterministes
- 7 Minimisation
- 8 Concepts avancés
- 9 Conclusions

Un langage est un ensemble de mots pouvant être arbitrairement compliqué. En tant qu'informaticiens, on est intéressé par les langages pour lesquels on peut *décider* si un mot appartient au langage. On appelle de tels langages des langages formels. Noam Chomsky (linguiste Américain) a établi une hiérarchie des langages formels :



The diagram illustrates Chomsky's hierarchy of formal languages using four concentric ellipses. The ellipses are nested, with the innermost being the darkest shade of purple and the outermost being the lightest. Each ellipse contains a label for a type of language, with the types ordered from the innermost to the outermost: Type-3 (rational), Type-2 (algébriques), Type-1 (contextuels), and Type-0 (rékursivement énumérables).

Type-0 : rékursivement énumérables

Type-1 : contextuels

Type-2 : algébriques

Type-3 : rationnels

## Définition (Classe des langages rationnels)

Soit  $\Sigma$  un alphabet. Soit  $\text{Rat} \subset \mathcal{P}(\Sigma^*)$  le plus petit ensemble vérifiant :

- $\emptyset \in \text{Rat}$  (langage vide)
- $\forall x \in \Sigma, \{x\} \in \text{Rat}$  (langages singletons)
- si  $L \in \text{Rat}$ , alors  $L^* \in \text{Rat}$  (stabilité par l'étoile de Kleene)
- si  $L_1, L_2 \in \text{Rat}$ ,  $L_1 \cup L_2 \in \text{Rat}$  (stabilité par union)
- si  $L_1, L_2 \in \text{Rat}$ ,  $L_1 L_2 \in \text{Rat}$  (stabilité par concaténation)

On appelle  $\text{Rat}$  la classe des langages rationnels. Un langage  $L$  est rationnel si et seulement si  $L \in \text{Rat}$ .

On a construit Rat de la façon suivante (on pose  $\Sigma = \{a, b\}$ ) :

■  $\text{Rat}_0 = \{\emptyset, \{a\}, \{b\}\}$

■  $\text{Rat}_1 =$

$$\text{Rat}_0 \cup \{\{a, b\}, \{ab\}, \{\epsilon\}, \{\epsilon, a, aa, aaa, \dots\}, \{\epsilon, b, bb, bbb, \dots\}\}$$

■  $\text{Rat}_2 = \text{Rat}_1 \cup \{a, ab\}, \{b, ab\}, \{\epsilon, a\}, \dots\}$

■ ...

Ce processus (infini) admet une limite  $\text{Rat}_\infty = \text{Rat}$ . L'existence de cette limite est assurée par le théorème de Tarski. On a créé un ensemble infini, mais qui ne contient pas *tous* les langages.

- Les langages rationnels sont clos par complémentaires, intersection et miroir.
- L'ensemble des préfixes, l'ensemble des suffixes, l'ensemble des facteurs, l'ensemble des sous-mots d'un langage rationnel est un langage rationnel.
- Le langage miroir d'un langage rationnel est un langage rationnel.

## Définition (expression régulière)

Une expression régulière  $r$  sur un alphabet  $\Sigma$  est une production finie de la grammaire suivante :

|                   |                                |
|-------------------|--------------------------------|
| $r ::= \emptyset$ | ensemble vide                  |
| $  \epsilon$      | expression vide                |
| $  x$             | $\forall x \in \Sigma$ symbole |
| $  r^*$           | étoile de Kleene               |
| $  r \mid r$      | alternative                    |
| $  r r$           | concaténation                  |

L'opérateur  $*$  est plus prioritaire que la concaténation, elle même plus prioritaire que l'alternative. Ainsi, l'expression  $a|bc^*$  soit être comprise comme  $a|(b(c^*))$ .



# Reconnaissance par une expressions régulières

La reconnaissance d'un mot par une expression régulière peut être définie inductivement sur la structure de l'expression :

- $\emptyset$  ne reconnaît aucun mot
- le mot vide est reconnu par l'expression  $\epsilon$
- le mot singleton  $a$  est reconnue par l'expression symbole  $a$
- un mot  $v$  est reconnu par  $r_1 \mid r_2$  s'il est reconnu par  $r_1$  ou par  $r_2$
- un mot  $v$  est reconnu par  $r_1 r_2$  s'il existe  $u$  et  $w$  tels que  $v = uw$  et  $u$  est reconnu par  $r_1$  et  $w$  est reconnu par  $r_2$
- un mot  $v$  est reconnu par  $r^*$  si :
  - soit  $v$  est le mot vide
  - soit  $v$  est reconnu par  $r r^*$

# Exemple de reconnaissance

On considère l'expression  $a(b|c)^*$  et le mot  $acb$ . Celui-ci est reconnu par l'expression :

$acb : a \cdot ((b|c)^*)$  (concaténation)

$a : a$  (symbole)

$cb : (b|c)^*$  (étoile)

$c : (b|c)$  (alternative)

$c : c$  (symbole)

$b : (b|c)^*$  (étoile)

$b : (b|c)$  (alternative)

$b : b$  (symbole)

$\epsilon : (b|c)^*$  (étoile)

Étant donné une expression régulière  $r$  on peut définir le langage  $L_r$  de l'ensemble des mots reconnus par  $r$  comme :

- $L_{\emptyset} = \emptyset$
- $L_{\epsilon} = \{\epsilon\}$
- $L_x = \{x\}, \forall x \in \Sigma$
- $L_{r_1|r_2} = L_{r_1} \cup L_{r_2}$
- $L_{r_1r_2} = L_{r_1}L_{r_2}$
- $L_{r^*} = (L_r)^*$

⇒ le langage d'une expression régulière est rationnel.

Les langages rationnels sont **exactement** l'ensemble des langages des expressions régulières.

On parle donc de langage rationnel ou régulier de façon interchangeable.

Certaines expressions couramment employées peuvent être construites à partir d'expressions de base :

- $r^? \equiv r \mid \epsilon$  (répétition 0 ou 1 fois)
- $r^+ \equiv r r^*$  (répétition 1 fois ou plus)
- $r^n \equiv r r \dots r$  ( $n$  fois)

- 1 Introduction
- 2 Alphabets, mots et leurs opérations
- 3 Langages et leurs opérations
- 4 Langage rationnels
- 5 Automates déterministes**
- 6 Automates non-déterministes
- 7 Minimisation
- 8 Concepts avancés
- 9 Conclusions

On dispose jusqu'à maintenant de deux types d'objets équivalents :

- Les langages rationnels : des ensembles particuliers de mots possédant des propriétés algébriques
- Les expressions régulières : une syntaxe et une sémantique permettant d'exprimer de façon compacte (en particulier finie) des ensembles de mots

On voudrait pouvoir calculer ou décider certains problèmes.

- Étant donné un mot  $w$ , appartient-il à un langage régulier  $L$ ?  
⇒ possible avec une expression régulière mais inefficace
- Étant donné un langage  $L$ , calculer son complémentaire
- Étant donné deux langages, calculer leur union (facile), intersection, concaténation ...
- Déterminer si un langage reconnaît le mot vide
- Déterminer si un langage est fini
- Déterminer si un langage est universel (est égal à  $\Sigma^*$ )
- Déterminer si deux langages sont équivalents
- Déterminer si un langage est inclus dans un autre



## Définition (Automate déterministe)

*Un automate déterministe (Deterministic Finite Automaton, DFA) est un 5-uplet  $(Q, \Sigma, \delta, q_0, F)$  où :*

- *$Q$  est un ensemble fini d'éléments appelés états*
- *$\Sigma$  est un alphabet*
- *$\delta : Q \times \Sigma \rightarrow Q$  est une fonction de transition*
- *$q_0 \in Q$  est l'état initial*
- *$F \subseteq Q$  est un ensemble d'états acceptant (ou finaux)*

## Definition (Exécution)

Soit  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$  et  $v \in \Sigma^*$ . On définit une *exécution* (ou un *run*) de  $\mathcal{A}$  pour  $v = x_1 \dots x_n$  une séquence d'états  $r_0, \dots, r_n$  in  $Q$  tels que :

- $r_0 = q_0$  (état initial)
- $r_{i+1} = \delta(r_i, x_{i+1}), \forall 0 \leq i \leq n$

De plus, si  $r_n \in F$ , on dit que l'exécution est *acceptante*.

