Théorie des langages

Julien BERNARD

Université de Franche-Comté – UFR Sciences et Technique

Licence Informatique – 3è année

Première partie

Introduction – Alphabets, mots, langages

Plan de ce cours

Plan

Votre enseignant
Qui suis-je ?

Qui suis-je ?

Julien BERNARD, Maître de Conférence (enseignant-chercheur)
`julien.bernard@univ-fcomte.fr`, Bureau 426C

Enseignement

- Responsable du semestre 1 (Starter) de la licence Informatique
- Cours : Bases de la programmation (L1), Publication scientifique (L1), Algorithmique (L2), Sécurité (L3), Théorie des Langages (L3), Analyse syntaxique (L3)

Recherche

Optimisation dans les réseaux de capteurs

Plan

UE Théorie des Langages
Organisation

Équipe pédagogique

- Julien Bernard : CM, TD (julien.bernard@univ-fcomte.fr)
- Karla Breschi, Serge Moulin, Lydia Yataghene : TP

Volume

- Cours : 12 x 1h30
- TD : 12 x 1h30
- TP : 12 x 1h30

Évaluation

- deux devoirs surveillés
- un projet en TP

UE Théorie des Langages

Comment ça marche ?

Mode d'emploi

1. Prenez des notes ! Posez des questions ! N'attendez pas du tout-cuit !
2. Comprendre plutôt qu'apprendre
3. Le but de cette UE n'est pas d'avoir une note !

Niveau d'importance des transparents

	trivial	pour votre culture
$\star$	intéressant	pour votre compréhension
$\star\star$	important	pour votre savoir
$\star\star\star$	vital	pour votre survie

Note : les contrôles portent sur *tous* les transparents !

UE Théorie des Langages

Contenu pédagogique

Objectif

Comprendre la théorie et les outils de la théorie des langages

- Alphabets, mots, langages
- Grammaires
- Langages réguliers
- Automates d'états finis
- Expressions régulières
- Langages algébriques
- Automates à piles
- Machines de Turing

UE Théorie des Langages
Bibliographie


P. Wolper
Introduction à la calculabilité.
2006, Dunod


P. Séébold.
Théorie des automates.
2009, Vuibert


J.M. Autebert
Théorie des langages et des automates.
1994, Masson


J. Hopcroft, J. Ullman
Introduction to Automata Theory, Languages and Compilation
1979, Addison-Wesley

Plan

Bref historique

Historique

- Notion de langage formel, Noam Chomsky, début des années 1950
  - Origine dans la linguistique
  - Étude des langues naturelles
  - Traitement automatique (exemple : traduction)

→ Hiérarchie de Chomsky, 1956

- Classification des langages selon leur pouvoir d'expression

→ Outil important en informatique !

Étude des langages formels
Niveaux d'études

Niveaux d'études des langages

Deux points de vue :

- Du **locuteur**. Le problème est de savoir engendrer les phrases (mots) du langage $\rightarrow$ Notion de **grammaire**
- De l'**auditeur**. Le problème est de savoir reconnaître les phrases (mots) du langage $\rightarrow$ Notion de **reconnaisseur** (automate)

Étude des langages formels
Buts

Buts de l'étude des langages

- Évaluer et classer les langages
  - Caractériser les langages
  - Trouver des grammaires
  - Trouver des reconnaisseurs
- Développer des algorithmes
  - Compilation des langages informatiques
  - Reconnaissance de la parole

Plan

Alphabet

Définition (Alphabet)

Un **alphabet** est un ensemble fini de symboles appelés **lettres**.

Remarque

Un alphabet est aussi appelé **vocabulaire**.

Exemples (Alphabet)

- $A = \{0, 1\}$
- $\Sigma = \{a, b, c\}$
- $\Theta = \{if, then, else, a, b\}$
- $F = \{\rightarrow, \leftarrow, \uparrow, \downarrow\}$

Mot

Définition (Mot)

Un **mot** sur l'alphabet Σ est une suite *finie* et *ordonnée*, éventuellement vide, de lettres de Σ. Le **mot vide** est toujours noté ε.

Exemples (Mot)

aba et *abbaccb* sont deux mots sur l'alphabet $\Sigma = \{a, b, c\}$.

Notation

Soit w un mot constitué de k lettres sur l'alphabet Σ, on notera :

$$w = w_1 \cdots w_k$$

Longueur d'un mot ⋆⋆

Définition (Longueur d'un mot)

La **longueur d'un mot** w est le nombre de lettres constituant le mot w. Elle est notée $|w|$. Le mot vide a une longueur de 0.

Exemples (Longueur d'un mot)

$|aba| = 3$, $|abbaccb| = 7$, $|\varepsilon| = 0$

Remarque

De nombreuses propriétés sur les mots se montreront par récurrence sur la longueur des mots.

Nombre d'occurrences d'une lettre dans un mot ⋆⋆

Définition (Nombre d'occurrence)

Le **nombre d'occurrences** d'une lettre a dans un mot w est le nombre de fois où la lettre a apparaît dans le mot w. Elle est notée $|w|_a$.

Exemples (Nombre d'occurrences)

$|aba|_a = 2$, $|abbaccb|_b = 3$, $|baab|_c = 0$

Ensemble des mots

Définition (Ensemble des mots)

L'**ensemble des mots non-vide** sur un alphabet Σ est noté Σ^+.

$$\Sigma^+ = \{w = w_1 \dots w_n, n > 0\}$$

L'**ensemble des mots** sur un alphabet Σ est noté Σ^*.

$$\Sigma^* = \{\varepsilon\} \cup \Sigma^+ = \{w = w_1 \dots w_n, n \geq 0\}$$

Produit de mots ⋆⋆

Définition (Produit de mots)

Soient Σ un alphabet et $x, y \in \Sigma^*$ deux mots sur l'alphabet Σ de longueur respective n et m. On définit le **produit** w de x et y noté $x \cdot y$ par :

$$w = x \cdot y = x_1 \dots x_n \cdot y_1 \dots y_m = x_1 \dots x_n y_1 \dots y_m$$

Remarque

Le produit est aussi appelé **concaténation**.

Exemple (Produit de mots)

$aba \cdot ab = abaab$

Monoïde $(\Sigma^*, \cdot, \varepsilon)$

Proposition (Monoïde $(\Sigma^*, \cdot, \varepsilon)$)

Σ^* *munie de l'opération produit d'élément neutre* ε *est un monoïde.*

Démonstration.

- Le produit est une loi interne : $x \cdot y \in \Sigma^*$
- Le produit est associatif : $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
- ε est l'élément neutre du produit : $x \cdot \varepsilon = \varepsilon \cdot x = x$

Remarque

Le produit n'est pas commutatif : $x \cdot y \neq y \cdot x$

Puissance d'un mot

Définition (Puissance d'un mot)

Soient Σ un alphabet, et $w \in \Sigma^*$. La **puissance** d'un mot, noté w^n est définie par :

$$w^n = \begin{cases} \varepsilon & \text{si } n = 0 \\ w \cdot w^{n-1} & \text{si } n > 0 \end{cases}$$

Exemple (Puissance d'un mot)

Soit $\Sigma = \{a, b\}$ et $w = abb$, alors :

- $w^0 = \varepsilon$
- $w^1 = w = abb$
- $w^2 = w \cdot w = abbabb$
- $w^3 = w \cdot w^2 = abbabbabb$

Égalité de mots

Définition (Égalité de mots)

Deux mots sont **égaux** si et seulement s'ils sont de même longueur et s'ils ont des lettres identiques à des positionnements identiques.
Soit Σ un alphabet et $x = x_1 \cdots x_n, y = y_1 \cdots y_m \in \Sigma^*$, alors :

$$x = y \iff n = m \text{ et } \forall i \in [1, n], x_i = y_i$$

Plan

Langage

Définition (Langage)

Un **langage** L sur un alphabet Σ est un sous-ensemble de Σ^*. C'est un ensemble de mots sur l'alphabet Σ.

Exemples (Langage)

Soit $\Sigma = \{a, b\}$ un alphabet :

- $L = \varnothing$ est un langage appelé **langage vide**
- $L = \{\varepsilon\}$ est un langage appelé **langage unité**
- $L = \{a, ab, abb\}$ est un langage
- $L = \{a^n, n \geq 0\}$ est un langage

Égalité de langage

Définition (Égalité de langage)

Deux langages L_1 et L_2 sont **égaux**, noté $L_1 = L_2$ si et seulement si $L_1 \subseteq L_2$ et $L_2 \subseteq L_1$.

Remarque

Cette définition donne une manière de prouver l'égalité de deux langages.

Complémentaire d'un langage ⋆⋆

Définition (Complémentaire d'un langage)

Soit L un langage sur l'alphabet Σ, le **complémentaire** de L, noté $\overline{L}$ est le langage défini par :

$$\overline{L} = \{w \in \Sigma^*, w \notin L\}$$

Exemple (Complémentaire d'un langage)

Soit $L = \{w \in \Sigma^*, |w| \equiv 0 \mod 2\}$, alors, $\overline{L} = \{w \in \Sigma^*, |w| \equiv 1 \mod 2\}$

Union de langages

Définition (Union de deux langages)

Soient L_1 et L_2 deux langages sur l'alphabet Σ, l'**union** de L_1 et L_2, notée $L_1 \cup L_2$ (ou parfois $L_1 + L_2$) est définie par :

$$L_1 \cup L_2 = \{w \in \Sigma^*, w \in L_1 \text{ ou } w \in L_2\}$$

Propriétés

L'union est :

- Associative : $L_1 \cup (L_2 \cup L_3) = (L_1 \cup L_2) \cup L_3$
- Commutative : $L_1 \cup L_2 = L_2 \cup L_1$
- Idempotente : $L \cup L = L$
- Élément neutre $\varnothing$: $L \cup \varnothing = \varnothing \cup L = L$

Intersection de langages

Définition (Intersection de deux langages)

Soient L_1 et L_2 deux langages sur l'alphabet Σ, l'**intersection** de L_1 et L_2, notée $L_1 \cap L_2$ est définie par :

$$L_1 \cap L_2 = \{w \in \Sigma^*, w \in L_1 \text{ et } w \in L_2\}$$

Propriétés

L'intersection est :

- Associative : $L_1 \cap (L_2 \cap L_3) = (L_1 \cap L_2) \cap L_3$
- Commutative : $L_1 \cap L_2 = L_2 \cap L_1$
- Idempotente : $L \cap L = L$
- Élément neutre Σ^* : $L \cap \Sigma^* = \Sigma^* \cap L = L$

Différence de langages

Définition (Différences de deux langages)

Soient L_1 et L_2 deux langages sur l'alphabet Σ, la **différence** de L_1 et L_2, notée $L_1 \setminus L_2$ est définie par :

$$L_1 \setminus L_2 = \{w \in \Sigma^*, w \in L_1 \text{ et } w \notin L_2\}$$

Proposition (Différence de deux langages)

$$L_1 \setminus L_2 = L_1 \cap \overline{L_2}$$

Opérations ensemblistes sur les langages

Exemples (Opérations ensemblistes sur les langages)

Soit $\Sigma = \{a, b\}$ un alphabet.
Soit $L_1 = \{a, ab\}$ et $L_2 = \{ab, ba\}$ deux langages sur Σ.

- $L_1 \cup L_2 = L_2 \cup L_1 = \{a, ab, ba\}$
- $L_1 \cap L_2 = L_2 \cap L_1 = \{ab\}$
- $L_1 \setminus L_2 = \{a\}$
- $L_2 \setminus L_1 = \{ba\}$

Produit de langages

Définition (Produit de deux langages)

Soient Σ un alphabet et $L_1, L_2 \subseteq \Sigma^*$ deux langages sur l'alphabet Σ. On définit le **produit** L de L_1 et L_2 noté $L_1.L_2$ par :

$$L = L_1.L_2 = \{u_1 \cdot u_2, u_1 \in L_1, u_2 \in L_2\}$$

Remarque

- Le produit est aussi appelé **concaténation**.
- Attention à ne pas confondre avec le produit cartésien (noté $\times$).

Produit de langages

Exemple (Produit de deux langages)

Soit $\Sigma = \{a, b\}$ un alphabet.
Soit $L_1 = \{\varepsilon, a, ab\}$ et $L_2 = \{b, ba\}$ deux langages sur Σ.

- $L_1.L_2 = \{b, ba, ab, aba, abb, abba\}$
- $L_2.L_1 = \{b, ba, bab, ba, baa, baab\} = \{b, ba, bab, baa, baab\}$

$\rightarrow$ $L_1.L_2 \neq L_2.L_1$

Propriétés du produit de langage ⋆⋆⋆

Proposition (Distributivité du produit par rapport à l'union)

Le produit de langage est distributif par rapport à l'union. Soient Σ un alphabet et $L_1, L_2, L_3 \subseteq \Sigma^*$, alors :

$$L_1.(L_2 \cup L_3) = (L_1.L_2) \cup (L_1.L_3) \text{ et } (L_1 \cup L_2).L_3 = (L_1.L_3) \cup (L_2.L_3)$$

Remarque importante

Le produit de langage n'est pas distributif par rapport à l'intersection. Plus précisément, on a :

$$L_1.(L_2 \cap L_3) \subseteq (L_1.L_2) \cap (L_1.L_3) \text{ et } (L_1 \cap L_2).L_3 \subseteq L_1.L_3 \cap L_2.L_3$$

Propriétés du produit de langage ⋆

Distributivité du produit par rapport à l'union.

1. Soit $w \in L_1.(L_2 \cup L_3)$, montrons que $w \in (L_1.L_2) \cup (L_1.L_3)$.
   $\exists w_1 \in L_1, w' \in L_2 \cup L_3, w = w_1 \cdot w'$. Donc $w' \in L_2$ ou $w' \in L_3$.
   Si $w' \in L_2$, alors $w = w_1 \cdot w' \in L_1.L_2$.
   Si $w' \in L_3$, alors $w = w_1 \cdot w' \in L_1.L_3$.
   Donc, $w \in (L_1.L_2) \cup (L_1.L_3)$.
   Donc $L_1.(L_2 \cup L_3) \subseteq (L_1.L_2) \cup (L_1.L_3)$.
2. Soit $w \in (L_1.L_2) \cup (L_1.L_3)$, montrons que $w \in L_1.(L_2 \cup L_3)$.
   $w \in L_1.L_2$ ou $w \in L_1.L_3$.
   Si $w \in L_1.L_i, i \in \{2,3\}$ alors $\exists w_1 \in L_1, w_i \in L_i, w = w_1 \cdot w_i$.
   Donc $w \in L_1.(L_2 \cup L_3)$.
   Donc, $(L_1.L_2) \cup (L_1.L_3) \subseteq L_1.(L_2 \cup L_3)$

Donc $(L_1.L_2) \cup (L_1.L_3) = L_1.(L_2 \cup L_3)$ □

Puissance d'un langage

Définition (Puissance d'un langage)

Soient Σ un alphabet, et $L \subseteq \Sigma^*$. La **puissance** d'un langage, noté L^n est définie par :

$$L^n = \begin{cases} \{\varepsilon\} & \text{si } n = 0 \\ L.L^{n-1} & \text{si } n > 0 \end{cases}$$

Itéré d'un langage ⋆⋆

Définition (Itéré d'un langage)

L'**itéré strict** d'un langage L, noté L^+, est défini par :

$$L^+ = \bigcup_{i>0} L^i$$

L'**itéré** d'un langage L, appelé aussi l'**étoile de Kleene**, noté L^*, est défini par :

$$L^* = \bigcup_{i\geq 0} L^i = \{\varepsilon\} \cup L^+$$

Proposition

$$L^+ = L.L^* = L^*.L$$

Plan

Appartenance d'un élément à un ensemble ⋆

Appartenance d'un élément à un ensemble

Soit X un ensemble défini par une propriété P_X :

$$X = \{x, P_X(x)\}$$

Pour prouver qu'un élément y appartient à l'ensemble X, il suffit de montrer qu'il satisfait la propriété P_X.

Inclusion d'un ensemble dans un autre ⋆

Inclusion d'un ensemble dans un autre

Soit X et Y deux ensembles. Pour prouver que l'ensemble X est inclus dans l'ensemble Y, on cherche à montrer :

$$\forall x \in X, x \in Y$$

- On considère un élément $x \in X$, c'est-à-dire qu'il satisfait P_X.
- On montre que l'élément x est dans Y, c'est-à-dire qu'il satisfait P_Y.
- On conclue.

Égalité de deux ensembles ⋆

Égalité de deux ensembles

Soit X et Y deux ensembles. Pour prouver que les ensembles X et Y sont égaux, on chercher à montrer :

$$X \subseteq Y \text{ et } Y \subseteq X$$

Deuxième partie

Grammaires

Plan de ce cours

3 Grammaires
- Définitions
- Réécriture, dérivation et langage engendré
- Arbre de dérivation
- Hiérarchie de Chomsky

Plan

Grammaire
Principe

Principe d'une grammaire

Ensemble de règles pour générer les mots du langage

- On part d'un symbole spécial appelé l'**axiome** ou la **source**
- On applique des **règles de réécriture**
  - Remplacement d'une séquence de symboles par une autre séquence
- On génère des mots

Grammaire
Exemple introductif

Exemple introductif

- On considère la phrase suivante :

  `la vieille dame regarde la petite fille`

→ Peut-on construire une grammaire qui génère cette phrase ?

- Alphabet : { `la`, `vieille`, `petite`, `dame`, `fille`, `regarde` }
- Structure de la phrase :
  - Un groupe sujet (article, adjectif,nom)
  - Un verbe
  - Un groupe complément d'objet (article, adjectif, nom)

Grammaire
Exemple introductif

Règles de production

1. $\langle \text{Phrase} \rangle \rightarrow \langle \text{Sujet} \rangle \langle \text{Verbe} \rangle \langle \text{Complément} \rangle$
2. $\langle \text{Sujet} \rangle \rightarrow \langle \text{Groupe Nominal} \rangle$
3. $\langle \text{Complément} \rangle \rightarrow \langle \text{Groupe Nominal} \rangle$
4. $\langle \text{Groupe Nominal} \rangle \rightarrow \langle \text{Article} \rangle \langle \text{Nom} \rangle$
5. $\langle \text{Groupe Nominal} \rangle \rightarrow \langle \text{Article} \rangle \langle \text{Adjectif} \rangle \langle \text{Nom} \rangle$
6. $\langle \text{Article} \rangle \rightarrow$ `la`
7. $\langle \text{Nom} \rangle \rightarrow$ `dame` | `fille`
8. $\langle \text{Adjectif} \rangle \rightarrow$ `vieille` | `petite`
9. $\langle \text{Verbe} \rangle \rightarrow$ `regarde`

Grammaire

Définition (Grammaire)

Une **grammaire** G est un quadruplet (N, T, S, R) où :

- N est l'ensemble des **symboles non-terminaux**
- T est l'ensemble des **symboles terminaux**
- $V = N \cup T$ est le **vocabulaire** de la grammaire $(N \cap T = \varnothing)$
- $S \in N$ est l'**axiome**
- R est un ensemble de **règles de production** de la forme :

$$\alpha \to \beta, \alpha \in V^+, \beta \in V^*$$

Grammaire

Remarques

- Généralement, les symboles non-terminaux sont écrits en majuscules
- Généralement, les symboles terminaux sont écrits en minuscules
- $\alpha \to \beta$ signifie que α peut être remplacé par β

Notation

Plusieurs règles qui ont le même membre gauche peuvent être factorisées.