On dit que  $\mathcal{A}$  s'arrête en  $r_n$  dans son exécution.

## Définition (Reconnaissance par un DFA)

*Soit  $v$  un mot de  $\Sigma^*$  et  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$  un DFA. On dit que  $\mathcal{A}$  reconnaît  $v$  si et seulement s'il existe une exécution acceptante de  $v$  par  $\mathcal{A}$ .*

On dit que  $\mathcal{A}$  accepte  $v$  ou qu'il reconnaît  $v$ . S'il n'existe pas d'exécution acceptante, on dit que  $\mathcal{A}$  rejette  $v$ .

On considère le mot *abba* et l'automate  $\mathcal{A}$  donné par :

$$\mathcal{A} = (\{q_0, q_1, q_2, q_\perp\}, \{a, b\}, \delta, q_0, \{q_2\})$$

$$\delta(q_0, a) = q_1 \quad \delta(q_2, a) = q_\perp$$

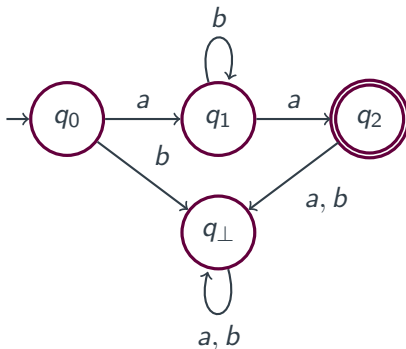
$$\delta(q_0, b) = q_\perp \quad \delta(q_2, b) = q_\perp$$

$$\delta(q_1, a) = q_2 \quad \delta(q_\perp, a) = q_\perp$$

$$\delta(q_1, b) = q_1 \quad \delta(q_\perp, b) = q_\perp$$

Le mot *abba* est reconnu par  $\mathcal{A}$ , via la séquence d'états  $q_0, q_1, q_1, q_2$

On peut représenter  $\delta$  de façon graphique comme un graphe orienté dont les sommets sont les états et les arcs sont étiquetés par des symboles de  $\Sigma$  :



## Définition (Langage d'un automate)

*Soit  $\mathcal{A}$  un DFA. Le langage  $L_{\mathcal{A}}$  est l'ensemble des mots de  $\Sigma^*$  reconnus par  $\mathcal{A}$ .*

La classe des langages *reconnaissables* est l'ensemble de tous les langages reconnaissables par un automate.

## Définition (Automate complet)

Un DFA,  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$  est complet si et seulement si  $\delta$  est une fonction totale.

## Lemme (Complétion)

Pour tout automate  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ , il existe un automate complet  $\mathcal{A}'$  tel que  $L_{\mathcal{A}} = L_{\mathcal{A}'}$ .

Preuve : si  $\mathcal{A}$  n'est pas complet, on pose  $\mathcal{A}' = (Q \cup q_{\perp}, \Sigma, \delta', q_0, F)$  et

$$\delta'(q, x) = \begin{cases} \delta(q, x) & \text{si défini} \\ q_{\perp} & \text{sinon} \end{cases}$$
$$\delta'(q_{\perp}, x) = q_{\perp}, \quad \forall x \in \Sigma$$

Un état  $q_{\perp}$  tel que  $\forall x \in \Sigma, \delta(q_{\perp}, x) = q_{\perp}$  est appelé un *état puit* (*sink state*).

Les automates incomplets sont utiles d'un point de vue pratique car ils évitent de dessiner (ou de stocker en mémoire) des états qui ne vont pas aider à la reconnaissance.

S'il n'existe aucune transition possible (dans un automate incomplet) l'exécution est bloquée (et n'avance plus et le mot est rejeté).

Dans la suite on pourra toujours supposer que l'on travaille avec des automates complets, sans perte de généralité, même si on les dessine de façon incomplète pour les exemples.



Étant donnés deux automates  $\mathcal{A}_1$  et  $\mathcal{A}_2$  on peut construire un automate  $\mathcal{A}_{1 \wedge 2}$  tel que  $L_{\mathcal{A}_{1 \wedge 2}} = L_{\mathcal{A}_1} \cap L_{\mathcal{A}_2}$ . De même on peut construire  $\mathcal{A}_{1 \vee 2}$  tel que  $L_{\mathcal{A}_{1 \vee 2}} = L_{\mathcal{A}_1} \cup L_{\mathcal{A}_2}$ .

Soit  $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$  et  $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, p_0, F_2)$ . On définit  $\mathcal{A}_{1 \wedge 2} = (Q_{1 \wedge 2}, \Sigma, \delta_{1 \wedge 2}, (q_0, p_0), F_{1 \wedge 2})$  avec :

- $Q_{1 \wedge 2} = Q_1 \times Q_2$  (chaque état est une paire d'états des automates de départ)
- $\forall (q, p) \in Q_{1 \wedge 2}, \forall x \in \Sigma, \delta_{1 \wedge 2}((q, p), x) = (\delta_1(q, x), \delta_2(p, x))$  (on suppose que les automates sont complets).
- $F_{1 \wedge 2} = \{(q, p) \mid q \in F_1 \wedge p \in F_2\}$

L'automate  $\mathcal{A}_{1 \wedge 2}$  reconnaît bien le langage  $L_{\mathcal{A}_1} \cap L_{\mathcal{A}_2}$

- Supposons que  $v$  est reconnu par  $\mathcal{A}_{1\wedge 2}$ . Par définition, il existe un run acceptant  $(r_0, s_0) \dots (r_n, s_n)$  (avec  $n = |v|$ ). Considérons la séquence  $r_0 \dots r_n$ . On montre (par récurrence sur  $n$ ) que c'est un run acceptant de  $\mathcal{A}_1$  pour  $v$ , donc  $v \in L_{\mathcal{A}_1}$ . De façon similaire,  $v \in L_{\mathcal{A}_2}$  donc  $L_{\mathcal{A}_{1\wedge 2}} \subseteq L_{\mathcal{A}_1} \cap L_{\mathcal{A}_2}$ .
- Supposons que  $v \in L_{\mathcal{A}_1} \cap L_{\mathcal{A}_2}$ . On a donc  $v \in L_{\mathcal{A}_1}$  donc il existe un run acceptant  $r_0 \dots r_n$ . De même  $v \in L_{\mathcal{A}_2}$  donc il existe un run acceptant  $s_0 \dots s_n$ . Le run  $(r_0, s_0) \dots (r_n, s_n)$  de  $\mathcal{A}_{1\wedge 2}$  est acceptant :
  - $(r_0, s_0) = (q_0, p_0)$
  - $(r_{i+1}, s_{i+1}) = (\delta(r_i, x_{i+1}), \delta(s_i, x_{i+1}))$  par construction.
  - $r_n \in F_1, s_n \in F_2$  donc  $(r_n, s_n) \in F_{1\wedge 2}$ .



On utilise une construction similaire à  $\mathcal{A}_{1\wedge 2}$  en changeant l'ensemble des états acceptants.  $\mathcal{A}_{1\vee 2} = (Q_{1\vee 2}, \Sigma, \delta_{1\vee 2}, (q_0, p_0), F_{1\vee 2})$  avec :

- $Q_{1\vee 2} = Q_1 \times Q_2$  (chaque état est une paire d'états des automates de départ)
- $\forall (q, p) \in Q_{1\vee 2}, \forall x \in \Sigma, \delta_{1\vee 2}((q, p), x) = (\delta_1(q, x), \delta_2(p, x))$  (on suppose que les automates sont complets).
- $F_{1\vee 2} = \{(q, p) \mid q \in F_1 \vee p \in F_2\}$

Il suffit que le mot soit accepté d'un côté ou de l'autre.

Un produit d'automates peut être vu comme un automate qui simule deux automates en parallèle. Pour chaque symbole, l'automate produit prend une transition dans chacun des automates.

La construction est simplifiée par le fait que les automates de départ sont supposés complets. S'il sont incomplets :

- Dans le cas de l'intersection, il faut s'arrêter si on est bloqué sur l'un des deux automates.
- Dans le cas de l'union, il faut continuer dans l'automate non bloqué si on est bloqué sur l'un des deux automates.

La définition reste naturelle pour  $\delta_{1\wedge 2}$  mais est pénible pour  $\delta_{1\vee 2}$ , car son résultat est soit une paire, soit un unique état.

Soit  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ . On définit l'automate complémentaire de  $\mathcal{A}$ ,  $\bar{\mathcal{A}}$  par :

$$\bar{\mathcal{A}} = (Q, \Sigma, \delta, q_0, Q \setminus F)$$

On a bien  $\overline{L_{\mathcal{A}}} = L_{\bar{\mathcal{A}}}$  (trivial). Encore une fois, la construction est simplifiée par le fait qu'on considère un automate complet.

Une version alternative pour calculer l'union de deux automates  $\mathcal{A}_1$  et  $\mathcal{A}_2$  est d'utiliser les lois de De Morgan pour calculer l'automate :

$$\mathcal{A}_{1 \vee 2} = \overline{\overline{\mathcal{A}_1} \cap \overline{\mathcal{A}_2}}$$

- 1 Introduction
- 2 Alphabets, mots et leurs opérations
- 3 Langages et leurs opérations
- 4 Langage rationnels
- 5 Automates déterministes
- 6 Automates non-déterministes**
- 7 Minimisation
- 8 Concepts avancés
- 9 Conclusions

Comme on l'a vu, étant donné un mot  $v$ , un DFA complet sait toujours dans quel état il doit aller depuis son état courant et la lettre suivante du mot  $v$  lors d'un run. On peut généraliser les DFA aux automates *non-deterministe* (NFA). Pour un même état courant et une même lettre, la fonction de transition renvoie plusieurs états successeurs possibles. Si *l'un de ces états* permet d'arriver à un état acceptant, alors le mot est accepté.

En pratique on peut imaginer :

- Que le NFA devienne le bon état à choisir pour la suite (oracle)
- Qu'il essaye tous les états les un à la suite des autres jusqu'à en trouver un qui donne un run acceptant (*backtracking*)
- Qu'il essaye tous les états en parallèle



## Définition (Automate non-déterministe)

*Un automate non-déterministe (Non-deterministic Finite Automaton, NFA) est un 5-uplet  $(Q, \Sigma, \delta, I, F)$  où :*

- *$Q$  est un ensemble fini d'éléments appelés états*
- *$\Sigma$  est un alphabet*
- *$\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$  est une fonction de transition*
- *$I \subseteq Q$  est un ensemble d'états initiaux*
- *$F \subseteq Q$  est un ensemble d'états acceptant (ou finaux)*

## Definition (Exécution)

Soit  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  et  $v \in \Sigma^*$ . On définit une *exécution* (ou un *run*) de  $\mathcal{A}$  pour  $v = x_1 \dots x_n$  une séquence d'états  $r_0, \dots, r_n \in Q$  tels que :

- $r_0 \in I$  (état initial)
- $r_{i+1} \in \delta(r_i, x_{i+1}), \forall 0 \leq i \leq n$

De plus, si  $r_n \in F$ , on dit que l'exécution est *acceptante*.

À l'inverse des DFA, un NFA peut avoir plusieurs run pour un même mot et même plusieurs run acceptants.

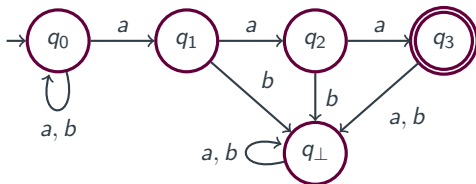
## Définition (Reconnaissance par un NFA)

*Soit  $v$  un mot de  $\Sigma^*$  et  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  un NFA. On dit que  $\mathcal{A}$  reconnaît  $v$  si et seulement s'il existe une exécution acceptante de  $v$  par  $\mathcal{A}$ .*

On dit que  $\mathcal{A}$  accepte  $v$  ou qu'il reconnaît  $v$ . S'il n'existe pas d'exécution acceptante, on dit que  $\mathcal{A}$  rejette  $v$ .

# Exemple

On considère :  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$



Pour le mot  $abaaa$ , il y a deux runs possibles :

- $q_0, q_1, q_\perp, q_\perp, q_\perp, q_\perp$
- $q_0, q_0, q_1, q_2, q_3$  qui est acceptant.

Remarque : cet automate reconnaît les mots finissant par trois  $a$ .  
Lorsque l'automate lit un  $a$ , il doit « deviner » s'il reste en  $q_0$  (car il y a des  $b$  plus loins) ou s'il passe en  $q_1$  (c'est le troisième  $a$  avant la fin).

Un NFA  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  est dit complet, si :

- $\delta$  est totale
- $\forall q \in Q, \forall x \in \Sigma, \delta(q, x) \neq \emptyset$

Sinon, le NFA est dit incomplet. Comme dans le cas des DFA, on peut compléter un NFA en rajoutant un état puit pour les transitions non définies.

## Théorème (Déterminisation)

*Soit  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  un NFA. Il existe  $\mathcal{A}_{det} = (Q_{det}, \Sigma, \delta_{det}, q_0, F_{det})$ , un DFA, tel que  $L_{\mathcal{A}} = L_{\mathcal{A}_{det}}$ .*

On donne une preuve constructive qui indique comment construire le DFA à partir du NFA.

L'idée est que le DFA va simuler « en parallèle » tous les runs du NFA. Chaque état du DFA représente donc un **ensemble des états courants** du NFA.

On considère la fonction  $\delta_{\text{det}} : \mathcal{P}_f(Q) \times \Sigma \rightarrow \mathcal{P}_f(Q)$  définie par :

$$\delta_{\text{det}}(P, x) = \{q \mid q \in \delta(p, x) \text{ for } p \in P\}$$

Considérons maintenant la suite d'ensembles suivants :

- $S_0 = \{I\}$
- $S_{i+1} = S_i \cup \bigcup_{P \in S_i} \bigcup_{x \in \Sigma} \{\delta_{\text{det}}(P, x)\}$

Alors, il existe  $l$  tel que  $S_{l+1} = S_l$  (en d'autre termes, on peut saturer les  $S_i$  jusqu'à obtenir une limite).

Cette limite existe car la suite  $S_i$  est strictement croissante (à chaque étape on ajoute l'ensemble précédant) et l'ensemble des éléments qu'on peut ajouter est fini, c'est  $\mathcal{P}_f(Q)$  de taille  $2^{|Q|}$ . Le DFA recherché est :

$$\mathcal{A}_{\text{det}} = (S_I, \Sigma, \delta_{\text{det}}, I, \{P \in S_n \mid P \cap F \neq \emptyset\})$$

On peut montrer en effet que pour tout mot  $v \in \Sigma^*$  :

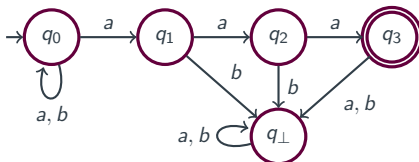
- Si  $R_0 \dots R_n$  est un run de  $\mathcal{A}_{\text{det}}$  acceptant  $v$ , alors  
 $\exists r_0 \in R_0 = I, \exists r_1 \in R_1 \dots r_n \in R_n$  tel que  $r_0 \dots r_n$  est un run de  $\mathcal{A}$  acceptant  $v$ .
- Si  $r_0 \dots r_n$  est un run de  $\mathcal{A}$  acceptant  $v$ , alors  
 $\exists R_0 = I \in S_I, \dots, \exists R_n \in S_I$  tel que  $\forall i, r_i \in R_i$  et  $R_0 \dots R_n$  est un run de  $\mathcal{A}_{\text{det}}$  acceptant  $v$ .



Les deux directions se font par récurrence sur  $n$  la longueur du mot, avec  $l$  comme cas de base en utilisant les propriétés de  $\delta_{\text{det}}$ .