- $\alpha \to \beta$
- $\alpha \to \gamma$

est équivalent à :

- $\alpha \to \beta \mid \gamma$

Exemples de grammaire

Exemple (Grammaire)

$G = (\{E, T, F\}, \{+, \times, (,), a, b\}, E, R)$ avec R :

- $E \to E + T \mid T$
- $T \to T \times F \mid F$
- $F \to (E) \mid a \mid b$

Exemples de grammaire

Exemple (Grammaire)

$G = (\{S\}, \{a, b\}, S, R)$ avec R :

- $S \to aSa \mid SbS \mid \varepsilon$

Exemples de grammaire

Exemple (Grammaire)

$G = (\{S, B, C\}, \{a, b, c\}, S, R)$ avec R :

- $S \to aSBC \mid \varepsilon$
- $CB \to BC$
- $aB \to ab$
- $bB \to bb$
- $bC \to bc$
- $cC \to cc$

Plan

Réécriture

Définition (Réécriture)

Soit $G = (N, T, S, R)$ une grammaire, $u \in V^+, v \in V^*$.
La **réécriture** de u en v par G, notée $u \to v$, est définie par :

- $u = \alpha \cdot u' \cdot \beta, v = \alpha \cdot v' \cdot \beta$ avec $\alpha, \beta \in V^*$
- $(u' \to v') \in R$

Réécriture

Exemple (Réécriture)

Soit $G = (\{E, T, F\}, \{+, \times, (,), a, b\}, E, R)$ avec R :

- $E \to E + T \mid T$
- $T \to T \times F \mid F$
- $F \to (E) \mid a \mid b$

On peut effectuer des réécritures jusqu'au mot $a + b \times a$:

$$E \to E + T \to T + T \to F + T \to F + T \times F$$
$$\to a + T \times F \to a + F \times F \to a + F \times a \to a + b \times a$$

Dérivation

Définition (Dérivation)

Soit $G = (N, T, S, R)$ une grammaire, $u \in V^+, v \in V^*$.
La **dérivation** de u en v par G, notée $u \to^* v$, est définie par :

- $\exists k > 0$ et $\exists w_0, \ldots, w_k \in V^*$, avec $w_0 = u$ et $w_k = v$
- $w_i \to w_{i+1}$ pour tout $0 \leq i < k$

Exemple (Dérivation)

Avec la grammaire précédente, on a :

$$E \to^* F + T \times F \to^* a + b \times a$$

Mot engendré par une grammaire ⋆⋆

Définition (Mot engendré par une grammaire)

Soit $G = (N, T, S, R)$ une grammaire,
$u \in T^*$ (symboles terminaux) est un **mot engendré par la grammaire** G s'il peut être dérivé depuis l'axiome S, c'est-à-dire $S \rightarrow^* u$.

Exemple (Mot engendré par une grammaire)

$a + b \times a$ est un mot engendré par la grammaire G précédente.

Langage engendré par une grammaire

Définition (Langage engendré par une grammaire)

Soit $G = (N, T, S, R)$ une grammaire,
Le **langage engendré par la grammaire** G, noté $\mathcal{L}(G)$, est l'ensemble des mots engendrés par G.

$$\mathcal{L}(G) = \{u \in T^*, S \rightarrow^* u\}$$

Langage engendré par une grammaire

Exemple (Langage engendré par une grammaire)

Soit $G = (\{S\}, \{a, b\}, S, R)$ avec R :

- $S \to aXbX \mid bXaX \mid \varepsilon$

Le langage $\mathcal{L}(G)$ engendré par G est le langage L des mots qui contiennent autant de a que de b.

$$\mathcal{L}(G) = \{u \in T^*, |u|_a = |u|_b\}$$

Dérivation la plus à gauche ⋆⋆

Définition (Dérivation la plus à gauche)

Soit $G = (N, T, S, R)$ une grammaire, et $w \in \mathcal{L}(G)$, la dérivation $S \rightarrow^* w$ est la **dérivation la plus à gauche** si, à chaque étape de la dérivation, c'est le symbole non-terminal le plus à gauche qui est dérivé.

Plan

Arbre de dérivation

Définition (Arbre de dérivation)

Soit $G = (N, T, S, R)$ une grammaire et $w \in \mathcal{L}(G)$.
L'**arbre de dérivation** du mot w est un arbre où :

- la racine est S
- les feuilles sont étiquetées par des éléments terminaux de T
- les nœuds sont étiquetés par des éléments non-terminaux de N
- si un nœud est étiqueté Y et ses fils sont étiquetés $Z_1, \ldots, Z_k$ dans cet ordre, alors il existe une règle $Y \to Z_1 \ldots Z_k$ dans R
- la lecture des feuilles de gauche à droite donne le mot w

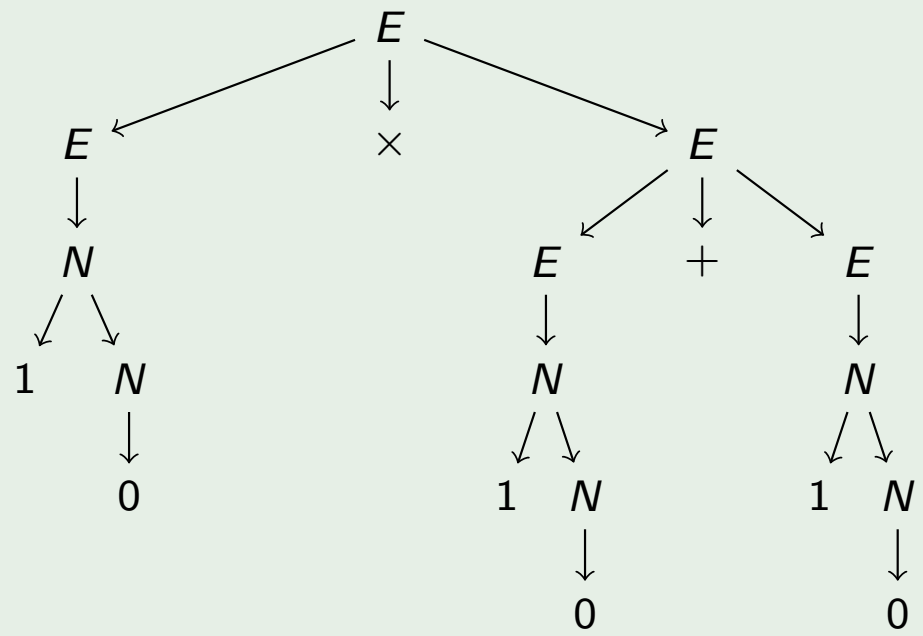
Arbre de dérivation

Exemple (Arbre de dérivation)

Soit $G = (\{E, N\}, \{+, \times, 0, 1\}, E, R)$ avec R :

- $E \to E + E \mid E \times E \mid N$
- $N \to 0 \mid 1 \mid 0N \mid 1N$

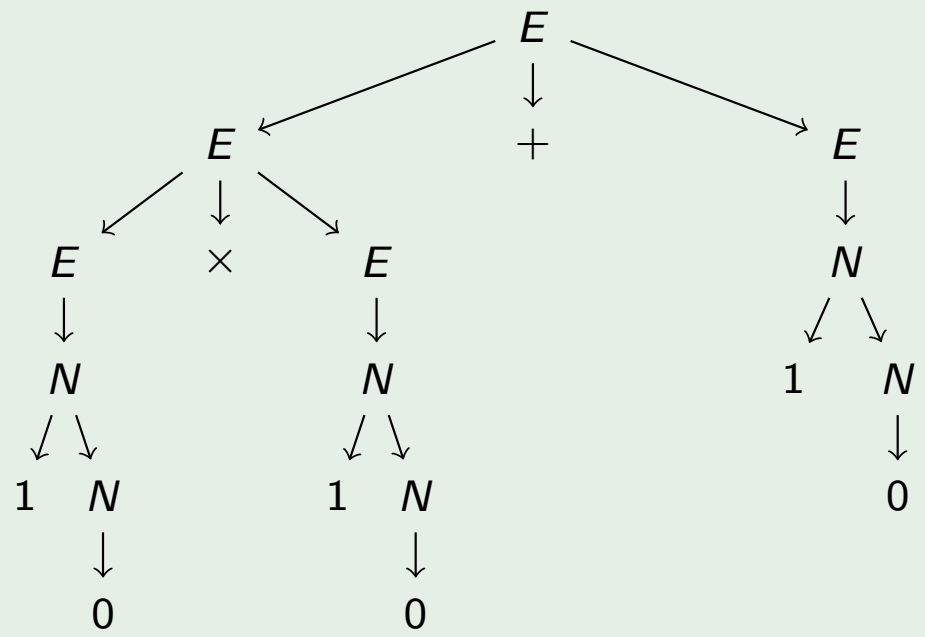
Le mot $10 \times 10 + 10$ a pour arbre de dérivation :



Arbre de dérivation

Exemple (Arbre de dérivation)

Le mot $10 \times 10 + 10$ a également pour arbre de dérivation :



Grammaire ambiguë ⋆⋆

Définition (Grammaire ambiguë)

Une grammaire G est **ambiguë** s'il existe un mot de $\mathcal{L}(G)$ qui a au moins deux arbres de dérivation, c'est-à-dire deux dérivations la plus à gauche.

Exemple (Grammaire ambiguë)

La grammaire $G = (\{E, N\}, \{+, \times, 0, 1\}, E, R)$ avec R :

- $E \to E + E \mid E \times E \mid N$
- $N \to 0 \mid 1 \mid 0N \mid 1N$

est ambiguë.

Plan

Grammaire générale (type 0) ⋆⋆

Définition (Grammaire générale (type 0))

Une **grammaire de type 0** ou **grammaire générale** est une grammaire sans restriction sur la forme des règles.

Définition (Langage général)

Un **langage général** est un langage engendré par une grammaire générale.

Grammaire contextuelle (type 1) ⋆⋆

Définition (Grammaire contextuelle (type 1))

Une **grammaire de type 1** ou **grammaire contextuelle** est une grammaire où les règles sont de la forme :

$$\alpha A \beta \to \alpha w \beta$$

avec $\alpha, \beta \in V^*$, $w \in V^+$ et $A \in N$. Le symbole A est remplacé par w si on a le contexte α à gauche et β à droite.

Définition (Langage contextuel (type 1))

Un **langage contextuel** est un langage engendré par une grammaire contextuelle.

Grammaire contextuelle (type 1)

Exemple (Grammaire contextuelle (type 1))

La grammaire $G = (\{S, B, C, H\}, \{a, b, c\}, S, R)$ avec R :

- $S \to aSBC$
- $S \to aBC$
- $CB \to HB$
- $HB \to HC$
- $HC \to BC$
- $aB \to ab$
- $bB \to bb$
- $bC \to bc$
- $cC \to cc$

engendre le langage $L = \{a^n b^n c^n, n \geq 1\}$

Grammaire contextuelle (type 1) ⋆⋆

Définition (Grammaire croissante)

Une **grammaire croissante** est une grammaire où les règles sont de la forme :

$$\alpha \to \beta$$

avec $\alpha, \beta \in V^*$ et $|\alpha| \leq |\beta|$

Proposition

Les langages engendrés par les grammaires croissantes sont les langages contextuels.

Grammaire contextuelle (type 1)

Exemple (Grammaire croissante)

La grammaire $G = (\{S, B\}, \{a, b, c\}, S, R)$ avec R :

- $S \to abc \mid aSBc$
- $cB \to Bc$
- $bB \to bb$

engendre le langage $L = \{a^n b^n c^n, n \geq 1\}$

Grammaire algébrique (type 2) ⋆⋆

Définition (Grammaire algébrique (type 2))

Une **grammaire de type 2** ou **grammaire algébrique** ou **grammaire hors contexte** est une grammaire où les règles sont de la forme :

$$A \to \alpha$$

avec $A \in N$ et $\alpha \in V^*$, la partie gauche est réduite à un non-terminal.

Définition (Langage algébrique (type 2))

Un **langage algébrique** est un langage engendré par une grammaire algébrique.

Grammaire algébrique (type 2)

Exemple (Grammaire algébrique (type 2))

La grammaire $G = (\{S\}, \{a, b\}, S, R)$ avec R :

- $S \to \varepsilon \mid aSb$

engendre le langage $L = \{a^n b^n, n \geq 0\}$

Grammaire régulière (type 3) ⋆⋆

Définition (Grammaire régulière (type 3))

Une **grammaire de type 3** ou **grammaire régulière** est une grammaire où les règles sont de la forme :

$$A \to a \text{ ou } A \to aB$$

avec $A, B \in N$ et $a \in T$.

Définition (Langage régulier (type 3))

Un **langage régulier** est un langage engendré par une grammaire régulière.

Grammaire régulière (type 3)

Exemple (Grammaire régulière (type 3))

La grammaire $G = (\{S, A, B\}, \{a, b\}, S, R)$ avec R :

- $S \to aA$
- $A \to bA \mid bB$
- $B \to a$

engendre le langage $L = \{ab^n a, n \geq 1\}$

Hiérarchie de Chomsky

Hiérarchie de Chomsky

Type	Grammaire	Reconnaisseur
0	Générale	Machine de Turing
1	Contextuelle	
2	Algébrique	Automate à pile
3	Régulière	Automate

Troisième partie

Automates d'états finis

Plan de ce cours

Plan

Automate fini déterministe ★★★

Définition (Automate fini déterministe)

Un **automate fini déterministe** $\mathcal{A}$ sur un alphabet Σ est un quadruplet (Q, q_0, δ, Q_F) où :

- Q est un ensemble fini d'états
- $q_0 \in Q$ est un état initial
- $\delta : Q \times \Sigma \to Q$ est la fonction de transition
- $Q_F \subseteq Q$ est un ensemble d'états finaux

Remarque

La fonction de transition δ peut être partielle, c'est-à-dire non-définie sur tout l'ensemble $Q \times \Sigma$.

Automate fini déterministe

Exemple (Automate fini déterministe)

$\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, q_0, \delta, \{q_3\})$ avec

$$\delta(q_0, b) = q_1, \delta(q_1, a) = q_2, \delta(q_2, a) = q_2, \delta(q_2, b) = q_3$$

Fonction de transition

Fonction de transition étendue δ

On étend la fonction $\delta : Q \times \Sigma^* \to Q$ aux mots sur l'alphabet Σ de la manière suivante :

- $\delta(q, \varepsilon) = q$
- $\delta(q, a \cdot \alpha) = \delta(\delta(q, a), \alpha), a \in \Sigma, \alpha \in \Sigma^*$

Notation alternative

La fonction $\delta(q, a)$ est parfois notée $q \,.\, a$. Les deux propriétés précédentes s'écrivent alors :

- $q \,.\, \varepsilon = q$
- $q \,.\, (a \cdot \alpha) = (q \,.\, a) \,.\, \alpha$

Fonction de transition

Exemple (Fonction de transition)

Soit $\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, q_0, \delta, \{q_3\})$ avec

$$\delta(q_0, b) = q_1, \delta(q_1, a) = q_2, \delta(q_2, a) = q_2, \delta(q_2, b) = q_3$$

Alors, par exemple, on a :

- $\delta(q_0, ba) = q_2$
- $\delta(q_0, baaaaaaaa) = q_2$
- $\delta(q_0, baaaaab) = q_3$
- $\delta(q_2, aab) = q_3$
- $\delta(q_0, bb)$ n'est pas défini parce que $\delta(q_1, b)$ n'est pas défini

Dérivation en une étape

Définition (Configuration)

Une **configuration** est une paire (q, w), où $q \in Q, w \in \Sigma^*$

Définition (Dérivation en une étape)

Soit $\mathcal{A}$ un automate et (q, w) et (q', w') deux configurations. La configuration (q', w') est **dérivable en une étape** de la configuration (q, w) par $\mathcal{A}$, noté $(q, w) \mapsto (q', w')$ si :

- $w = a \cdot w'$ avec $a \in \Sigma$
- $\mathcal{A}$ est dans l'état q
- $q' = \delta(q, a)$

Remarque

On dit qu'on «lit» la lettre a.

Dérivation en une étape

Exemple (Dérivation en une étape)

Soit $\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, q_0, \delta, \{q_3\})$ avec

$$\delta(q_0, b) = q_1, \delta(q_1, a) = q_2, \delta(q_2, a) = q_2, \delta(q_2, b) = q_3$$

Alors, on a les dérivations suivantes :

$$(q_0, baaab) \mapsto (q_1, aaab) \mapsto (q_2, aab) \mapsto (q_2, ab) \mapsto (q_2, b) \mapsto (q_3, \varepsilon)$$

Dérivation

Définition (Dérivation)

Soit $\mathcal{A}$ un automate et (q, w) et (q', w') deux configurations. La configuration (q', w') est **dérivable** de la configuration (q, w) par $\mathcal{A}$, noté $(q, w) \mapsto^* (q', w')$ si :

- $\exists k \geq 0$ et k configurations $(q_i, w_i), 1 \leq i \leq k$ avec $(q_0, w_0) = (q, w)$ et $(q_k, w_k) = (q', w')$
- $(q_i, w_i) \mapsto (q_{i+1}, w_{i+1}), 1 \leq i < k$

Dérivation

Exemple (Dérivation)

Soit $\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, q_0, \delta, \{q_3\})$ avec

$$\delta(q_0, b) = q_1, \delta(q_1, a) = q_2, \delta(q_2, a) = q_2, \delta(q_2, b) = q_3$$

Alors, on a les dérivations suivantes :

$$(q_0, baaab) \mapsto^* (q_2, aab) \mapsto^* (q_3, \varepsilon)$$

Mot accepté par un automate ⋆⋆

Définition (Mot accepté)

Un mot w est **accepté** par un automate si :

$$(q_0, w) \mapsto^* (q, \varepsilon), q \in Q_F$$

Définition (Mot accepté)

Un mot w est **accepté** par un automate si :

$$\delta(q_0, w) = q, q \in Q_F$$

Remarque

On dit aussi que w est **reconnu** par un automate.