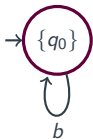
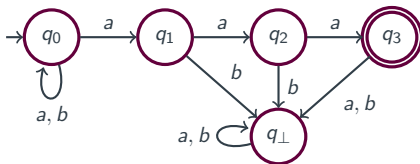
# Exemple

On prend l'automate  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$

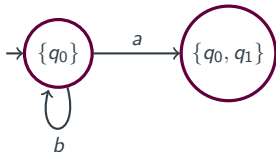
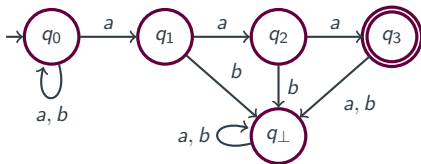


# Exemple

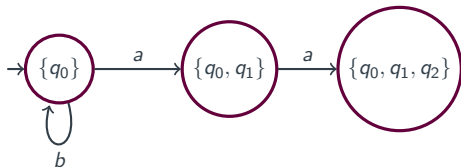
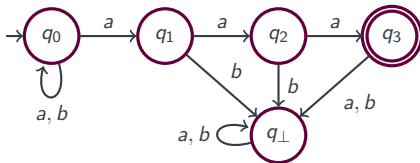
On prend l'automate  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$



On prend l'automate  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$

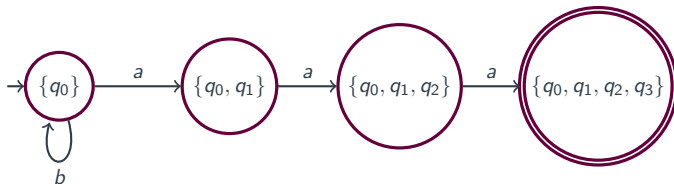
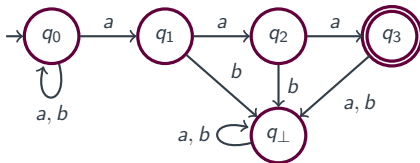


On prend l'automate  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$



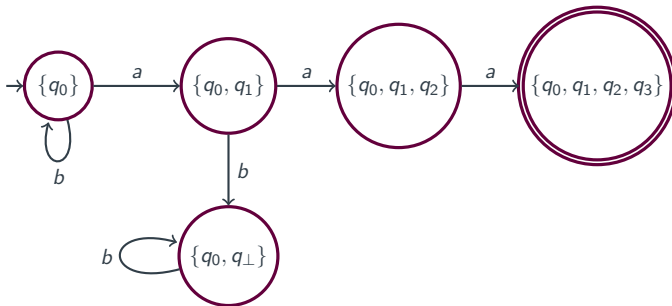
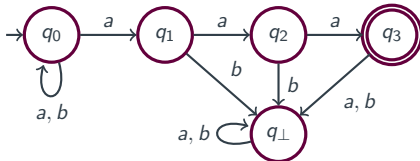
# Exemple

On prend l'automate  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$



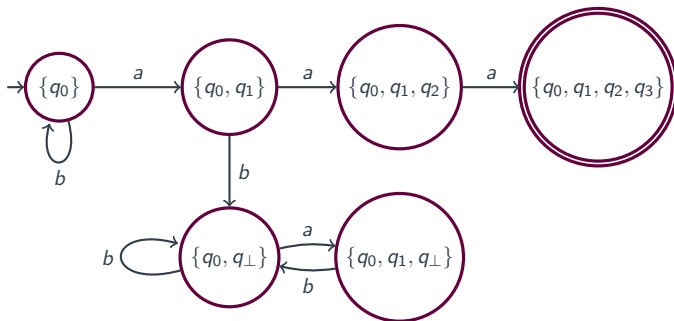
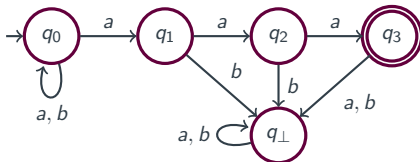
# Exemple

On prend l'automate  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$



# Exemple

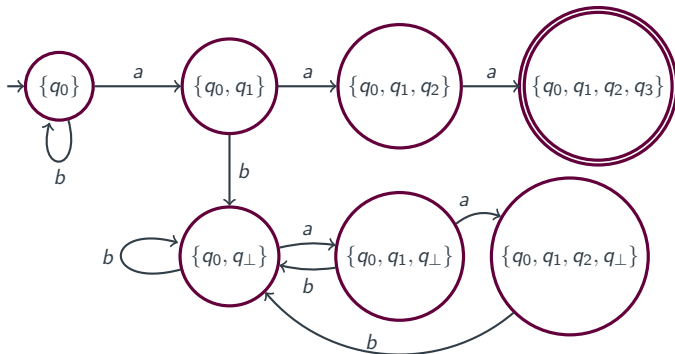
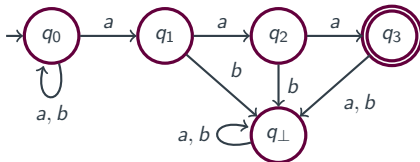
On prend l'automate  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$



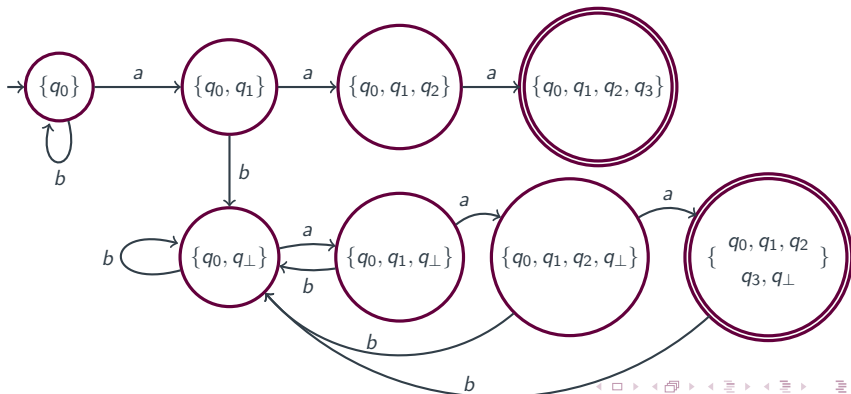
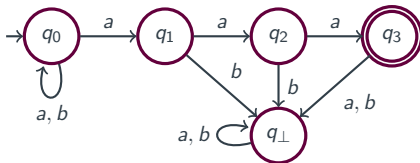


# Exemple

On prend l'automate  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$

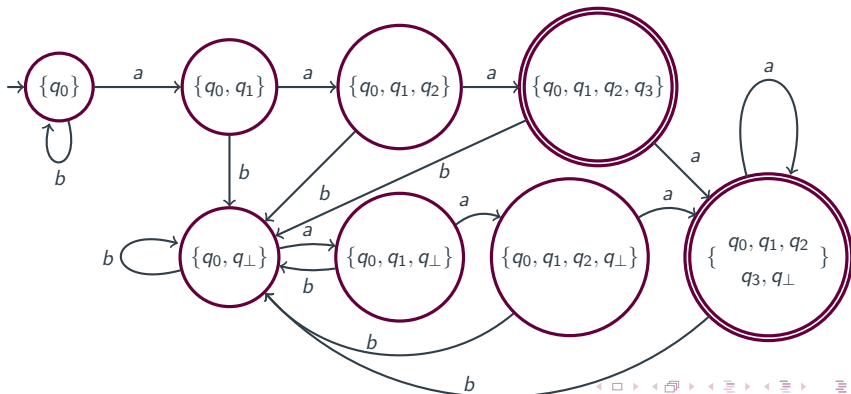
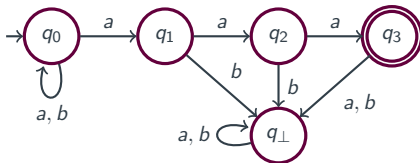


On prend l'automate  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$



# Exemple

On prend l'automate  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$



Comme on l'a vu, les langages reconnaissables par des DFA sont les même que les langages reconnaissables par des NFA (et on les appelle simplement « langages reconnaissables » ).

Remarque : formellement on prouve l'équivalence ainsi :

- $Rec(DFA) \subseteq Rec(NFA)$  : Trivial (un DFA est un cas particulier de NFA)
- $Rec(NFA) \subseteq Rec(DFA)$  : Car on peut déterminer : pour tout langage reconnue par un automate non déterministe, on peut reconnaître le même langage avec un automate déterministe.

Comme DFA et NFA sont équivalents on s'attend (à raison) que les NFA aient les même propriété de clôture avec l'union, l'intersection et le complémentaire. Il faut cependant être prudent.

Soit  $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, l_1, F_1)$  et  $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, l_2, F_2)$  deux NFA.