Mot accepté par un automate

Exemple (Mot accepté par un automate)

Soit $\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, q_0, \delta, \{q_3\})$ avec

$$\delta(q_0, b) = q_1, \delta(q_1, a) = q_2, \delta(q_2, a) = q_2, \delta(q_2, b) = q_3$$

Alors, les mots suivants sont acceptés par l'automate :

- *bab*
- *baaaaab*

Les mots suivants ne sont pas acceptés par l'automate :

- *baaa*
- *bb*

Langage accepté par un automate ⋆⋆

Définition (Langage accepté)

Le **langage accepté** par l'automate $\mathcal{A}$, noté $\mathcal{L}(\mathcal{A})$, est défini par :

$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^*, (q_0, w) \mapsto^* (q, \varepsilon), q \in Q_F\}$$

Définition (Langage reconnaissable)

Un **langage reconnaissable** L est un langage tel qu'il existe un automate $\mathcal{A}$ qui accepte le langage L :

$$L = \mathcal{L}(\mathcal{A})$$

Langage accepté par un automate

Exemple (Langage accepté par un automate)

Soit $\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, q_0, \delta, \{q_3\})$ avec

$$\delta(q_0, b) = q_1, \delta(q_1, a) = q_2, \delta(q_2, a) = q_2, \delta(q_2, b) = q_3$$

Alors, le langage accepté par l'automate est :

$$\mathcal{L}(\mathcal{A}) = \{baa^n b, n \geq 0\} = \{ba^n b, n \geq 1\}$$

Plan

Représentations d'un automate ⋆

Représentations d'un automate

- Table de transition d'état
- Diagramme d'états-transitions

Représentation par une table ⋆⋆

Table de transition

Une table de transition d'état est une table à deux dimensions avec :

- l'ensemble des états verticalement
- l'ensemble des lettres de l'alphabet horizontalement

Chaque case contient l'état suivant correspondant à l'état actuel et la lettre de l'alphabet.

Table de transition

Exemple (Table de transition)

Soit $\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, q_0, \delta, \{q_3\})$ avec

$$\delta(q_0, b) = q_1, \delta(q_1, a) = q_2, \delta(q_2, a) = q_2, \delta(q_2, b) = q_3$$

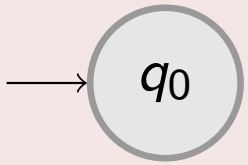
Alors, la table de transition de l'automate est :

	a	b
q_0		q_1
q_1	q_2	
q_2	q_2	q_3
q_3		

Représentation par un diagramme

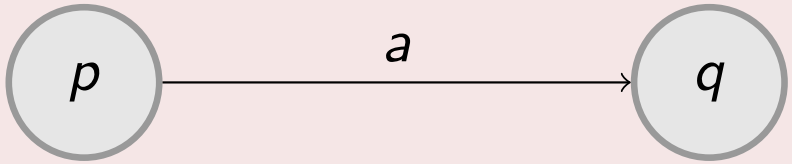
Notations

- État initial



- État final (deux notations)

- Transition entre l'état p et q : $\delta(p, a) = q$



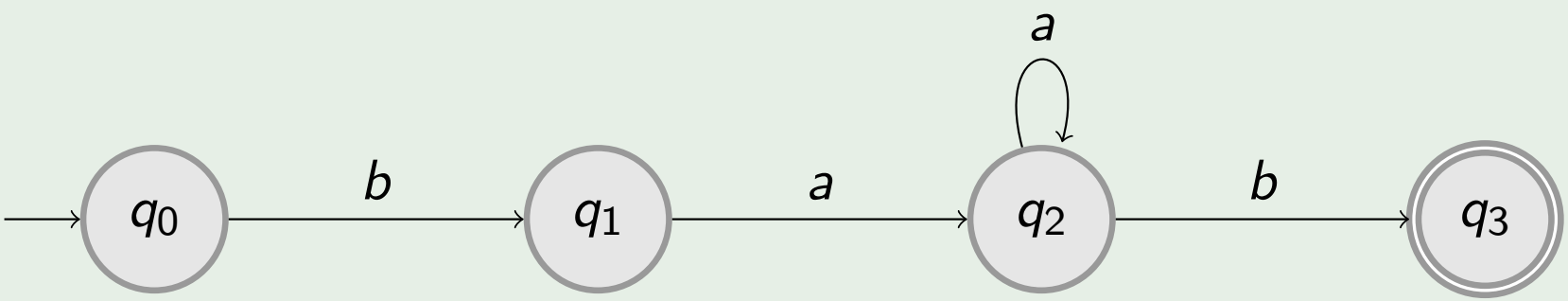
Représentation par un diagramme

Exemple (Diagramme)

Soit $\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, q_0, \delta, \{q_3\})$ avec

$$\delta(q_0, b) = q_1, \delta(q_1, a) = q_2, \delta(q_2, a) = q_2, \delta(q_2, b) = q_3$$

Alors le diagramme d'états-transitions de l'automate est :



Plan

Automates équivalents

Définition (Automates équivalents)

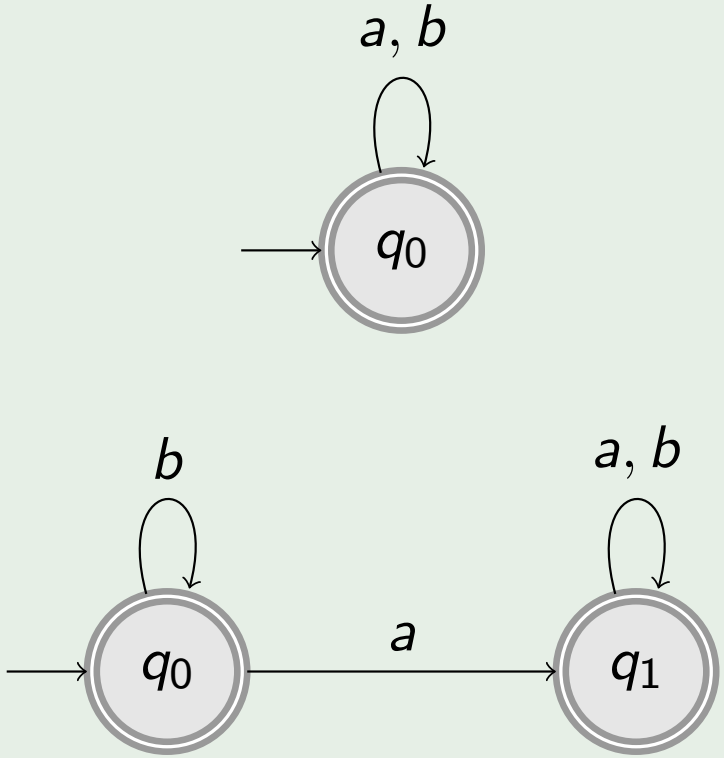
Deux automates $\mathcal{A}_1$ et $\mathcal{A}_2$ sont **équivalents** s'ils reconnaissent le même langage :

$$\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$$

Automates équivalents

Exemple (Automates équivalents)

Les deux automates suivants reconnaissent tous les deux le langage Σ^*, ils sont donc équivalents :



État accessible, co-accessible, utile ⋆⋆

Définition (État accessible)

Un état q est **accessible** s'il existe un chemin entre q_0 et q.

Définition (État co-accessible)

Un état q est **co-accessible** s'il existe un chemin entre q et $q_f \in Q_F$.

Définition (État utile)

Un état est **utile** s'il est à la fois accessible et co-accessible.

Automate complet

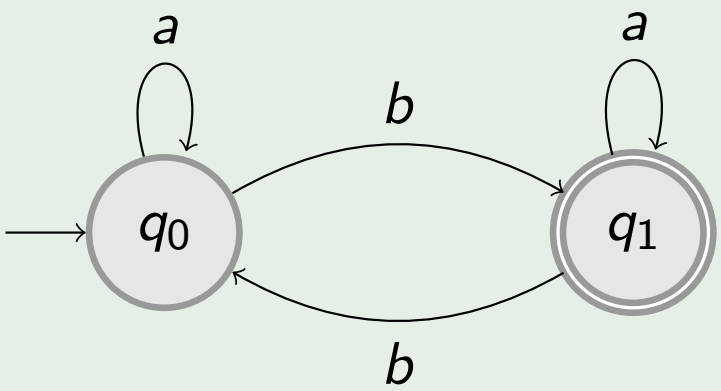
★★★

Définition (Automate complet)

Un automate est **complet** si pour tout état $q \in Q$, il existe une transition pour chaque lettre de l'alphabet Σ.

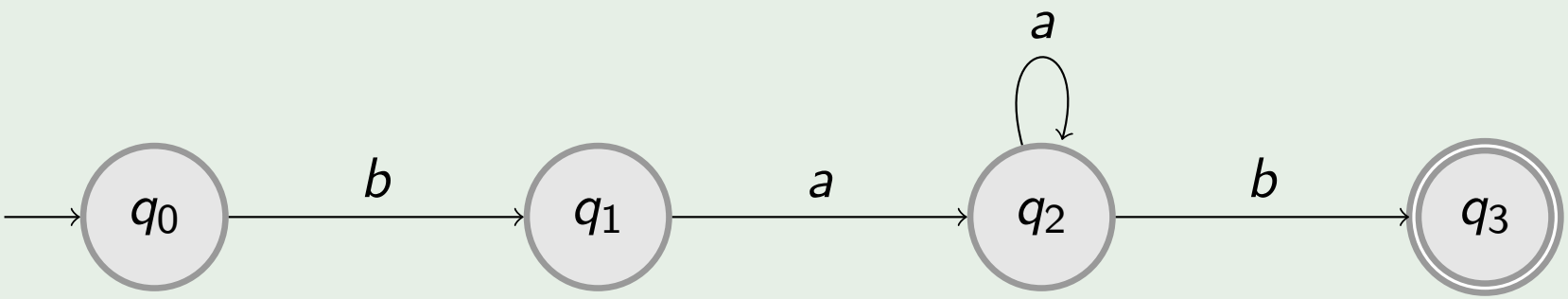
$$\forall q \in Q, \forall a \in \Sigma, \delta(q, a) \text{ est défini}$$

Exemple (Automate complet)



Automate complet

Exemple (Automate non-complet)



État puits et état poubelle ⋆

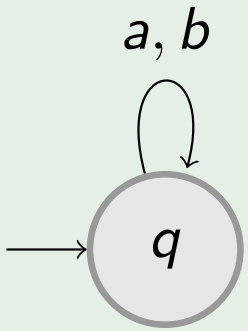
Définition (État puits)

Un **état puits** est un état $q \in Q$ pour lequel toutes les transitions sont de la forme $\delta(q, a) = q, a \in \Sigma$.

Définition (État poubelle)

Un **état poubelle** est un état puits non-final.

Exemple (État poubelle)



Automate complet ⋆⋆

Proposition (Automate complet)

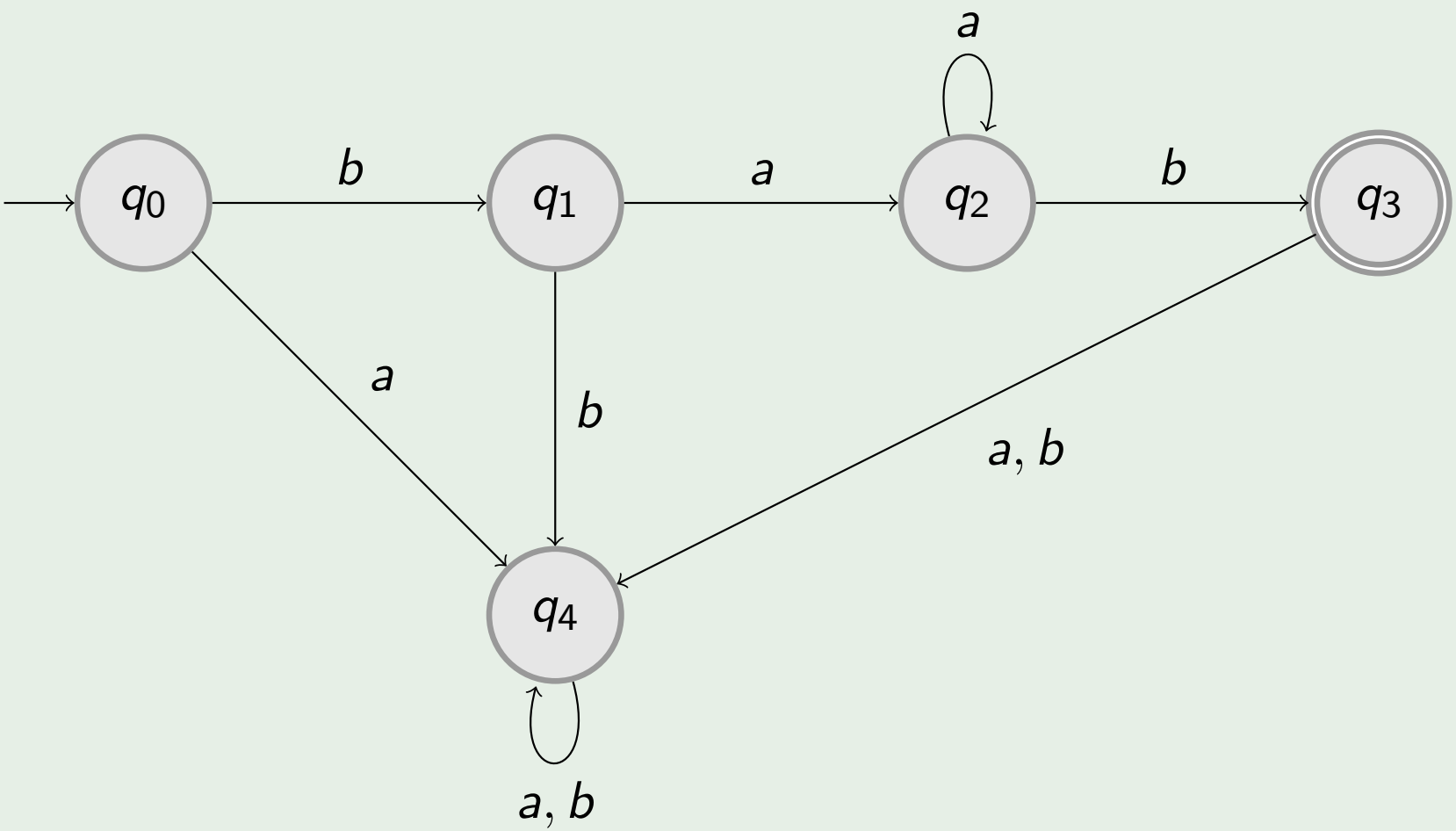
Pour tout automate fini, il existe un automate fini complet équivalent.

Démonstration.

Si l'automate n'est pas complet, on le complète en ajoutant un état poubelle. □

Automate complet

Exemple (Automate complet)



Automate complet

Propriété

On peut toujours dériver un mot w sur un automate complet :

$$(q_0, w) \mapsto (q_1, w_1) \mapsto (q_2, w_2) \mapsto \ldots \mapsto (q_n, \varepsilon)$$

On a deux possibilités :

- *Soit $q_n \in Q_F$, et le mot w est accepté par l'automate*
- *Soit $q_n \notin Q_F$, et le mot w n'est pas accepté par l'automate*

Plan

Automate fini non-déterministe

Définition (Automate fini non-déterministe)

Un **automate fini non-déterministe** $\mathcal{A}$ sur un alphabet Σ est un quadruplet (Q, Q_I, Δ, Q_F) où :

- Q est un ensemble fini d'états
- $Q_I \subseteq Q$ est un ensemble d'état initiaux
- $\Delta \subseteq (Q \times \Sigma \cup \{\varepsilon\} \times Q)$ est une relation de transition
- $Q_F \subseteq Q$ est un ensemble d'états finaux

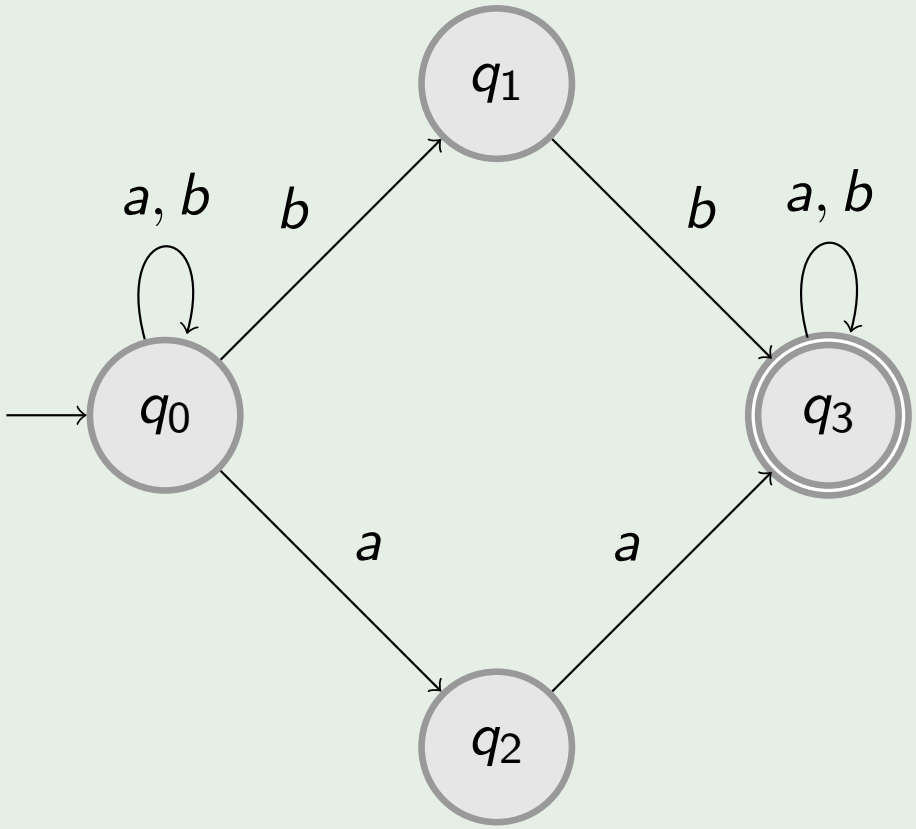
Automate fini non-déterministe

Différences entre automate fini déterministe et non-déterministe

- Il peut y avoir plusieurs états initiaux
- On n'a plus une fonction de transition mais une relation de transition
- Il peut y avoir des ε-transitions

Automate fini non-déterministe

Exemple (Automate fini non-déterministe)



Déterminisation

★★★

Théorème (Déterminisation)

Pour tout automate non-déterministe $\mathcal{A}_N$, il existe un automate fini déterministe $\mathcal{A}_D$ équivalent.

$$\mathcal{L}(\mathcal{A}_N) = \mathcal{L}(\mathcal{A}_D)$$

Preuve constructive

Pour montrer ce théorème, on établit une preuve constructive, c'est-à-dire on donne un algorithme qui permet de produire un automate déterministe équivalent : l'algorithme de déterminisation d'un automate.