L'automate  $\mathcal{A}_{1 \vee 2} = (Q_1 \cup Q_2, \Sigma, \delta_{1 \vee 2}, l_1 \cup l_2, F_1 \cup F_2)$  reconnaît le langage  $L_{\mathcal{A}_1} \cup L_{\mathcal{A}_2}$ , où  $\delta_{1 \vee 2}$  est défini par :

$$\delta_{1 \vee 2}(q, x) = \begin{cases} \delta_1(q, x) & \text{si } q \in Q_1 \\ \delta_2(q, x) & \text{si } q \in Q_2 \end{cases}$$

Le calcul est efficace (linéaire en la taille des automates :

$$O(|Q_1| + |\delta_1| + |Q_2| + |\delta_2|))$$

On peut appliquer la construction d'automate produit des DFA aux NFA (pour le cas de l'intersection). Soit  $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, l_1, F_1)$  et  $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, l_2, F_2)$  deux NFA. L'automate produit  $\mathcal{A}_{1 \wedge 2} = (Q_1 \times Q_2, \Sigma, \delta_{1 \wedge 2}, l_1 \times l_2, F_1 \times F_2)$  reconnaît le langage  $L_{\mathcal{A}_1} \cap L_{\mathcal{A}_2}$ , où  $\delta_{1 \vee 2}$  est défini par :

$$\delta_{1 \wedge 2}((r, s), x) = \delta_1(r, x) \times \delta_2(s, x)$$

La méthode simple consistant à inverser les états acceptants et non acceptants d'un DFA ne fonctionne pas dans le cas des NFA. Pourquoi? Dans le cas des DFA :

- Si le run pour un mot  $v$  dans un DFA est acceptant, alors ce même run dans le DFA complémentaire est forcément non acceptant (et vice-versa).

Dans le cas des NFA, si on se contente d'inverser les états acceptants/non acceptant :

- Si un run pour un mot  $v$  dans un NFA est acceptant, alors ce même run dans le NFA « complémentaire » est non acceptant. Mais il est possible qu'il existe aussi un run non-acceptant (et même plusieurs) et ceux la deviennent alors acceptants!

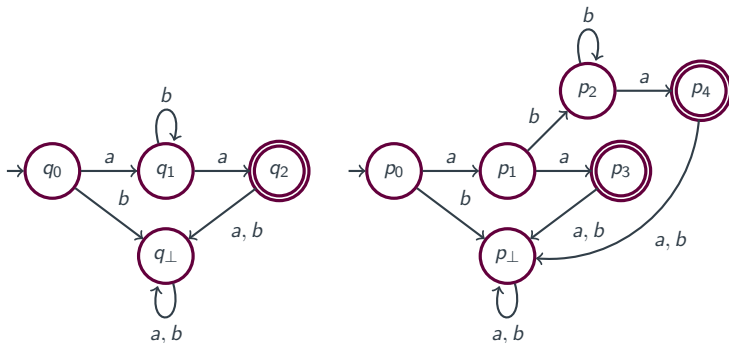
⇒ déterminer puis prendre le complémentaire du DFA.



- 1 Introduction
- 2 Alphabets, mots et leurs opérations
- 3 Langages et leurs opérations
- 4 Langage rationnels
- 5 Automates déterministes
- 6 Automates non-déterministes
- 7 Minimisation**
- 8 Concepts avancés
- 9 Conclusions

# Automate minimal

On remarque que deux automates distincts (ici des DFA) peuvent avoir le même langage :



Intuitivement l'automate de gauche « se comporte comme »  $ab^*a$ , celui de droite comme  $a(a|b^+a)$ .

## Théorème (Théorème de Myhill-Nerode)

Soit  $L$  un langage. Étant donné un mot  $w \in \Sigma^*$ , on définit l'ensemble  $Suite_L(w) = \{t \in \Sigma^* \mid wt \in L\}$ . Soit  $\equiv_L$  la relation d'équivalence entre mots définie par  $u \equiv_L v$  si et seulement si  $Suite_L(u) = Suite_L(v)$ .

- Le langage  $L$  est reconnaissable si et seulement si le nombre de classes d'équivalence de  $\equiv_L$  est fini. Appelons ce nombre  $k$ .
- Dans ce cas, il existe un unique DFA à  $k$  états reconnaissant  $L$ , et il n'existe aucun DFA ayant un nombre  $n < k$  d'états reconnaissant  $L$ .

On ne va pas faire la preuve, mais expliquer le théorème et donner l'algorithme minimisation.

- $\text{Suite}_L(w) = \{t \in \Sigma^* \mid wt \in L\}$  est l'ensemble des suffixes permettant de compléter  $w$  pour obtenir un mot de  $L$ . Par exemple, pour le langage fini  $L = \{aab, bab, aaa, baa\}$ ,  $\text{Suite}_L(a) = \{ab, aa\}$ .
- Deux mots  $u$  et  $v$  sont équivalents ( $u \equiv_L v$ ) si et seulement si  $\text{Suite}_L(u) = \text{Suite}_L(v)$ . Pour notre exemple :  $a \equiv_L b$ . Cela signifie que si on considère  $u$  et  $v$  comme des débuts de mots reconnus par un automate, alors leur suite est reconnue par la même portion d'automate.

# Algorithme de Moore (minimisation)

**Entrée** un DFA  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$

**Sortie** un DFA  $\mathcal{A}_{\min} = (Q_{\min}, \Sigma, \delta_{\min}, q'_0, F_{\min})$  où  $|Q_{\min}| \leq |Q|$

$$\mathbb{P} = \{F, Q \setminus F\}$$

répéter:

$$\mathbb{P}' := \mathbb{P}$$

pour tout  $C \in \mathbb{P}$ :

pour tout  $x \in \Sigma$ :

$$\mathbb{C} := \emptyset$$

pour tout  $D \in \mathbb{P}$ :

$$\mathbb{C} := \mathbb{C} \cup \{\{q \in C \mid \delta(q, x) \in D\}\}$$

$$\mathbb{P} := \mathbb{P} \setminus \{C\} \cup \mathbb{C}$$

tant que  $\mathbb{P} \neq \mathbb{P}'$

Renvoyer l'automate  $(\mathbb{P}, \Sigma, \delta_{\min}, \{q_0\}, \{F\})$  où  $\delta_{\min}$  est défini par :

$$\delta_{\min}(C, x) = [\delta(q, x)]_{\mathbb{P}}, \text{ en choisissant } q \in C$$

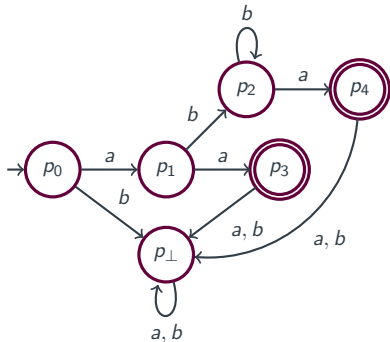
$[r]_{\mathbb{P}}$  est l'unique ensemble  $C$  de  $\mathbb{P}$  contenant  $r$ .

L'idée de l'algorithme est de grouper dans les même paquets les états de l'automate « qui font le même travail », *i.e.* qui reconnaissent le même suffixe. Chacun des paquets obtenus (à la fin de l'algorithme) représente un état de l'automate minimal : tous les états de l'automate original faisaient le même travail, ils ont donc été fusionnés.

L'algorithme commence par créer deux paquets « évidents » : états acceptants (qui reconnaissent  $\epsilon$ ) et états non-acceptants. Pour chaque paquet  $C$  trouvé jusqu'à présent, pour chaque lettre, on découpe  $C$  en des sous-ensembles (disjoints) qui font arriver leurs états dans les même paquets.