Déterminisation

Déterminisation

Soit $\mathcal{A}_N = (Q, Q_I, \Delta, Q_F)$, on définit l'automate $\mathcal{A}_D = \{R, r_0, \delta, R_F\}$ de la manière suivante :

- $R \subseteq 2^Q$,
  R est l'ensemble des sous-ensembles de Q
- $r_0 = \{q, q \in Q_I\}$,
  r_0 est l'ensemble des états initiaux de Q
- $\delta(r_1, a) = r_2 \iff r_2 = \{q_2 \in Q, \exists q_1 \in r_1, (q_1, a, q_2) \in \Delta\}$,
  r_2 est l'ensemble des états d'arrivée d'une transition par a depuis tous les états de r_1
- $R_F = \{r \in R, \exists q \in Q_F, q \in r\}$,
  R_F est l'ensemble des sous-ensembles de Q qui contiennent au moins un état final de Q_F

Déterminisation

★★★

Proposition

Soit $\mathcal{A}_D$ l'automate obtenu après application de l'algorithme précédent sur l'automate $\mathcal{A}_N$, alors :

- *$\mathcal{A}_D$ est déterministe*
- $\mathcal{L}(\mathcal{A}_D) = \mathcal{L}(\mathcal{A}_N)$

Remarque

Si l'automate $\mathcal{A}_N$ contient n états, l'automate $\mathcal{A}_D$ obtenu par l'algorithme de déterminisation peut contenir jusqu'à 2^n états.

Déterminisation en pratique

Déterminisation en pratique

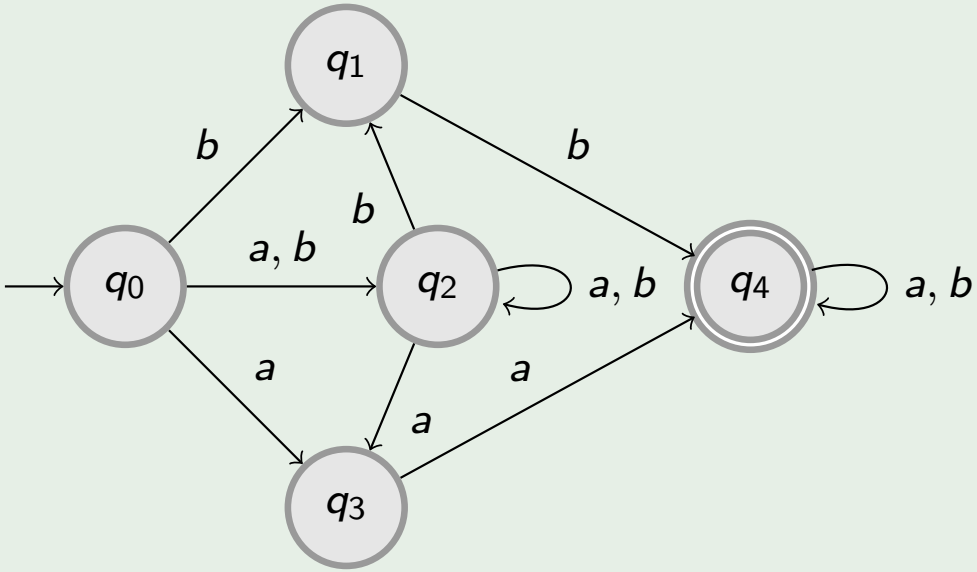
En pratique, on construit la table de transition de $\mathcal{A}_D$ au fur et à mesure :

1. On part de l'état initial de $\mathcal{A}_D$.
2. Pour chaque nouvel ensemble d'états qui apparaît dans la table, on ajoute une ligne dans la table.
3. On recommence jusqu'à ce tous les ensembles aient été traités.

	a	b	$\ldots$
$\{q_{i_1}, \ldots, q_{i_k}\}$	$\{\ldots\}$	$\{\ldots\}$	$\ldots$
$\ldots$	$\ldots$	$\ldots$	$\ldots$

Déterminisation

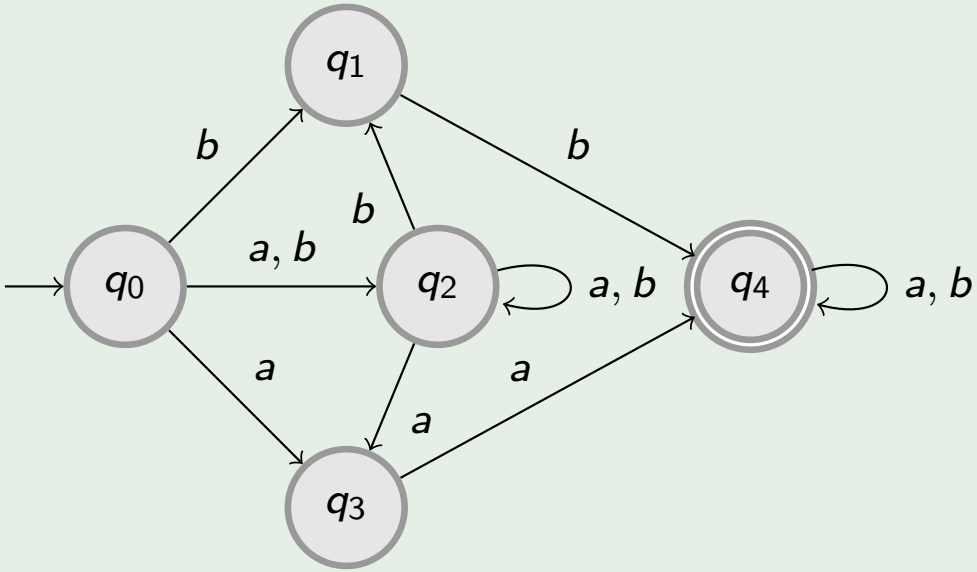
Exemple (Déterminisation)



	a	b
$\{q_0\}$		

Déterminisation

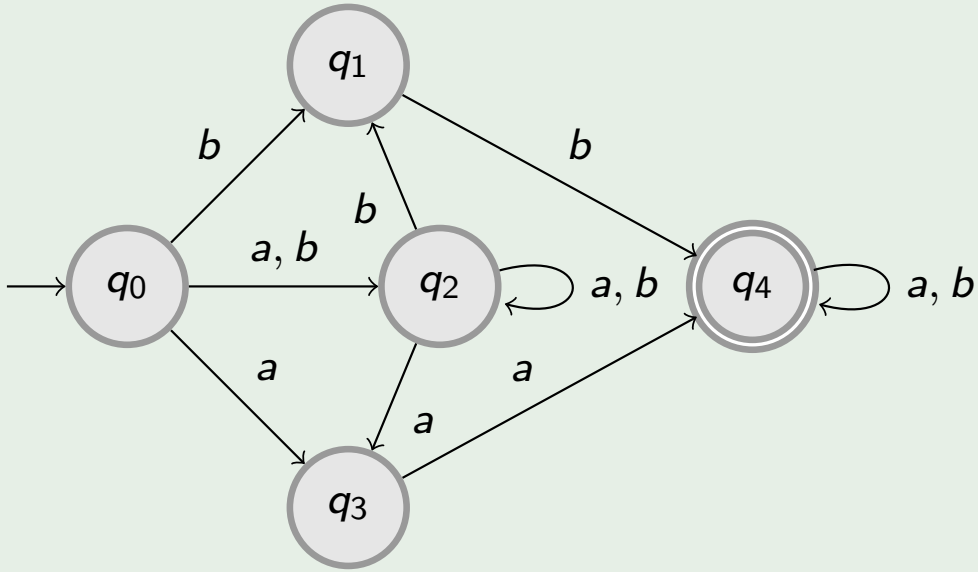
Exemple (Déterminisation)



	a	b
$\{q_0\}$ $\{q_2, q_3\}$ $\{q_1, q_2\}$	$\{q_2, q_3\}$	$\{q_1, q_2\}$

Déterminisation

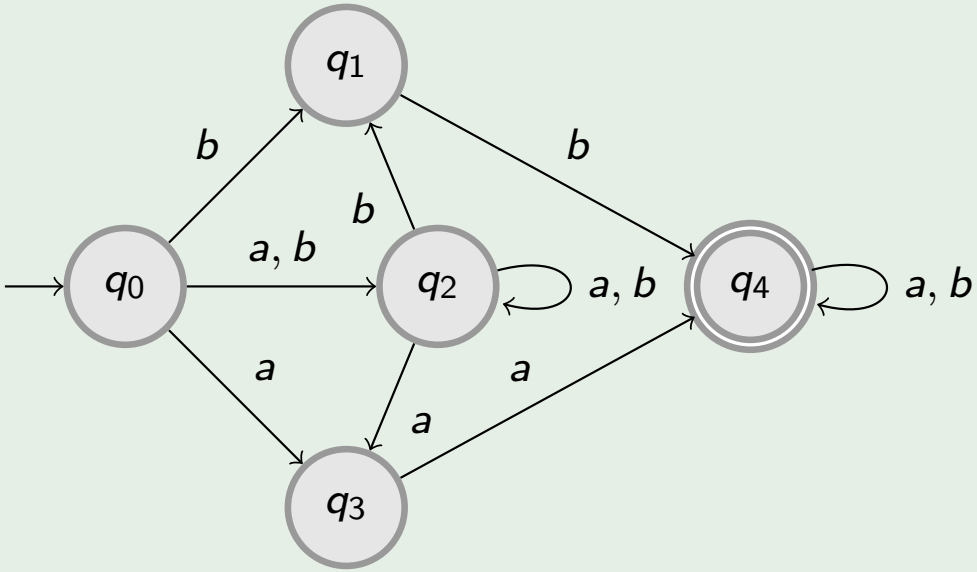
Exemple (Déterminisation)



	a	b
$\{q_0\}$ $\{q_2, q_3\}$ $\{q_1, q_2\}$ $\{q_2, q_3, q_4\}$	$\{q_2, q_3\}$ $\{q_2, q_3, q_4\}$	$\{q_1, q_2\}$ $\{q_1, q_2\}$

Déterminisation

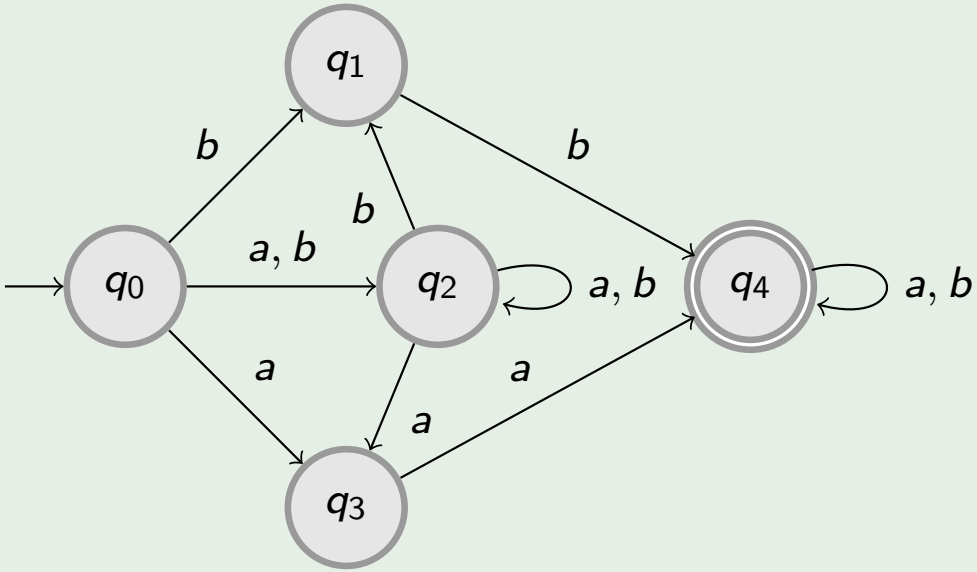
Exemple (Déterminisation)



	a	b
$\{q_0\}$	$\{q_2, q_3\}$	$\{q_1, q_2\}$
$\{q_2, q_3\}$	$\{q_2, q_3, q_4\}$	$\{q_1, q_2\}$
$\{q_1, q_2\}$	$\{q_2, q_3\}$	$\{q_1, q_2, q_4\}$
$\{q_2, q_3, q_4\}$		
$\{q_1, q_2, q_4\}$		

Déterminisation

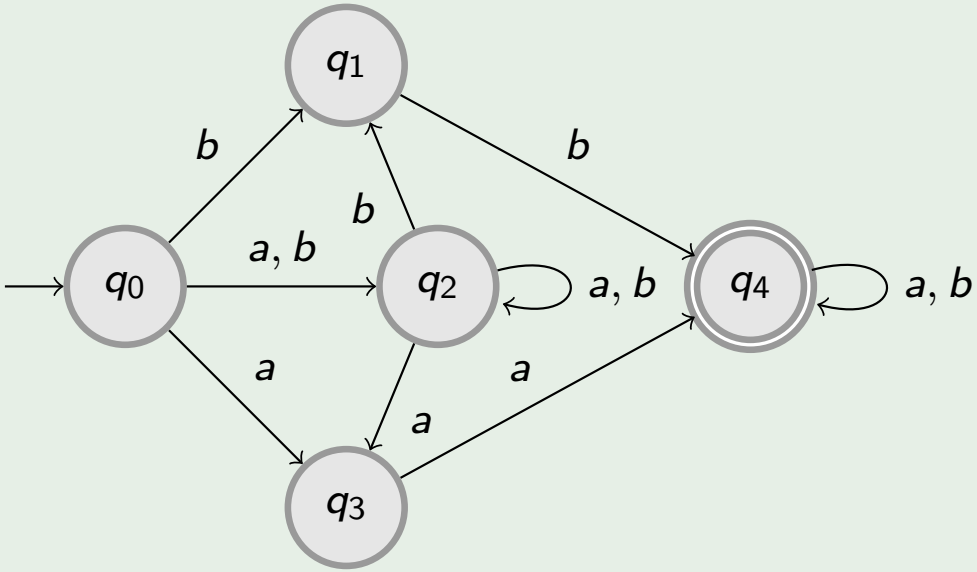
Exemple (Déterminisation)



	a	b
$\{q_0\}$	$\{q_2, q_3\}$	$\{q_1, q_2\}$
$\{q_2, q_3\}$	$\{q_2, q_3, q_4\}$	$\{q_1, q_2\}$
$\{q_1, q_2\}$	$\{q_2, q_3\}$	$\{q_1, q_2, q_4\}$
$\{q_2, q_3, q_4\}$	$\{q_2, q_3, q_4\}$	$\{q_1, q_2, q_4\}$
$\{q_1, q_2, q_4\}$		

Déterminisation

Exemple (Déterminisation)

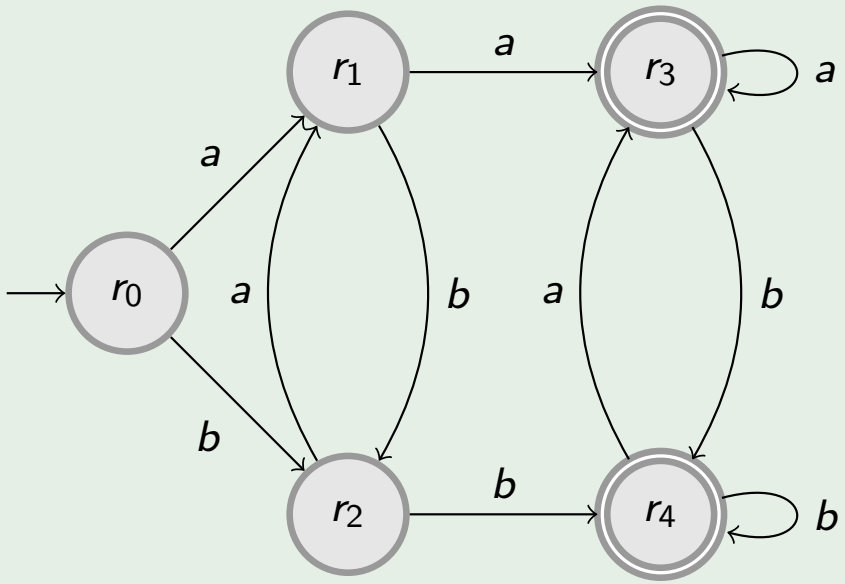


	a	b
$\{q_0\}$	$\{q_2, q_3\}$	$\{q_1, q_2\}$
$\{q_2, q_3\}$	$\{q_2, q_3, q_4\}$	$\{q_1, q_2\}$
$\{q_1, q_2\}$	$\{q_2, q_3\}$	$\{q_1, q_2, q_4\}$
$\{q_2, q_3, q_4\}$	$\{q_2, q_3, q_4\}$	$\{q_1, q_2, q_4\}$
$\{q_1, q_2, q_4\}$	$\{q_1, q_2, q_4\}$	$\{q_2, q_3, q_4\}$

Déterminisation

Exemple (Déterminisation)

		a	b
r_0	$\{q_0\}$	$\{q_2, q_3\}$	$\{q_1, q_2\}$
r_1	$\{q_2, q_3\}$	$\{q_2, q_3, q_4\}$	$\{q_1, q_2\}$
r_2	$\{q_1, q_2\}$	$\{q_2, q_3\}$	$\{q_1, q_2, q_4\}$
r_3	$\{q_2, q_3, q_4\}$	$\{q_2, q_3, q_4\}$	$\{q_1, q_2, q_4\}$
r_4	$\{q_1, q_2, q_4\}$	$\{q_2, q_3, q_4\}$	$\{q_1, q_2, q_4\}$



Quatrième partie

Automates et langages réguliers

Plan de ce cours

5 Transformation d'automates
- Automate émondé et problèmes de décision
- Élimination des transitions instantanées
- Automate standard

6 Automates et langages réguliers
- Équivalence automates–langages réguliers
- Propriétés sur les langages réguliers

Plan

État accessible et co-accessible ⋆

Définition (État accessible)

Soit $\mathcal{A} = (Q, q_0, \Delta, Q_F)$ un automate fini, un état $q \in Q$ est un **état accessible** s'il existe un chemin depuis l'état initial q_0 jusqu'à l'état q. C'est-à-dire s'il existe un mot w tel que $(q_0, w) \mapsto^* (q, \varepsilon)$.

Définition (État co-accessible)

Soit $\mathcal{A} = (Q, q_0, \Delta, Q_F)$ un automate fini, un état $q \in Q$ est un **état co-accessible** (ou état productif) s'il existe un chemin depuis l'état q jusqu'à un état final $q_f \in Q_F$. C'est-à-dire s'il existe un mot w tel que $(q, w) \mapsto^* (q_f, \varepsilon)$.

État utile

Définition (État utile)

Un **état utile** est un état accessible et co-accessible.

Définition (Automate émondé)

Un **automate émondé** est un automate dont tous les états sont utiles.

Remarque

On peut toujours supprimer les états inutiles d'un automate sans changer le langage reconnu.

Décision du vide

★★

Proposition (Décision du vide)

Soit $\mathcal{A} = (Q, q_0, \Delta, Q_F)$ un automate fini, le langage accepté par l'automate $\mathcal{A}$ est non-vide si au moins un état final $q_f \in Q_F$ est accessible.

Remarques

- Décider si le langage accepté par un automate est non-vide revient à trouver un chemin de l'état initial à un état final dans le graphe de l'automate, grâce à un parcours du graphe.
- Le problème du vide d'un langage accepté par un automate est donc décidable en temps linéaire.

Décision de la finitude ⋆⋆

Proposition (Décision de la finitude)

Soit $\mathcal{A} = (Q, q_0, \Delta, Q_F)$ un automate fini, le langage accepté par l'automate $\mathcal{A}$ est infini s'il existe un état q ayant les deux propriétés suivantes :

1. *q est utile*
2. *q est situé sur un cycle non-trivial*

Remarques

- Décider si le langage accepté par un automate est non-vide revient à trouver un cycle dans le graphe de l'automate.
- Le problème de la finitude (ou de l'infinitude) d'un langage accepté par un automate est donc décidable en temps linéaire.

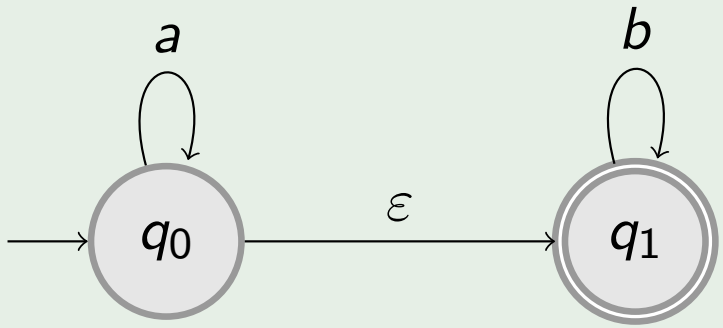
Plan

ε-transition

Définition (ε-transition)

Une ε-**transition** ou **transition instantanée** permet de passer d'un état à un autre d'un automate sans lire de symbole d'entrée.

Exemple (ε-transition)



Remarque

Un automate contenant une ε-transition est non-déterministe.

ε-clôture

Définition (ε-clôture)

Soit $\mathcal{A} = (Q, Q_I, \Delta, Q_F)$ un automate et $q \in Q$, alors, l'ε-**clôture** de q, noté $E(q)$, est l'ensemble des états accessibles depuis q en passant uniquement par des ε-transitions. $E(q)$ est définie par :
$p \in E(q) \iff \exists q_1, q_2, \ldots, q_n \in Q$ avec :

- $q_0 = q$, $q_n = p$
- $\forall 1 \leq i < n, (q_i, \varepsilon, q_{i+1}) \in \Delta$

Élimination des ε-transitions

Algorithme d'élimination des ε-transitions

Soit $\mathcal{A} = (Q, Q_I, \Delta, Q_F)$ un automate, on définit l'automate $\mathcal{A}' = (Q, Q_I, \Delta', Q'_F)$ par :

- $(q, a, p) \in \Delta' \iff (q, a, p) \in \Delta \vee \exists r \in E(q), (r, a, p) \in \Delta$
  Pour tout état r dans l'ε-clôture de q, pour toute transition (r, a, p), alors on ajoute une transition (q, a, p), puis on supprime les ε-transitions partant de q.
- $q \in Q'_F \iff q \in Q_F \vee \exists p \in E(q), p \in Q_F$
  On ajoute à Q_F les états qui permettent d'aller à un état de Q_F via des ε-transitions.

Élimination des ε-transitions ⋆⋆

Proposition

Soit $\mathcal{A}'$ l'automate obtenu après application de l'algorithme précédent sur l'automate $\mathcal{A}$, alors :

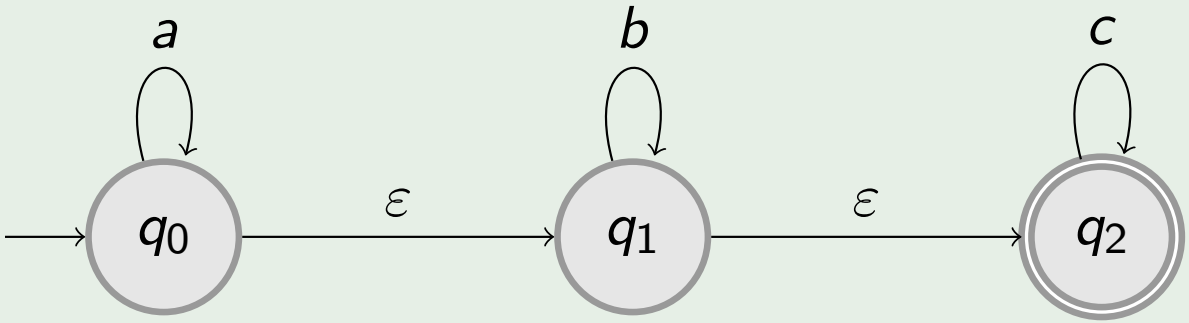
- *$\mathcal{A}'$ ne contient pas d'ε-transition*
- $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$

Remarque

L'automate $\mathcal{A}$ et l'automate $\mathcal{A}'$ ont le même nombre d'états puisque l'algorithme ne fait qu'ajouter et enlever des transitions.

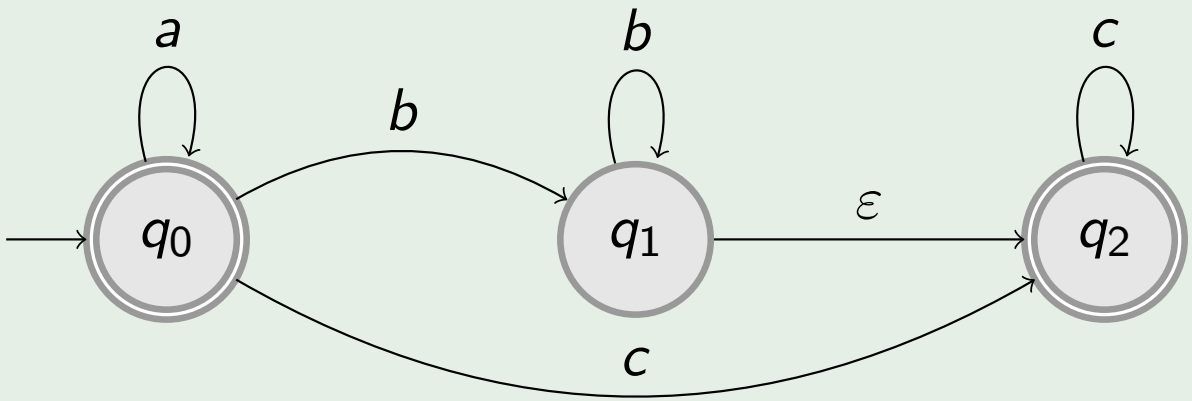
Élimination des ε-transitions

Exemple (Élimination des ε-transitions)



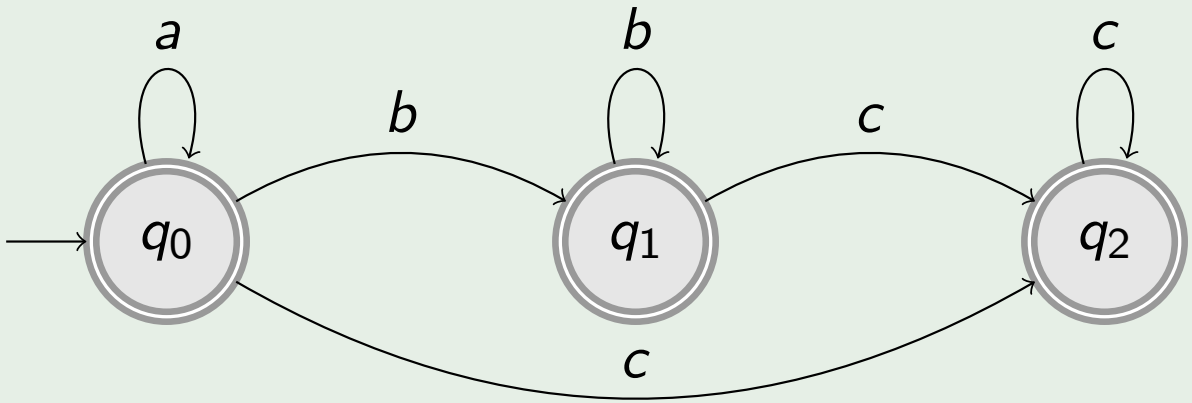
Élimination des ε-transitions

Exemple (Élimination des ε-transitions)



Élimination des ε-transitions

Exemple (Élimination des ε-transitions)



Élimination des ε-transitions

Remarque

Pour déterminiser un automate non-déterministe avec des ε-transitions, on procédera en deux étapes :

1. élimination des ε-transitions
2. déterminisation

Plan

Automate standard

Définition (Automate standard)

Un **automate standard** est un automate fini non-déterministe tel que :

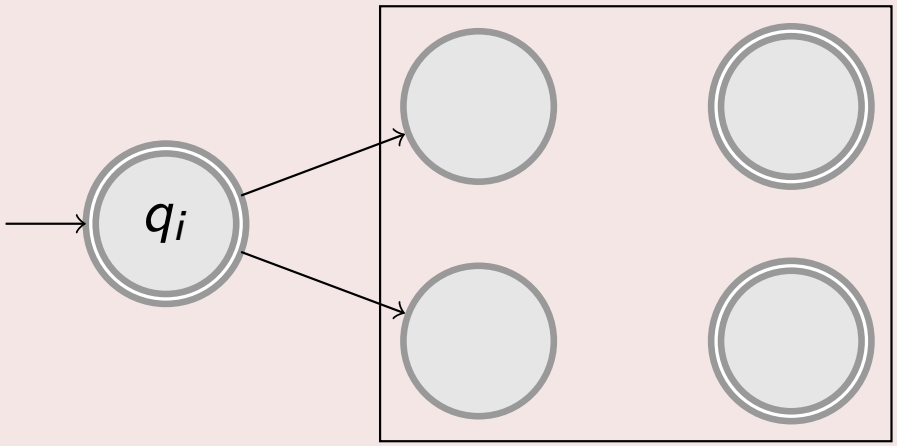
- il n'a qu'un seul état initial q_i
- cet état initial n'est la destination d'aucune transition de l'automate

Remarque

L'état initial q_i peut également être final.

Automate standard

Représentation d'un automate standard



Automate standard

Proposition (Automate standard)

Tout automate fini est équivalent à un automate standard.

Démonstration.

Soit $\mathcal{A}$ un automate. Pour obtenir un automate standard, on ajoute à $\mathcal{A}$ un nouvel état initial q_i qu'on relie via des ε-transitions aux états initiaux de $\mathcal{A}$. L'automate ainsi obtenu est équivalent à $\mathcal{A}$ et est standard. □

Plan

Langages reconnaissables et langages réguliers ⋆⋆⋆

Rappels

- Un langage régulier est engendré par une grammaire régulière
- Un langage reconnaissable est accepté par un automate d'états fini

Théorème

Les langages réguliers sont les langages reconnaissables. Autrement dit :

- *pour toute grammaire régulière, il existe un automate d'états fini qui reconnaît le langage engendré par la grammaire*
- *pour tout automate d'états fini, il existe une grammaire régulière qui engendre le langage accepté par l'automate*

Transformation d'un automate en grammaire ⋆⋆

Transformation d'un automate fini en grammaire régulière

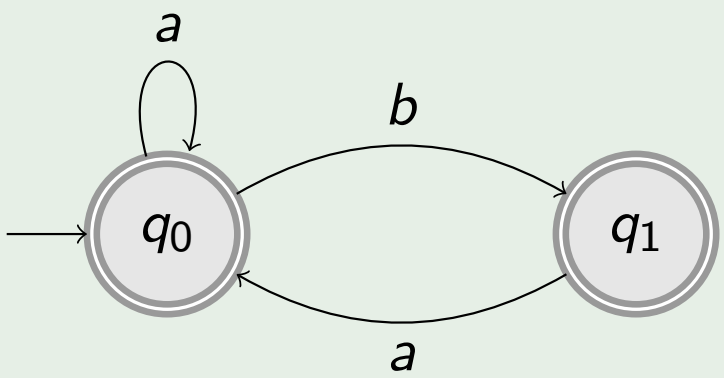
Soit $\mathcal{A} = (Q, q_0, \Delta, Q_F)$ un automate fini, on définit la grammaire $G = (N, T, S, R)$ avec :

- $N = Q$
  L'ensemble des non-terminaux est l'ensemble des états
- $T = \Sigma$
  L'ensemble des terminaux est l'alphabet de l'automate
- $X = q_0$
  L'axiome est l'état initial q_0
- $R = \{X \to aY, (X, a, Y) \in \Delta\} \cup \{X \to \varepsilon, X \in Q_F\}$
  Pour chaque transition, on crée une règle de production, ainsi que pour chaque état final

Alors, $\mathcal{L}(G) = \mathcal{L}(\mathcal{A})$

Transformation d'un automate en grammaire

Exemple (Transformation d'un automate fini en grammaire régulière)



$G = (\{S, U\}, \{a, b\}, S, R)$ avec R :

- $S \to aS \mid bU \mid \varepsilon$
- $U \to aS \mid \varepsilon$

Transformation d'un automate en grammaire ⋆

Remarque

L'automate considéré n'a qu'un seul état initial. Or, un automate non-déterministe peut avoir plusieurs états initiaux. Est-ce un problème ?

Transformation d'une grammaire en automate ⋆⋆

Transformation d'une grammaire régulière en automate fini

Soit $G = (N, T, S, R)$ une grammaire régulière, on définit l'automate $\mathcal{A} = (Q, q_0, \Delta, Q_F)$ avec :

- $Q = N$
  L'ensemble des états est l'ensemble des symboles non-terminaux
- $q_0 = S$
  L'état initial est l'axiome de la grammaire
- $\Delta = \{(X, a, Y), X \to aY \in R\}$
  Pour chaque règle de production de la forme $X \to aY$, on crée une transition étiquetée par a
- $Q_F = \{X, X \to \varepsilon \in R\}$
  Pour chaque règle de production de la forme $X \to \varepsilon$, on crée un état final

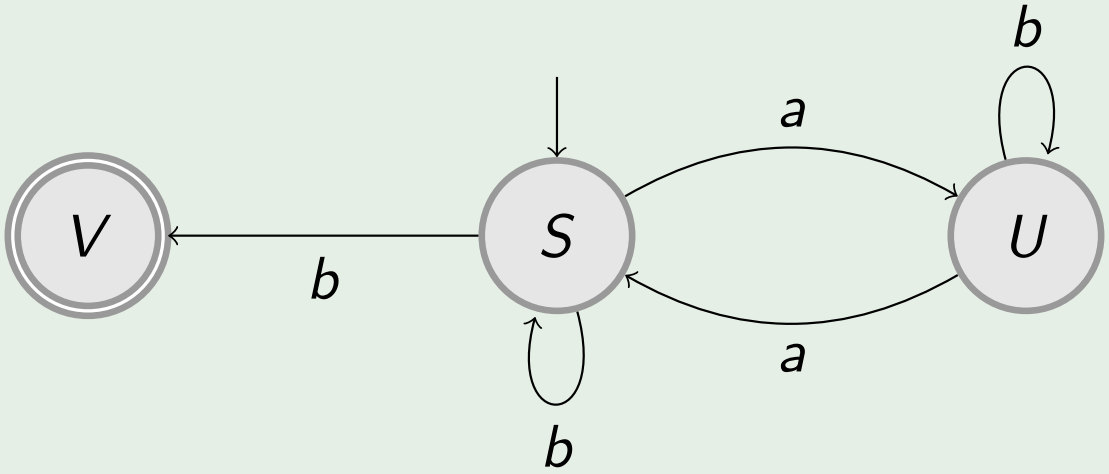
Alors, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(G)$

Transformation d'une grammaire en automate

Exemple (Transformation d'une grammaire régulière en automate fini)

$G = (\{S, U, V\}, \{a, b\}, S, R)$ avec R :

- $S \to bS \mid aU \mid bV$
- $U \to aS \mid bU$
- $V \to \varepsilon$



Transformation d'une grammaire en automate ⋆

Remarque

Une grammaire régulière peut avoir des règles de la forme $A \to a$. Dans ce cas, pour appliquer l'algorithme précédent, on transforme cette règle en deux règles :

- $A \to aA'$
- $A' \to \varepsilon$

Plan

Remarques préliminaires ⋆

Remarques préliminaires

- On a maintenant une équivalence entre langage régulier et automate fini. On peut donc montrer des propriétés sur les langages réguliers en utilisant les automates finis. Plus précisément, on peut contruire un automate qui reconnaît le langage considéré pour montrer que ce langage est régulier.
- De plus, on sait qu'un automate standard non-déterministe avec ε-transitions peut être transformé en un automate déterministe complet équivalent. On peut donc utiliser n'importe quel type d'automate pour montrer une propriété puisque tous les automates reconnaissent les mêmes langages.

Complémentaire

Proposition (Complémentaire d'un langage régulier)

Soit L un langage régulier. Alors, le complémentaire $\overline{L}$ de L est régulier.

Complémentaire ⋆⋆

Démonstration.