# Exemple

On cherche à minimiser :



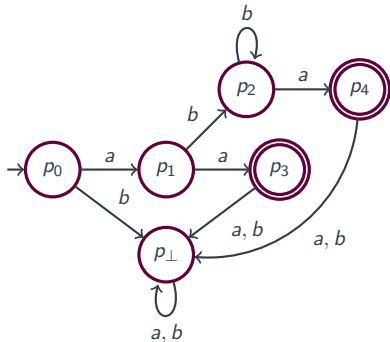
$\mathbb{P}_0 \mid \{\{p_0, p_1, p_2, p_\perp\}, \{p_3, p_4\}\}$

acceptants vs. non-acceptants



# Exemple

On cherche à minimiser :



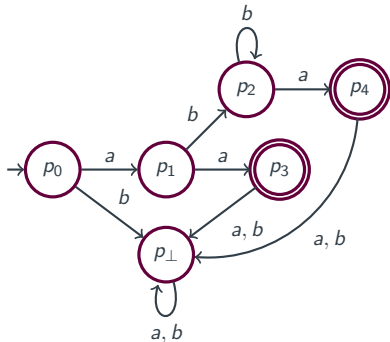
|                |  |
|----------------|--|
| $\mathbb{P}_0$ | $\{\{p_0, p_1, p_2, p_\perp\}, \{p_3, p_4\}\}$     |
| $\mathbb{P}_1$ | $\{\{p_0, p_\perp\}, \{p_1, p_2\}, \{p_3, p_4\}\}$ |

acceptants vs. non-acceptants

$a$  distingue  $p_1, p_2$  et  $p_0, p_\perp$

# Exemple

On cherche à minimiser :



|                |  |
|----------------|--|
| $\mathbb{P}_0$ | $\{\{p_0, p_1, p_2, p_\perp\}, \{p_3, p_4\}\}$         |
| $\mathbb{P}_1$ | $\{\{p_0, p_\perp\}, \{p_1, p_2\}, \{p_3, p_4\}\}$     |
| $\mathbb{P}_2$ | $\{\{p_0\}, \{p_\perp\}, \{p_1, p_2\}, \{p_3, p_4\}\}$ |

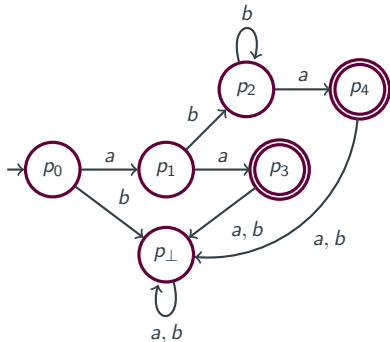
acceptants vs. non-acceptants

$a$  distingue  $p_1, p_2$  et  $p_0, p_\perp$

$a$  distingue  $p_0$  et  $p_\perp$

# Exemple

On cherche à minimiser :



|                |  |
|----------------|--|
| $\mathbb{P}_0$ | $\{\{p_0, p_1, p_2, p_\perp\}, \{p_3, p_4\}\}$         |
| $\mathbb{P}_1$ | $\{\{p_0, p_\perp\}, \{p_1, p_2\}, \{p_3, p_4\}\}$     |
| $\mathbb{P}_2$ | $\{\{p_0\}, \{p_\perp\}, \{p_1, p_2\}, \{p_3, p_4\}\}$ |
| $\mathbb{P}_3$ | rien ne change   |

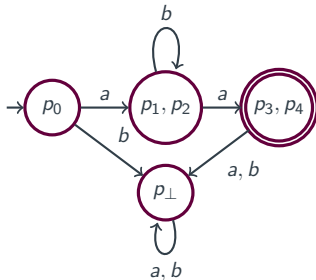
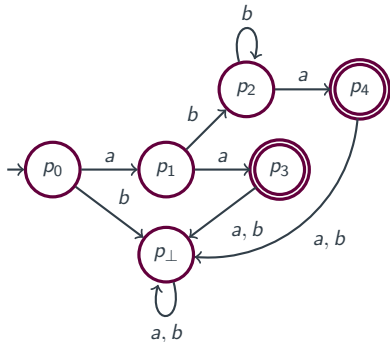
acceptants vs. non-acceptants

$a$  distingue  $p_1, p_2$  et  $p_0, p_\perp$

$a$  distingue  $p_0$  et  $p_\perp$

# Exemple

On cherche à minimiser :



|                |  |
|----------------|--|
| $\mathbb{P}_0$ | $\{\{p_0, p_1, p_2, p_\perp\}, \{p_3, p_4\}\}$         |
| $\mathbb{P}_1$ | $\{\{p_0, p_\perp\}, \{p_1, p_2\}, \{p_3, p_4\}\}$     |
| $\mathbb{P}_2$ | $\{\{p_0\}, \{p_\perp\}, \{p_1, p_2\}, \{p_3, p_4\}\}$ |
| $\mathbb{P}_3$ | rien ne change   |

acceptants vs. non-acceptants

$a$  distingue  $p_1, p_2$  et  $p_0, p_\perp$

$a$  distingue  $p_0$  et  $p_\perp$

- 1 Introduction
- 2 Alphabets, mots et leurs opérations
- 3 Langages et leurs opérations
- 4 Langage rationnels
- 5 Automates déterministes
- 6 Automates non-déterministes
- 7 Minimisation
- 8 Concepts avancés**
- 9 Conclusions

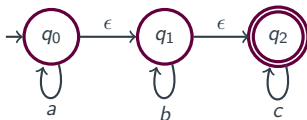
On introduit une généralisation des NFA, les  $\epsilon$ NFA qui peuvent passer spontanément d'un état à un autre sans consommer de symbole du mot que l'on cherche à reconnaître.

## Définition (Automate non-déterministe avec transitions vides)

*Un automate non-déterministe avec transitions vides ( $\epsilon$ NFA) est un 5-uplet  $(Q, \Sigma, \delta, I, F)$  où :*

- *$Q$  est un ensemble fini d'éléments appelés états*
- *$\Sigma$  est un alphabet*
- *$\delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow \mathcal{P}_f(Q)$  est une fonction de transition*
- *$I \subseteq Q$  est un ensemble d'états initiaux*
- *$F \subseteq Q$  est un ensemble d'états acceptant (ou finaux)*

Une difficulté pour définir l'exécution des  $\epsilon$ NFA est la présence de transitions vides. Si on considère l'automate ci-dessous :



En partant de l'état initial  $q_0$ , le mot  $cc$  est reconnu. En effet, on peut passer de  $q_0$  à  $q_1$  (en ne lisant aucune lettre), puis de  $q_1$  à  $q_2$ , puis enfin boucler 2 fois sur  $q_2$ . On a donc un run  $q_0, q_1, q_2, q_2$  (de taille 4) alors que le mot lu est de taille 2. (jusqu'à présent, on avait toujours des run de taille  $|w| + 1$  pour un mot  $w$ ). Il faut donc compter  $\epsilon$  comme un « symbole ».

## Definition (Exécution)

Soit  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  et  $v \in \Sigma^*$ . On définit une *exécution* (ou un *run*) de  $\mathcal{A}$  pour  $v$  s'il existe une séquence de mots  $v_1 \dots v_n$ ,  $v_i \in \Sigma \cup \{\epsilon\}$ , une séquence d'états  $r_0, \dots, r_n \in Q$  tels que :

- $r_0 \in I$  (état initial)
- $r_{i+1} \in \delta(r_i, x_{i+1})$ ,  $\forall 0 \leq i \leq n$

Dans l'exemple précédant, le « mot » reconnu est en fait  $\epsilon \in CC$



## Définition (Reconnaissance par un $\epsilon$ NFA)

*Soit  $v$  un mot de  $\Sigma^*$  et  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  un  $\epsilon$ NFA. On dit que  $\mathcal{A}$  reconnaît  $v$  si et seulement s'il existe une exécution acceptante de  $v$  par  $\mathcal{A}$ .*

Étant donné un  $\epsilon$ NFA de transition  $\delta$ , on appelle  $\epsilon$ -fermeture de  $q$  l'ensemble  $E(q)$  calculé de la façon suivante :

- $E_0(q) = \{q\}$
- $E_i(q) = E_{i-1} \cup \bigcup_{q' \in E_{i-1}} \delta(q', \epsilon)$

En s'arrêtant dès que  $E_i$  ne change plus (ce qui se produit forcément, car dans le pire cas  $E_i(q)$  contient tous les états de l'automate).

Étant donné un ensemble d'état  $S$ , on note  $E(S) = \{E(q) \mid q \in S\}$   
Ce processus permet d'obtenir pour chaque état tout ceux qui sont atteignable par une suite de transitions vides.