Soit L un langage régulier. Soit $\mathcal{A} = (Q, q_0, \delta, Q_F)$ un automate déterministe complet reconnaissant L. On définit $\mathcal{A}' = (Q, q_0, \delta, \overline{Q_F})$. Alors l'automate $\mathcal{A}'$ reconnaît $\overline{L}$. □

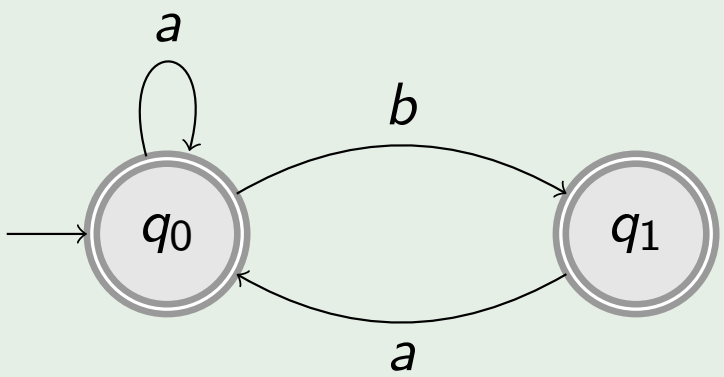
Remarques

- Il est très important que l'automate soit déterministe et complet !
- $\mathcal{A}'$ est également déterministe et complet

Complémentaire

Exemple (Complémentaire)

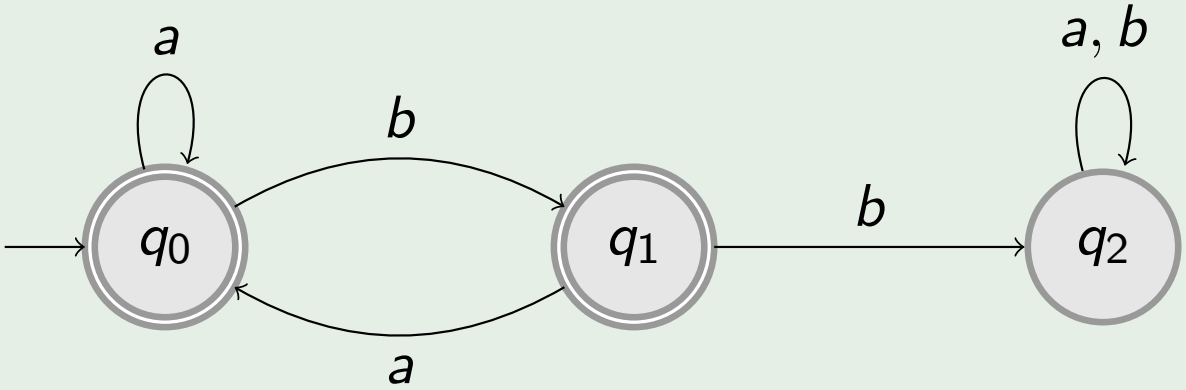
Soit L le langage des mots qui ne contiennent pas deux b consécutifs.
Un automate déterministe qui reconnaît L est :



Complémentaire

Exemple (Complémentaire)

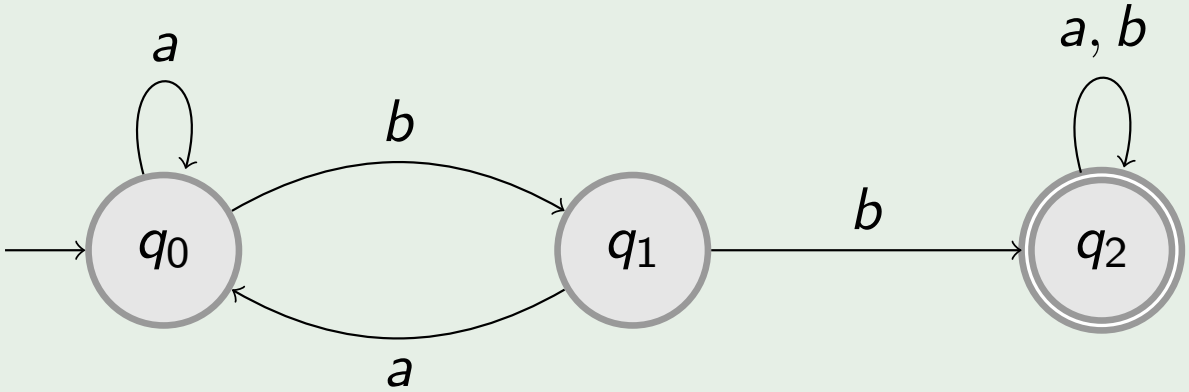
Soit L le langage des mots qui ne contiennent pas deux b consécutifs.
Un automate déterministe complet qui reconnaît L est :



Complémentaire

Exemple (Complémentaire)

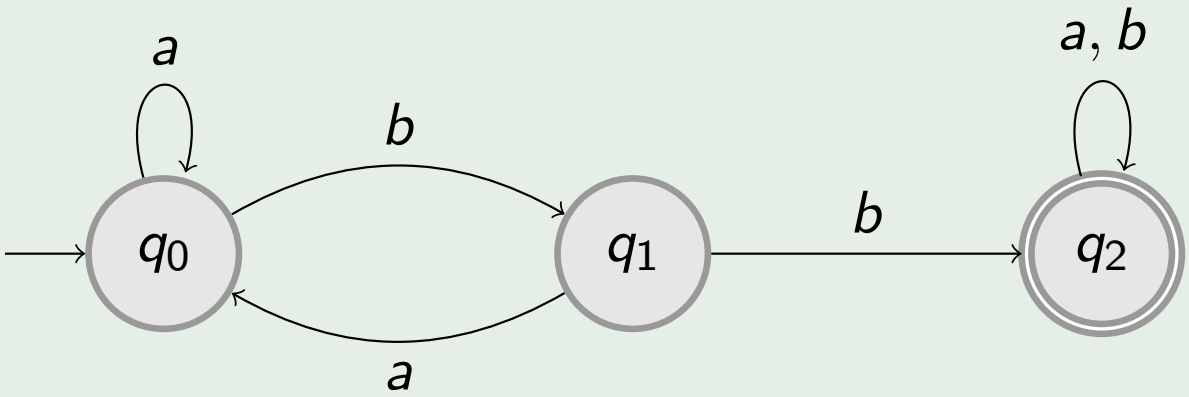
Soit L le langage des mots qui ne contiennent pas deux b consécutifs.
Un automate déterministe complet qui reconnaît $\overline{L}$ est :



Complémentaire

Exemple (Complémentaire)

Soit L le langage des mots qui ne contiennent pas deux b consécutifs.
Un automate déterministe complet qui reconnaît $\overline{L}$ est :



Le langage $\overline{L}$ est le langage des mots qui contiennent deux b consécutifs.

Clôture

Définition (Clôture)

On dit qu'un ensemble E est **clos** ou **fermé** ou **stable** pour une opération $\circ$ si, étant donné deux élément quelconque e_1 et e_2 appartenant à E, alors $e_1 \circ e_2$ appartient aussi à E.
On parle alors de propriété de **clôture** ou de **fermeture**.

Exemples (Clôture)

- $\mathbb{R}$ est clos pour l'opération $+$
- $\mathbb{N}$ n'est pas clos pour l'opération $-$

Union

Proposition (Union de deux langages réguliers)

Soit L_1 et L_2 deux langages réguliers. Alors l'union $L_1 \cup L_2$ est un langage régulier. Autrement dit, les langages réguliers sont clos par union.

Union

Démonstration.

Soit L_1 et L_2 deux langages réguliers. Soit $\mathcal{A}_1 = (Q, q_0, \Delta_Q, Q_F)$ un automate standard reconnaissant L_1 et $\mathcal{A}_2 = (R, r_0, \Delta_R, R_F)$ un automate standard reconnaissant L_2. On définit $\mathcal{A} = (S, s_0, \Delta_S, S_F\})$ avec :

- $S = Q \cup R \cup \{s_0\}$
- $\Delta_S = \Delta_Q \cup \Delta_R \cup \{(s_0, \varepsilon, q_0), (s_0, \varepsilon, r_0)\}$
- $S_F = Q_F \cup R_F$

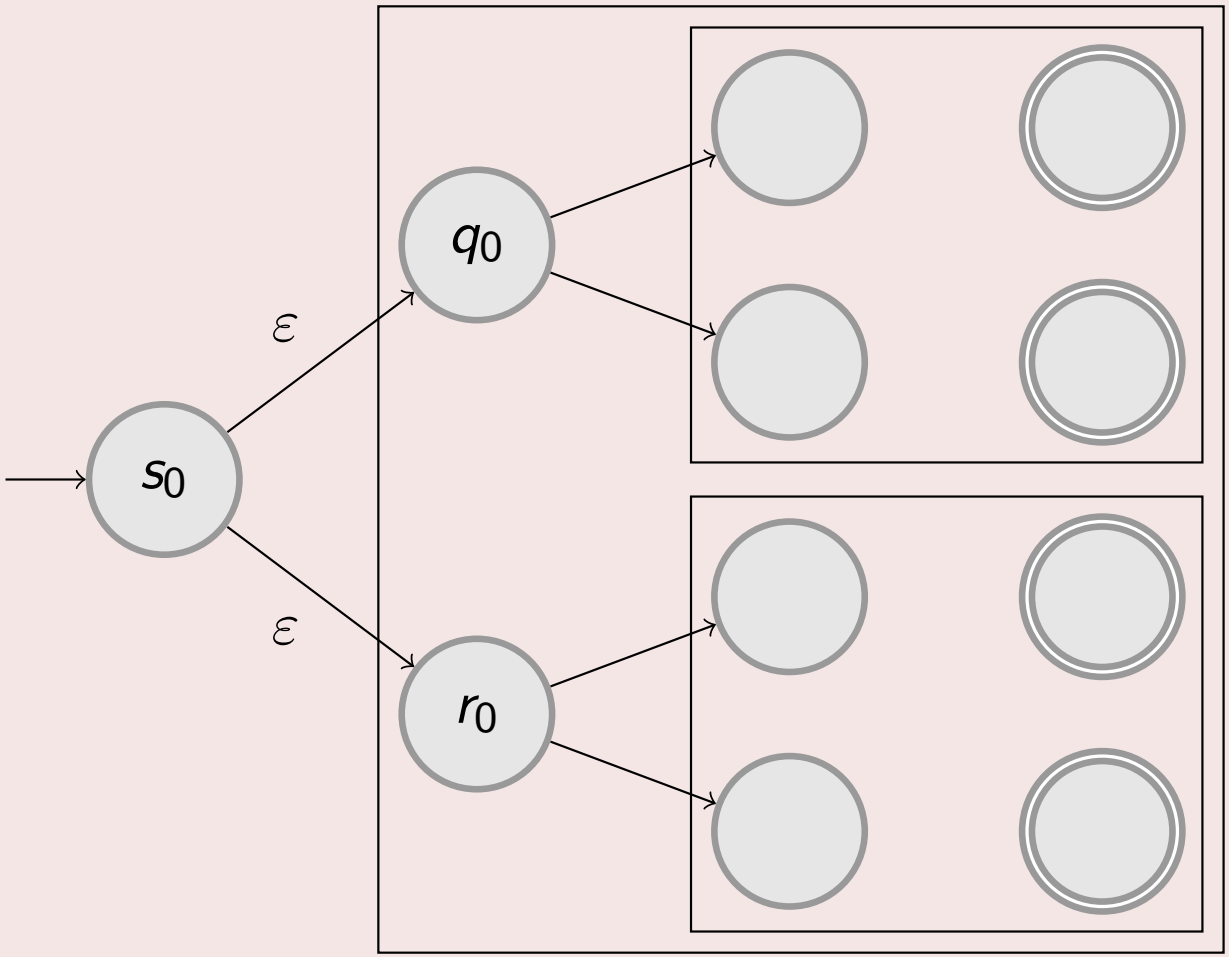
Alors l'automate $\mathcal{A}$ est standard et reconnaît $L_1 \cup L_2$. □

Remarque

$\mathcal{A}$ n'est pas déterministe.

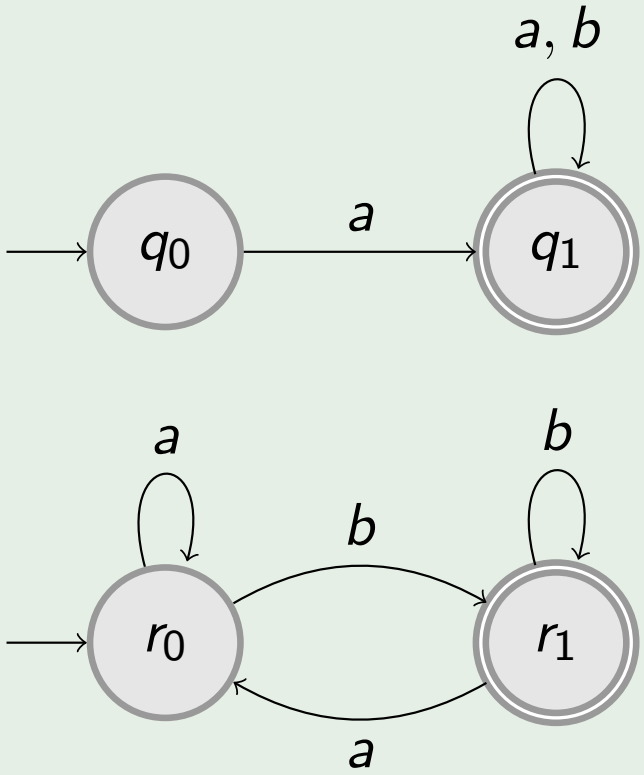
Union

Démonstration visuelle



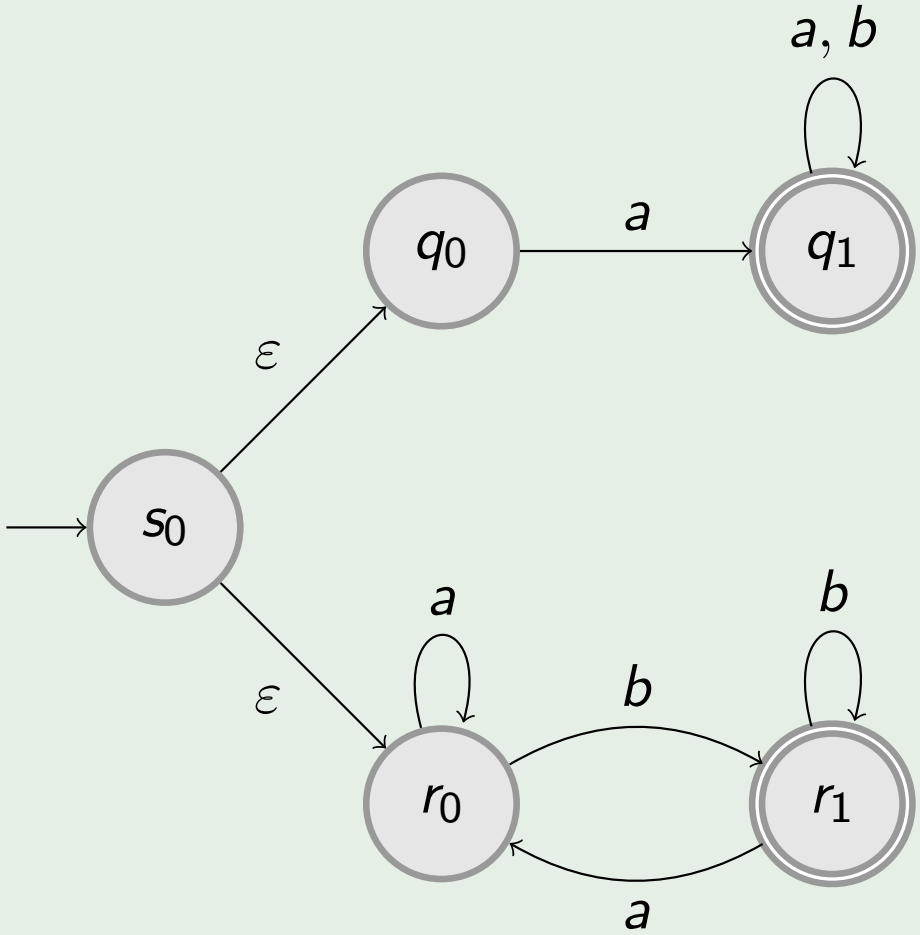
Union

Exemple (Union)



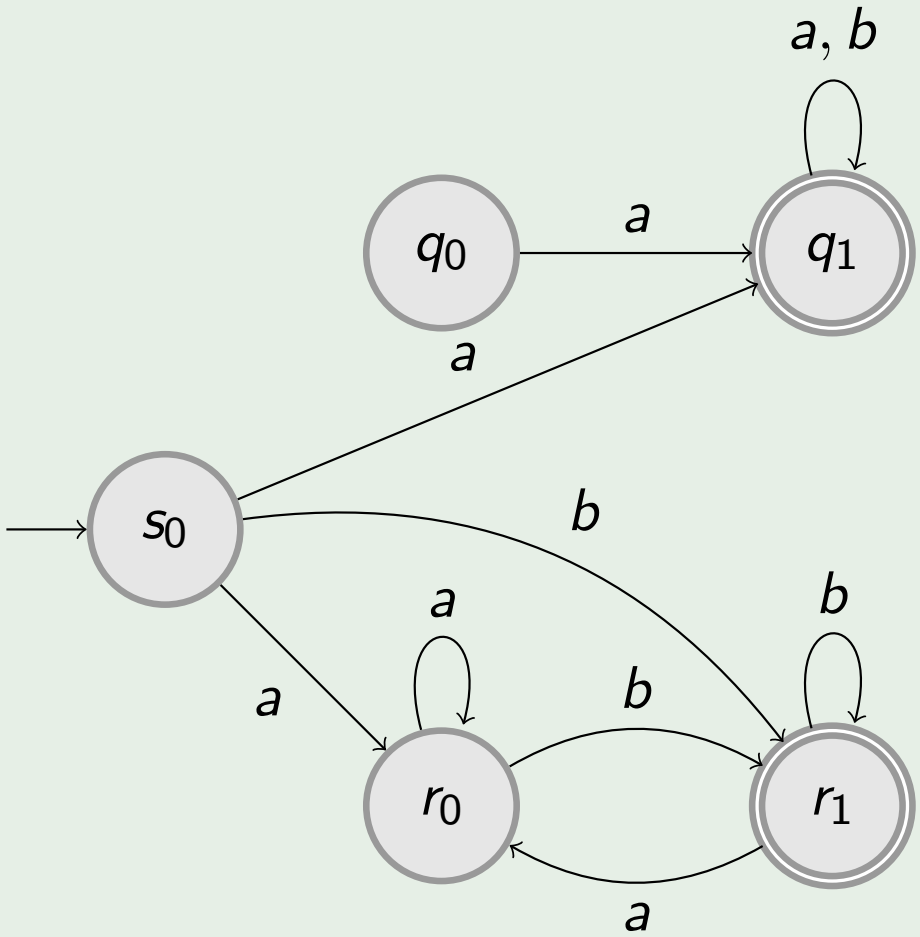
Union

Exemple (Union)



Union

Exemple (Union)



Intersection

Proposition (Intersection de deux langages réguliers)

Soit L_1 et L_2 deux langages réguliers. Alors l'intersection $L_1 \cap L_2$ est un langage régulier. Autrement dit, les langages réguliers sont clos par intersection.

Remarque

On peut le voir comme une conséquence directe des résultats précédents car $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

Intersection ⋆⋆

Démonstration.

Soit L_1 et L_2 deux langages réguliers. Soit $\mathcal{A}_1 = (Q, q_0, \Delta_Q, Q_F)$ un automate reconnaissant L_1 et $\mathcal{A}_2 = (R, r_0, \Delta_R, R_F)$ un automate reconnaissant L_2. On définit $\mathcal{A} = (S, s_0, \Delta_S, S_F)$ avec :

- $S = Q \times R$
- $s_0 = (q_0, r_0)$
- $\Delta_S = \{((q, r), a, (q', r')), (q, a, q') \in \Delta_Q, (r, a, r') \in \Delta_R\}$
- $S_F = Q_F \times R_F$

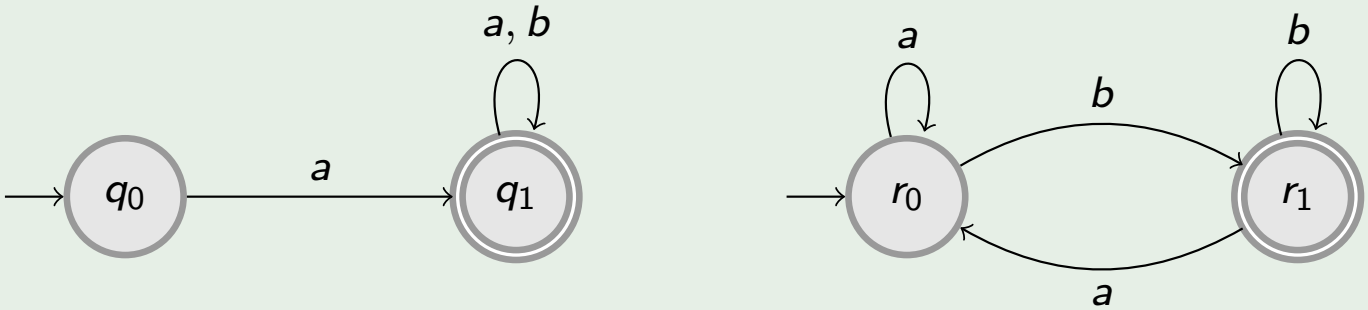
Alors l'automate $\mathcal{A}$ reconnaît $L_1 \cap L_2$. □

Remarque

L'automate $\mathcal{A}$ est appelé **produit** ou **produit synchronisé** des automates $\mathcal{A}_1$ et $\mathcal{A}_2$. Si $\mathcal{A}_1$ et $\mathcal{A}_2$ sont déterministes, alors $\mathcal{A}$ l'est également.

Intersection

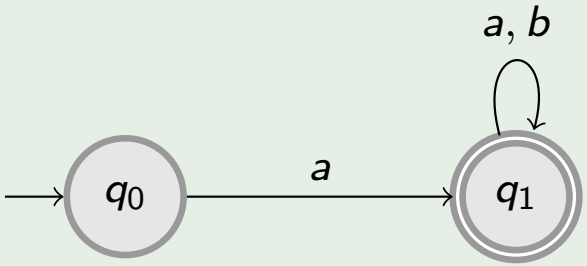
Exemple (Intersection)

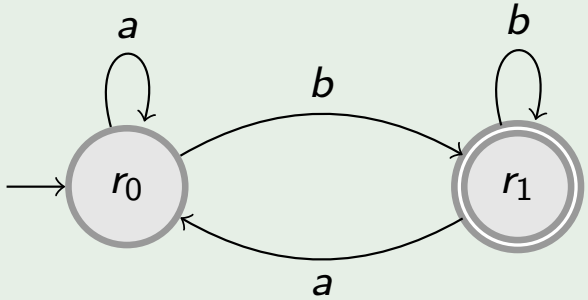


		a	b
s_0	(q_0, r_0)	(q_1, r_0)	
s_1	(q_1, r_0)	(q_1, r_0)	(q_1, r_1)
s_2	(q_1, r_1)	(q_1, r_0)	(q_1, r_1)

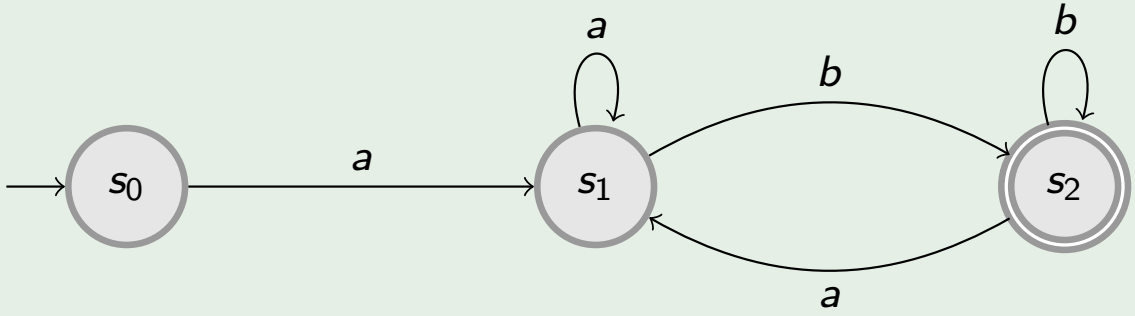
Intersection

Exemple (Intersection)





		a	b
s_0	(q_0, r_0)	(q_1, r_0)	
s_1	(q_1, r_0)	(q_1, r_0)	(q_1, r_1)
s_2	(q_1, r_1)	(q_1, r_0)	(q_1, r_1)



Produit

Proposition (Produit de deux langages réguliers)

Soit L_1 et L_2 deux langages réguliers. Alors le produit $L_1.L_2$ est un langage régulier. Autrement dit, les langages réguliers sont clos par produit.

Produit

Démonstration.

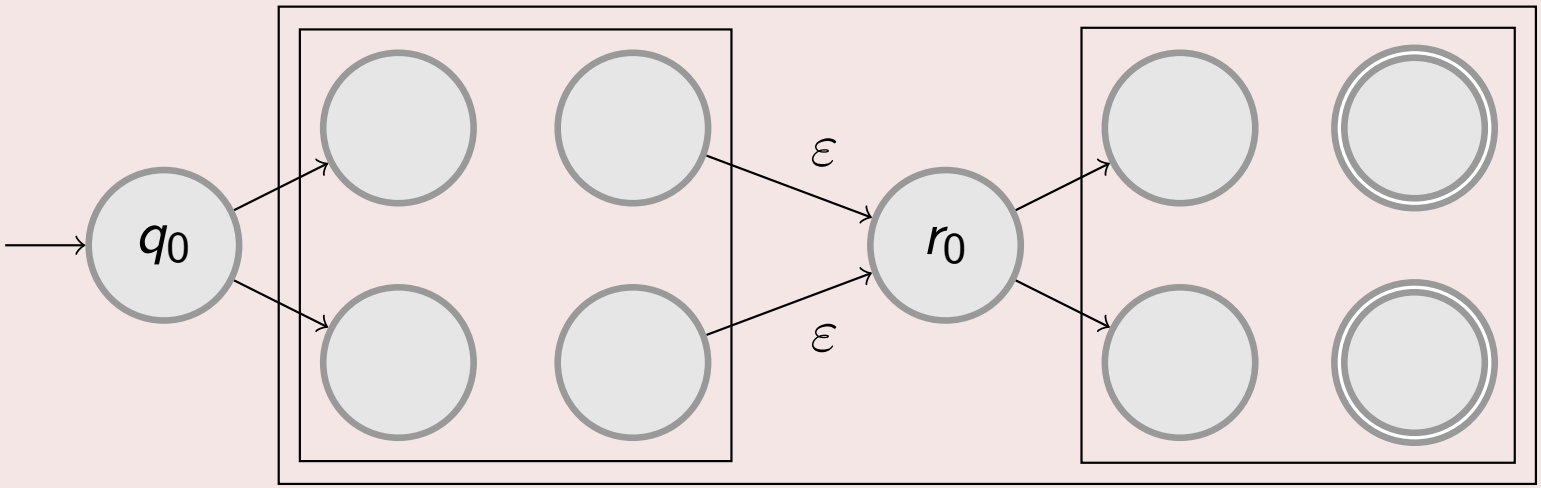
Soit L_1 et L_2 deux langages réguliers. Soit $\mathcal{A}_1 = (Q, q_0, \Delta_Q, Q_F)$ un automate standard reconnaissant L_1 et $\mathcal{A}_2 = (R, r_0, \Delta_R, R_F)$ un automate standard reconnaissant L_2. On définit $\mathcal{A} = (S, s_0, \Delta_S, S_F)$ avec :

- $S = Q \cup R$
- $s_0 = q_0$
- $\Delta_S = \Delta_Q \cup \Delta_R \cup \{(q, \varepsilon, r_0), q \in Q_F\}$
- $S_F = R_F$

Alors l'automate $\mathcal{A}$ est standard et reconnaît $L_1.L_2$. □

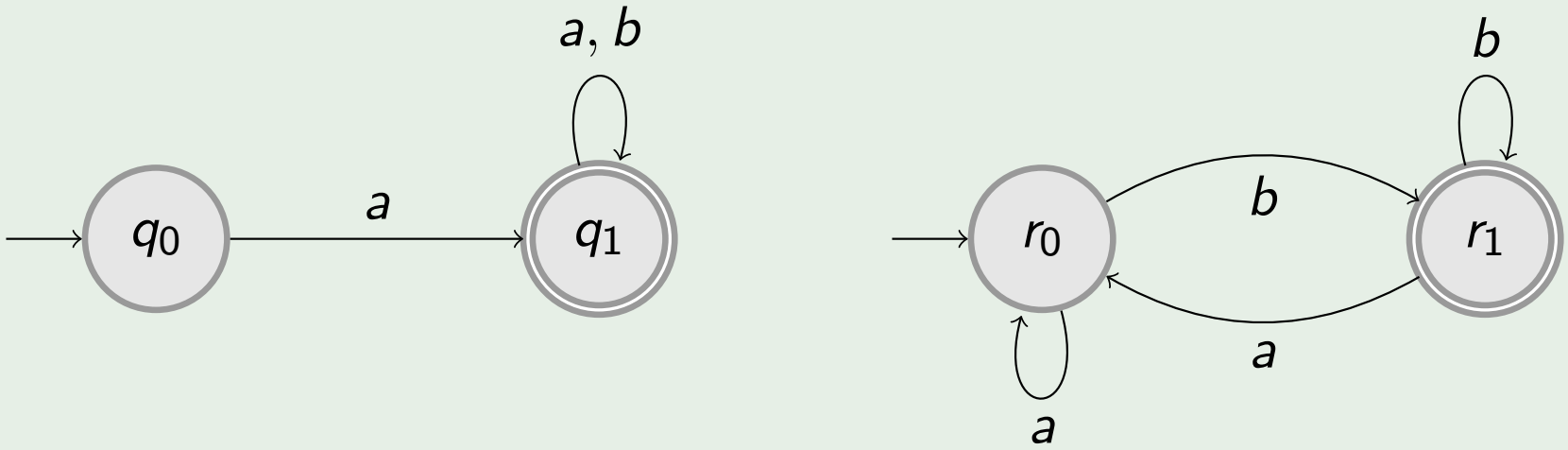
Produit

Démonstration visuelle



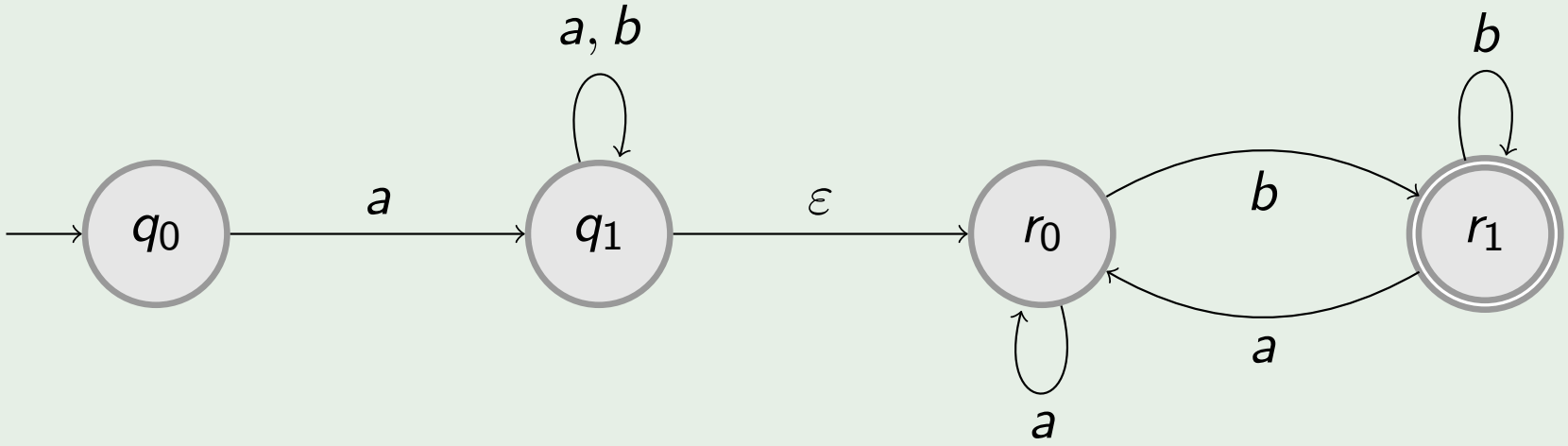
Produit

Exemple (Produit)



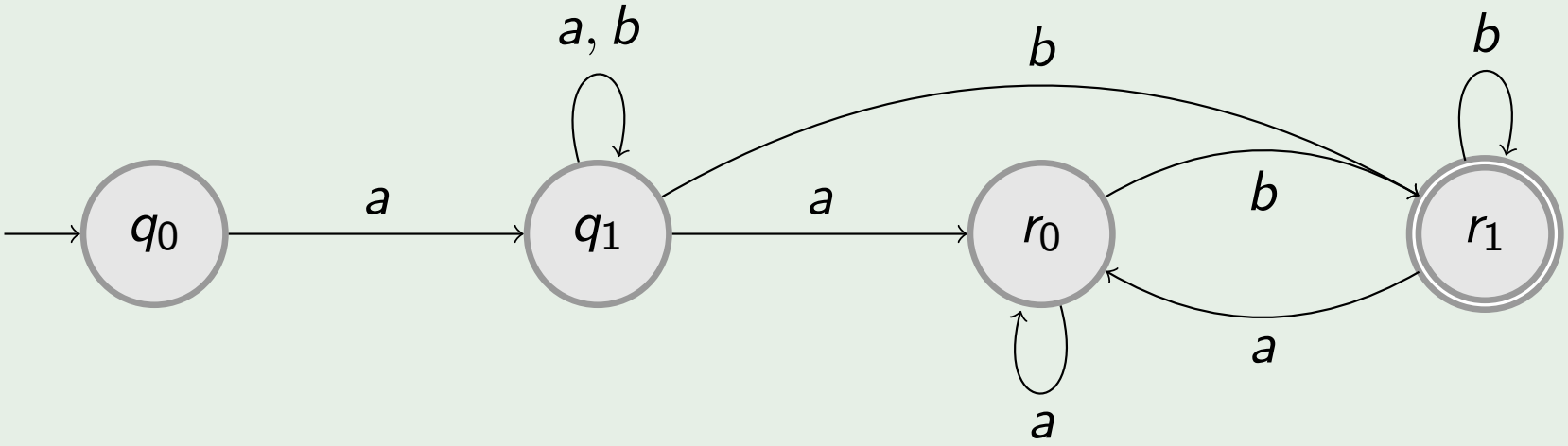
Produit

Exemple (Produit)



Produit

Exemple (Produit)



Itération ⋆⋆

Proposition (Itération d'un langage régulier)

Soit L un langage régulier. Alors l'itéré L^* du langage L est un langage régulier.

Itération ⋆⋆

Démonstration.

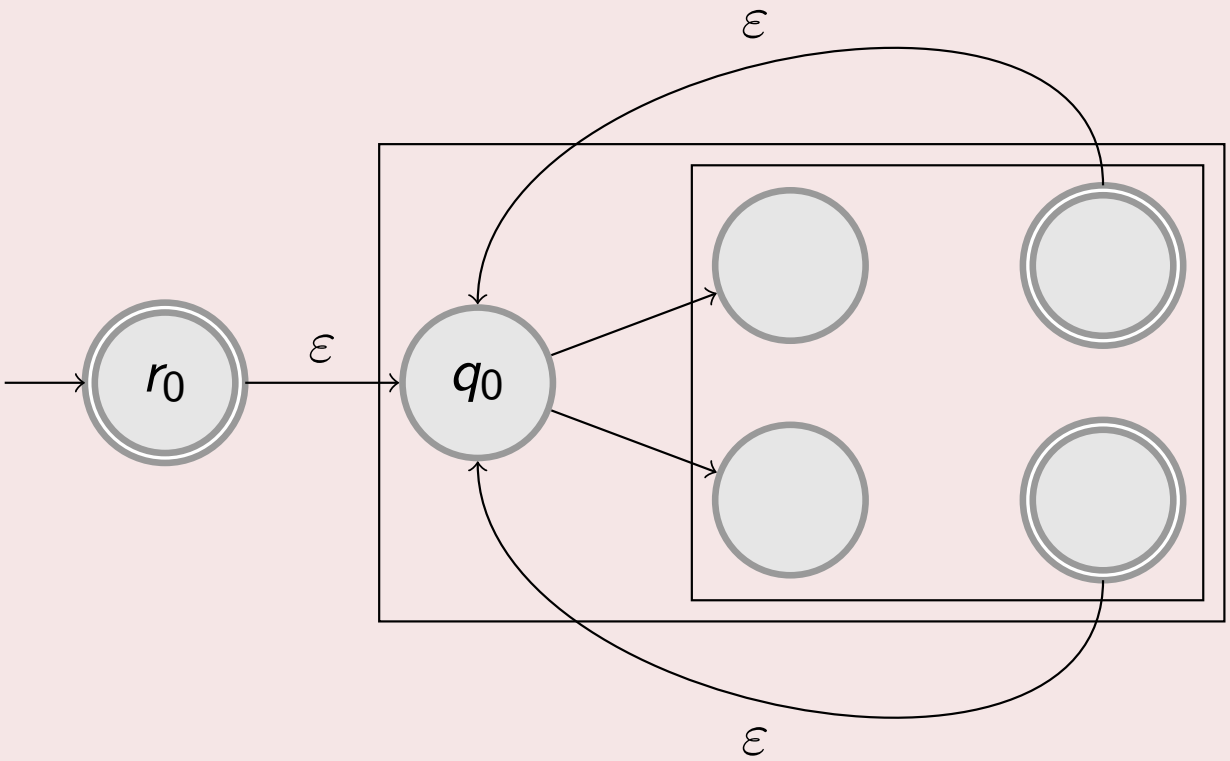
Soit L un langage régulier. Soit $\mathcal{A} = (Q, q_0, \Delta_Q, Q_F)$ un automate standard reconnaissant L. On définit $\mathcal{A}' = (R, r_0, \Delta_R, R_F)$ avec :

- $R = Q \cup \{r_0\}$
- $\Delta_R = \Delta_Q \cup \{r_0, \varepsilon, q_0\} \cup \{(q, \varepsilon, q_0), q \in Q_F\}$
- $R_F = Q_F \cup \{r_0\}$

Alors l'automate $\mathcal{A}'$ est standard reconnaît L^*. □

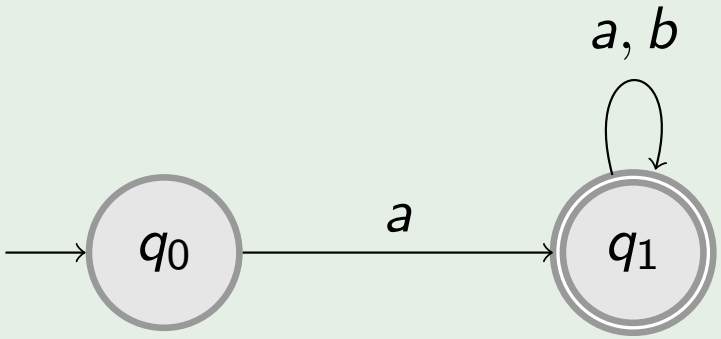
Itération

Démonstration visuelle



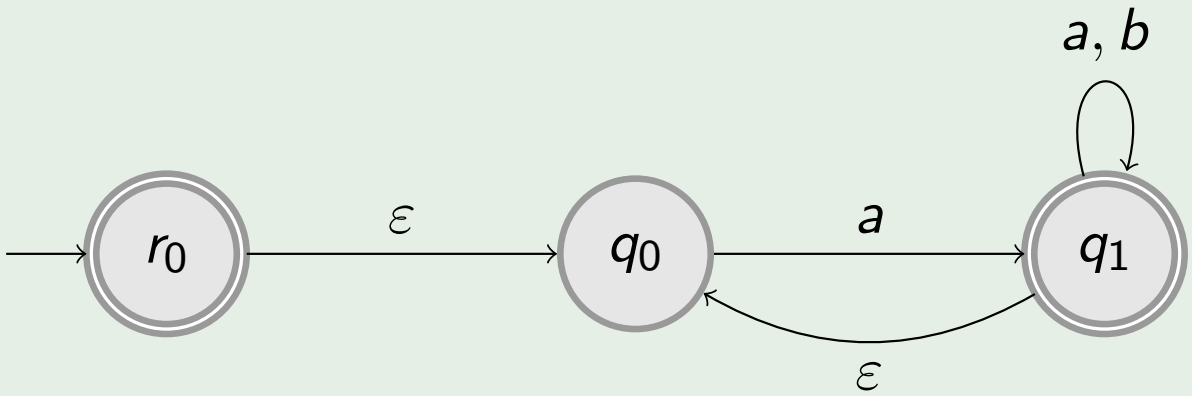
Itération

Exemple (Itération)



Itération

Exemple (Itération)



Bilan

Bilan

Soit L, L_1, L_2 des langages réguliers alors :

- $\overline{L}$ est régulier
- $L_1 \cup L_2$ est régulier
- $L_1 \cap L_2$ est régulier
- $L_1.L_2$ est régulier
- L^* est régulier

Cinquième partie

Expressions régulières

Plan de ce cours

Plan

Expression régulière ★★★

Définition (Expression régulière)

Une **expression régulière** ou **expression rationnelle** sur un alphabet Σ est définie par les règles suivantes :

- $\varnothing$ et ε sont des expressions régulières
- $\forall a \in \Sigma$, a est une expression régulière
- Si e_1 et e_2 sont des expressions régulières, alors :
  - $e_1 + e_2$ est une expression régulière appelée *somme* de e_1 et e_2
  - $e_1 e_2$ est une expression régulière appelée *produit* de e_1 et e_2
- Si e est une expression régulière, alors :
  - e^* est une expression régulière appelée *itérée* de e
  - (e) est une expression régulière

Remarque

$\varnothing$ et ε sont parfois notées 0 et 1.