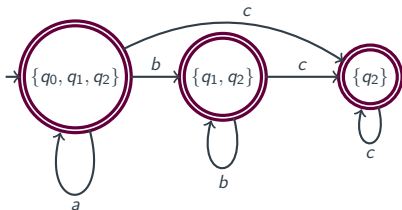
## Théorème (Élimination des transitions vides)

Pour tout  $\epsilon$ NFA,  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ , il existe un NFA  $\mathcal{A}_N = (Q_N, \Sigma, \delta_N, I_N, F_N)$  tel que  $L_{\mathcal{A}} = L_{\mathcal{A}_N}$ .

On construit l'automate ainsi :

- $Q_N = \{E(q) \mid q \in Q\}$ , chaque état de  $\mathcal{A}_N$  est un ensemble d'états atteignables par une suite de transitions vides.
- $I_N = \{E(q) \mid q \in I\}$  et  $F_N = \{P \mid P \in Q_N, P \cap F \neq \emptyset\}$ .
- $\delta_N(P, x) = \bigcup_{p \in E(P)} \delta(p, x)$

Pour notre exemple :



Remarque : la construction peut rendre des états  $q$  de la forme



inaccessibles, il faut les supprimer.

Pourquoi donc introduire les  $\epsilon$ NFA s'ils sont équivalents aux NFA?

Pourquoi donc introduire les  $\epsilon$ NFA s'ils sont équivalents aux NFA?

Il permettent de construire, de façon *compositionnelle* un  $\epsilon$ NFA à partir d'une expression régulière.


Pourquoi donc introduire les  $\epsilon$ NFA s'ils sont équivalents aux NFA?

Il permettent de construire, de façon *compositionnelle* un  $\epsilon$ NFA à partir d'une expression régulière.

C'est la construction de Thompson.

Cette construction prend en entrée une expression régulière  $r$  et produit un  $\epsilon$ NFA reconnaissant le même langage.

Tous les automates générés ont un unique état d'entrée  $q_{in}$  et un unique état de sortie  $q_{out}$  :

■  $\emptyset$  (expression vide qui ne reconnaît rien) : 

■  $\epsilon$  (reconnaît le mot vide) : 

■  $a$  (reconnaît un symbole) : 

Ce sont les trois seuls cas de base.



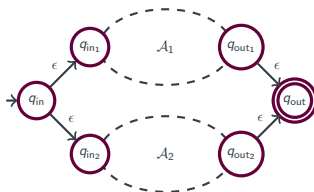
# Construction de Thompson (2)

Étant donnés deux expressions  $r_1$  et  $r_2$  ayant pour automates  $\mathcal{A}_1$  et  $\mathcal{A}_2$  :

■  $r_1 \cdot r_2$  (concaténation)



■  $r_1 \mid r_2$  (alternative)

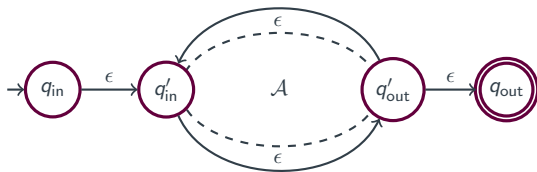


Attention, dans  $\mathcal{A}_1$

et  $\mathcal{A}_2$  les états  $q_{out1}$  et  $q_{out2}$  sont acceptants, mais il le sont plus dans l'automate résultat.

# Construction de Thompson (3)

Pour l'étoile de Kleene, étant donné une expression  $r$  et son automate  $\mathcal{A}$  :



Une construction plus complexe est due à Glushkov. Elle donne directement un NFA à partir de  $r$ . Il a été démontré que retirer les transitions vides de l'automate de Thompson donne l'automate de Glushkov.

L'utilité est double :

- On démontre que  $\text{Reg} \subseteq \text{Rec}$ , pour chaque expression régulière, on a un automate
- Donne une procédure d'évaluation pour les expressions régulières (on la transforme en automate que l'on évalue)

L'utilité est double :

- On démontre que  $\text{Reg} \subseteq \text{Rec}$ , pour chaque expression régulière, on a un automate
- Donne une procédure d'évaluation pour les expressions régulières (on la transforme en automate que l'on évalue)

Et dans l'autre sens?

Il existe plusieurs algorithmes pour traduire un automate en expression régulière. On présente l'algorithme de Brzozowski et McCluskey, appelé algorithme de réduction des états. On introduit pour cela une classe d'automates particuliers, les automates généralisés.

## Définition (Automate généralisé)

Un automate généralisé (GFA) est un 5-uplet  $(Q, \Sigma, \delta, q_{in}, q_{out})$  où :

- $Q$  est un ensemble fini d'éléments appelés états
- $\Sigma$  est un alphabet
- $\delta : Q \times \text{Reg}(\Sigma) \rightarrow \mathcal{P}_f(Q)$  est une fonction de transition
- $I \subseteq Q$  est un ensemble d'états initiaux
- $F \subseteq Q$  est un ensemble d'états acceptant (ou finaux)

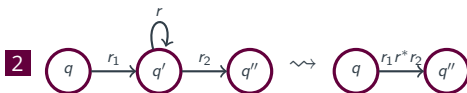
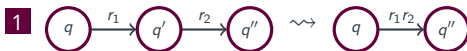
$\text{Reg}(\Sigma)$  est l'ensemble des expressions régulières sur  $\Sigma$ . On a donc des automates non déterministes, étiquetés par des expressions régulières, et possédant un unique état initial et un unique état final. On note que n'importe quel NFA se convertit trivialement en GFA.

Étant donné un NFA, on le considère comme un GFA dont les transitions sont étiquetées par les expressions régulières  $\epsilon$  ou symbole. Au besoin on rajoute un unique état initial et un unique état acceptant en les reliant par des transitions vides.

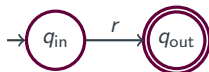
On applique alors deux règles :

réduction des transitions : 

réduction des états : 2 cas possibles :



À chaque étape, on réduit soit le nombre d'états soit le nombre de transitions. L'algorithme termine lorsque l'automate est réduit à :



Où  $r$  est une expression régulière ayant le même langage que l'automate initial. Attention, l'expression obtenue peut être complexe et en général, si on traduit une expression en automate (avec Thompson) puis de nouveau en expression on n'obtient pas l'expression de départ.

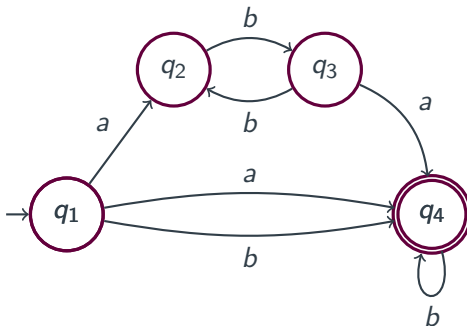


L'utilité est double :

- On démontre que  $\text{Rec} \subseteq \text{Reg}$  et donc que  $\text{Rec} = \text{Reg} = \text{Rat}$
- Les automates généralisés ne sont pas plus expressifs que les NFA car on peut toujours faire :  $\text{GFA} \rightarrow r \rightarrow \text{NFA}$

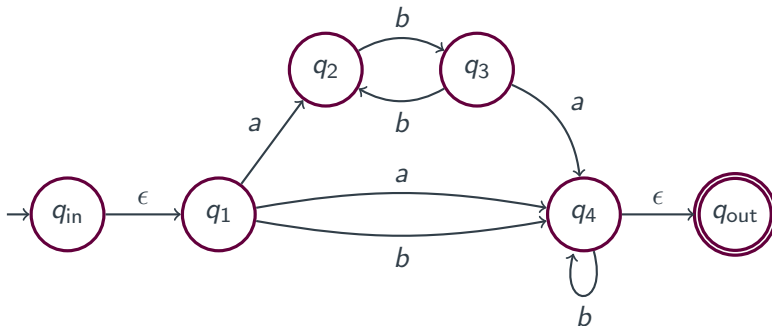
# Exemple

On considère le *NFA* suivant :



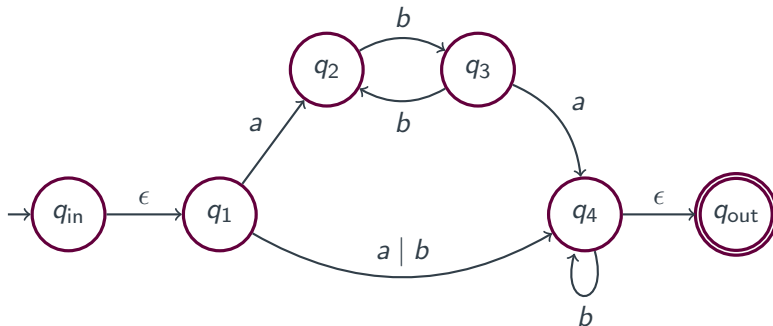
# Exemple

On considère le *NFA* suivant :



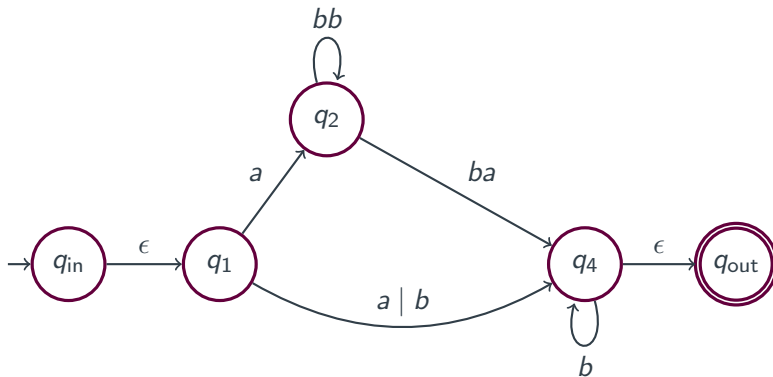
# Exemple

On considère le *NFA* suivant :



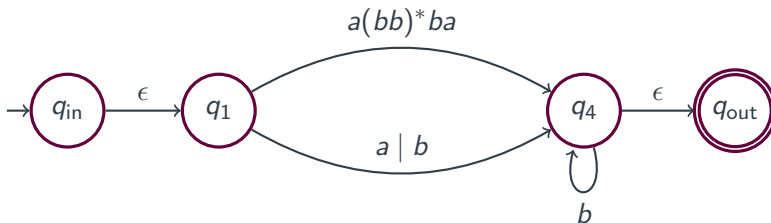
# Exemple

On considère le *NFA* suivant :



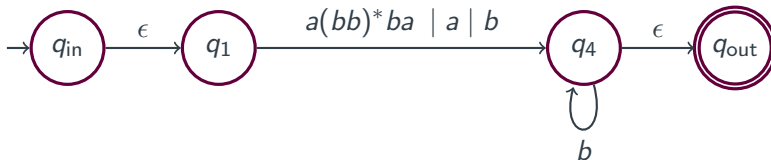
# Exemple

On considère le *NFA* suivant :



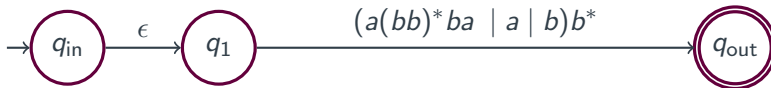
# Exemple

On considère le *NFA* suivant :



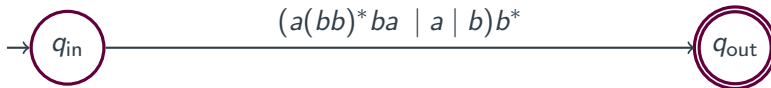
# Exemple

On considère le *NFA* suivant :





On considère le *NFA* suivant :



Étant donné un langage régulier, on a *des algorithmes* pour décider les problèmes suivants :

- calculer l'union, l'intersection, le complémentaire, le miroir (exercice) d'un langage (en passant par les automates)
- décider s'il reconnaît uniquement le langage vide (accessibilité d'un état final)
- décider s'il reconnaît  $\Sigma^*$  (= son complémentaire est vide)
- décider si  $L_A \subseteq L_B$

Étant donné un langage régulier, on a *des algorithmes* pour décider les problèmes suivants :

- calculer l'union, l'intersection, le complémentaire, le miroir (exercice) d'un langage (en passant par les automates)
- décider s'il reconnaît uniquement le langage vide (accessibilité d'un état final)
- décider s'il reconnaît  $\Sigma^*$  (= son complémentaire est vide)
- décider si  $L_A \subseteq L_B \Leftrightarrow L_A \setminus L_B = \emptyset \Leftrightarrow L_A \cap \overline{L_B} = \emptyset$
- décider si  $L_A = L_B$  (soit double inclusion, soit déterminer puis minimiser et comparer les automates)

Que manque-t-il ?

On n'aura pas d'algorithme :(

## Théorème

*Le problème de savoir si un langage algébrique est régulier est indécidable.*

Mais tout n'est pas perdu.

On a un moyen mathématique de montrer qu'un langage *n'est pas régulier*.

## Lemme (Lemme de la pompe)

*Soit  $L$  un langage régulier sur un alphabet  $\Sigma$ . Il existe un entier  $p \geq 1$  tel que pour tout mot de  $w \in L$  tel que  $|w| \geq p$ , il existe  $x, y, z \in \Sigma^*$  tels que :*

- $w = xyz$
- $y \neq \epsilon$
- $|xy| \leq p$
- $\forall n \geq 0, xy^n z \in L$

# Lemme de la pompe (2)

Que signifie ce lemme?

Que signifie ce lemme? Pour tout langage régulier, il existe une longueur de mot  $p$  telle que pour produire des mots plus grands que  $p$ , on est obligé « d'avoir une boucle dans l'automate » ou « une étoile dans l'expression régulière ». Et donc qu'une sous partie du mot (celle qui est sous l'étoile) peut être « pompée » c'est à dire répétée un nombre arbitraire de fois tout en restant un mot du langage.

# Lemme de la pompe (3)

Comment on s'en sert?



Comment on s'en sert? C'est une condition *nécessaire* des langages réguliers. Donc :

- ⇒ si un langage est régulier, alors il est pompable, autrement dit (contraposée) si un langage *n'est pas pompable* alors il *n'est pas régulier*.
- ⇏ si un langage est pompable, il n'est pas forcément régulier.

# Lemme de la pompe (4)

On veut montrer qu'un langage *n'est pas régulier*, on procède par l'absurde.

- On suppose que  $L$  est régulier
- Donc il existe bien un entier  $p \geq 1$  pour ce langage
- On choisit un mot  $w = xyz$  du langage (avec  $y \neq \epsilon$  et  $|xy| \leq p$ )
- On trouve une  $n$  tel que  $xy^n z \notin L$
- Contradiction ( $xy^n z$  devrait être dans  $L$ ) donc  $L$  n'est pas régulier

On ne peut rien supposer sur  $p$ , par contre on peut choisir  $w$  et  $n$  comme on veut.

# Lemme de la pompe (exemple)

Montrons :  $L = \{a^n b^n \mid n \in \mathbb{N}\}$  n'est pas régulier. Supposons le régulier :

- il existe une longueur de mot  $p$  (qu'on ne connaît pas) qui valide le lemme de la pompe.
- on choisit  $w = a^p b^p$ . C'est bien un mot de longueur plus grande que  $p$  (il est de longueur  $2p$ )
- on choisit de le découper en  $\underbrace{\epsilon}_x \underbrace{a^p}_y \underbrace{b^p}_z$ . On a bien  $|xy| \leq p$ .
- le mot  $xy^2z = a^{2p}b^p$  n'appartient pas à  $L$  (car  $p \geq 1$ ), or, il devrait, car on respecte toutes les hypothèses du lemme de la pompe. Contradiction,  $L$  n'est pas régulier.

- 1 Introduction
- 2 Alphabets, mots et leurs opérations
- 3 Langages et leurs opérations
- 4 Langage rationnels
- 5 Automates déterministes
- 6 Automates non-déterministes
- 7 Minimisation
- 8 Concepts avancés
- 9 Conclusions**

Ça fait beaucoup de choses, que faut-il retenir?

- avec le temps, tout!
  - mais dans l'immédiat : les équivalences entre DFA et NFA, entre régulier et NFA, la minimisation, la façon de reconnaître un mot avec un NFA/DFA.
- ⇒ *toutes* ces propriétés se transposent aux arbres et aux automates d'arbres, avec des preuves similaires.
- Les automates d'arbres sont moins faciles à dessiner, donc c'est bien d'avoir une intuition de ce qui se passe sur les langages de mots.