Langage représenté par une expression régulière ⋆⋆⋆

Définition (Langage représenté par une expression régulière)

Soit e une expression régulière, on définit le **langage représenté par** e, noté $\mathcal{L}(e)$, par :

- $\mathcal{L}(\varnothing) = \varnothing$ et $\mathcal{L}(\varepsilon) = \{\varepsilon\}$
- $\forall a \in \Sigma, \mathcal{L}(a) = \{a\}$
- Si e_1 et e_2 sont des expressions régulières, alors :
  - $\mathcal{L}(e_1 + e_2) = \mathcal{L}(e_1) \cup \mathcal{L}(e_2)$
  - $\mathcal{L}(e_1 e_2) = \mathcal{L}(e_1).\mathcal{L}(e_2)$
- Si e est une expression régulière, alors :
  - $\mathcal{L}(e^*) = \mathcal{L}(e)^*$
  - $\mathcal{L}((e)) = \mathcal{L}(e)$

Langages rationnels

Définition (Langage rationnel)

Un **langage rationnel** est un langage L tel qu'il existe une expression régulière e qui représente le langage L.

$$L = \mathcal{L}(e)$$

Expression régulière

Exemple (Expression régulière)

- L'expression $a(a+b)b$ représente le langage $L = \{aab, abb\}$
- L'expression baa^*b représente le langage $L = \{baa^n b, n \geq 0\}$
- L'expression a^*b^* représente le langage $L = \{a^n b^m, n \geq 0, m \geq 0\}$

Égalité d'expressions régulières ⋆⋆

Définition (Égalité d'expressions régulières)

Deux expressions régulières e_1 et e_2 sont égales, noté $e_1 = e_2$, si elles représentent le même langage :

$$\mathcal{L}(e_1) = \mathcal{L}(e_2)$$

Propriétés des expressions régulières ⋆⋆

Propriétés des opérations sur les expressions régulières

Soit e, e_1, e_2, e_3 des expressions régulières sur un alphabet Σ.

- Propriétés de la somme :
  - $e_1 + (e_2 + e_3) = (e_1 + e_2) + e_3 = e_1 + e_2 + e_3$
  - $e_1 + e_2 = e_2 + e_1$
  - $e + \varnothing = \varnothing + e = e$
  - $e + e = e$
- Propriétés du produit :
  - $(e_1 e_2)e_3 = e_1(e_2 e_3) = e_1 e_2 e_3$
  - $e\varepsilon = \varepsilon e = e$
  - $e\varnothing = \varnothing e = \varnothing$
- Distributivité du produit par rapport à la somme :
  - $e_1(e_2 + e_3) = e_1 e_2 + e_1 e_3$
  - $(e_1 + e_2)e_3 = e_1 e_3 + e_2 e_3$

Propriétés des expressions régulières ⋆⋆

Identités remarquables

Soit e, e_1, e_2 des expressions régulières sur un alphabet Σ.

- $e^* = (e^*)^* = e^*e^* = e^*(e+\varepsilon) = (e+\varepsilon)e^* = \varepsilon + ee^* = \varepsilon + e^*e$
- $(e_1 + e_2)^* = (e_1^*e_2)^*e_1^* = e_1^*(e_2e_1^*)^* = (e_1^*e_2^*)^*$
- $e_1(e_2e_1)^* = (e_1e_2)^*e_1$
- $(e_1^*e_2)^* = \varepsilon + (e_1 + e_2)^*e_2$
- $(e_1e_2^*)^* = \varepsilon + e_1(e_1 + e_2)^*$

Plan

Théorème de Kleene ★★★

Rappels

- Un langage rationnel est représenté par une expression régulière
- Un langage reconnaissable est accepté par un automate d'états fini

Théorème (Théorème de Kleene)

Les langages rationnels sont les langages reconnaissables. Autrement dit :

- *pour toute expression régulière, il existe un automate d'états fini qui reconnaît le langage représenté par l'expression*
- *pour tout automate d'états fini, il existe une expression régulière qui représente le langage accepté par l'automate*

Théorème de Kleene

Preuve du théorème de Kleene

Pour prouver le théorème de Kleene, on utilise des algorithmes constructifs. Il en existe plusieurs pour chaque sens du théorème.

- Automate à partir d'une expression régulière
  - Algorithme de Thompson
- Expression régulière à partir d'un automate
  - Système d'équations régulières
  - Méthode de Brzozowski et McCluskey

Algorithme de Thompson

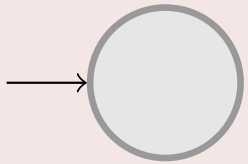
Algorithme de Thompson

L'idée de l'algorithme de Thompson est de construire des automates pour chaque cas dans la définition des expressions régulières.

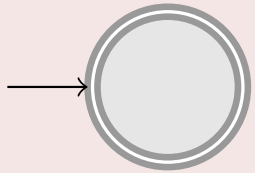
Algorithme de Thompson

Algorithme de Thompson – Cas de base

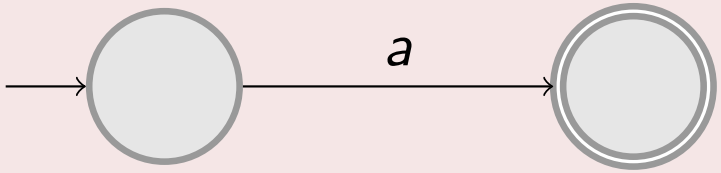
- $\varnothing$



- ε



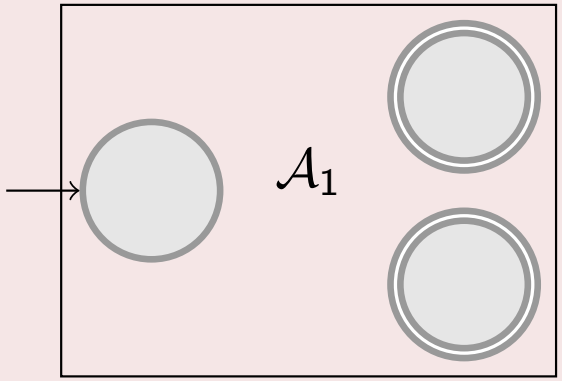
- $a \in \Sigma$

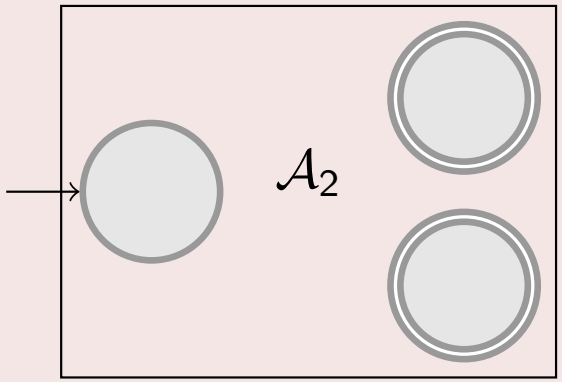


Algorithme de Thompson ⋆⋆

Algorithme de Thompson – Somme $e_1 + e_2$

Soit $\mathcal{A}_1$ et $\mathcal{A}_2$ les automates construits à partir de e_1 et e_2.

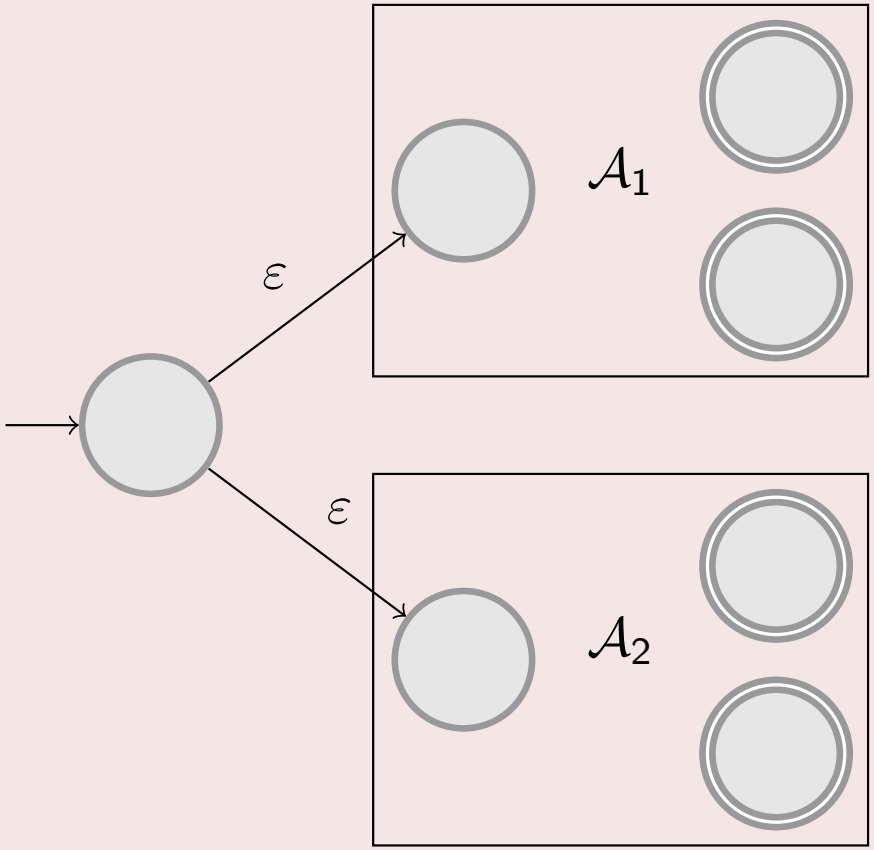




Algorithme de Thompson ⋆⋆

Algorithme de Thompson – Somme $e_1 + e_2$

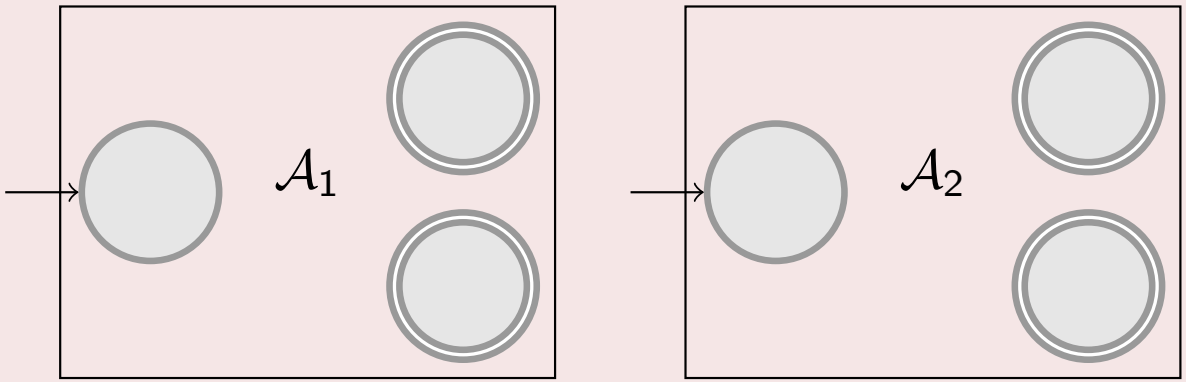
Soit $\mathcal{A}_1$ et $\mathcal{A}_2$ les automates construits à partir de e_1 et e_2.



Algorithme de Thompson

Algorithme de Thompson – Produit $e_1 e_2$

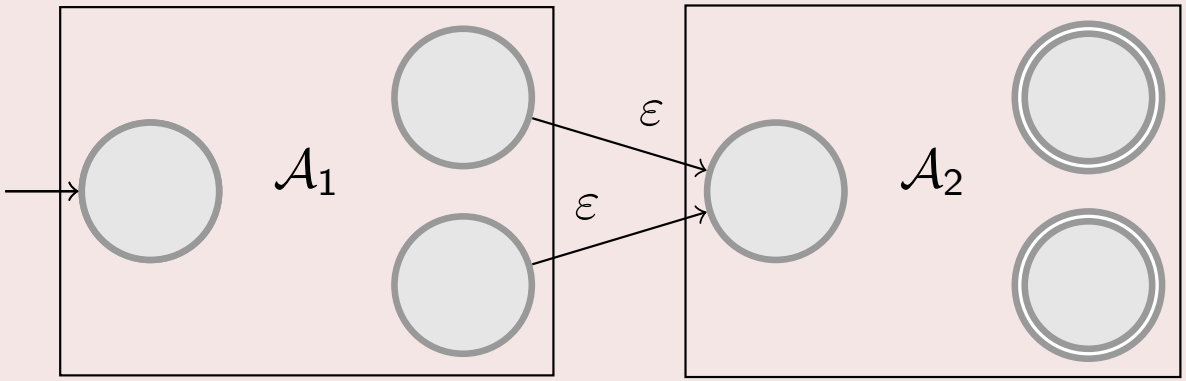
Soit $\mathcal{A}_1$ et $\mathcal{A}_2$ les automates construits à partir de e_1 et e_2.



Algorithme de Thompson

Algorithme de Thompson – Produit $e_1 e_2$

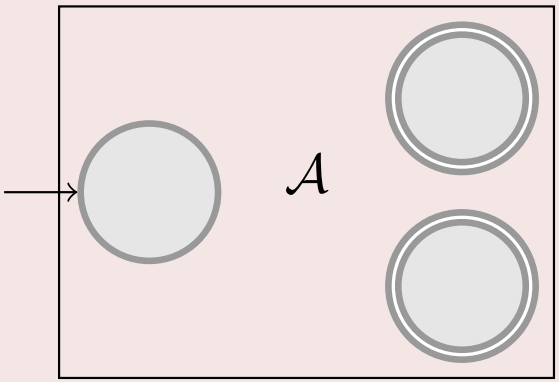
Soit $\mathcal{A}_1$ et $\mathcal{A}_2$ les automates construits à partir de e_1 et e_2.



Algorithme de Thompson

Algorithme de Thompson – Itéré e^*

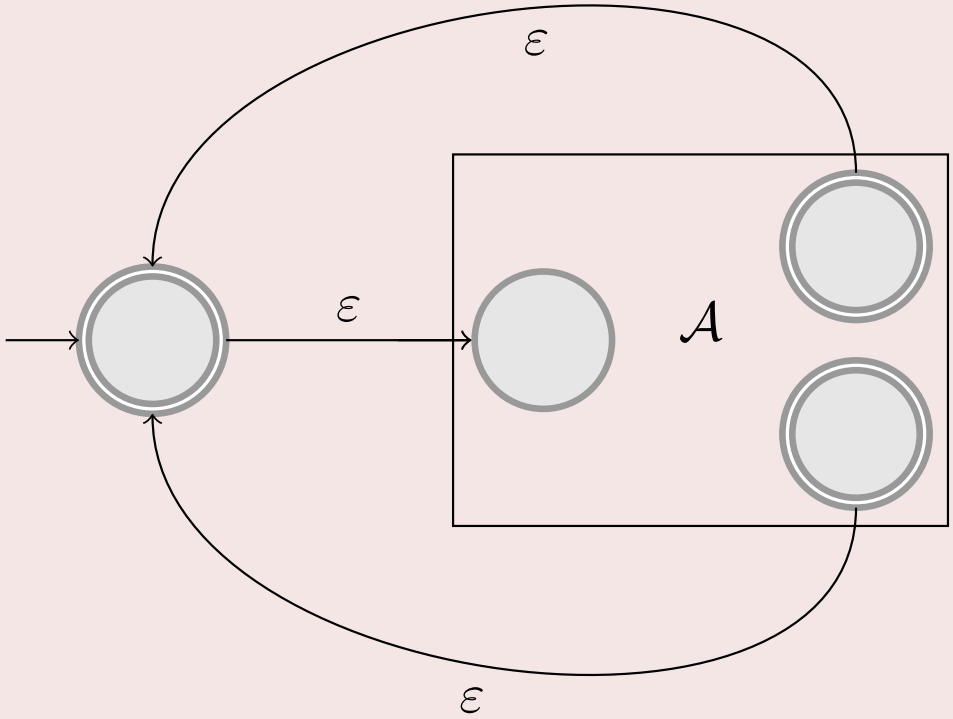
Soit $\mathcal{A}$ l' automate construit à partir de e.



Algorithme de Thompson

Algorithme de Thompson – Itéré e^*

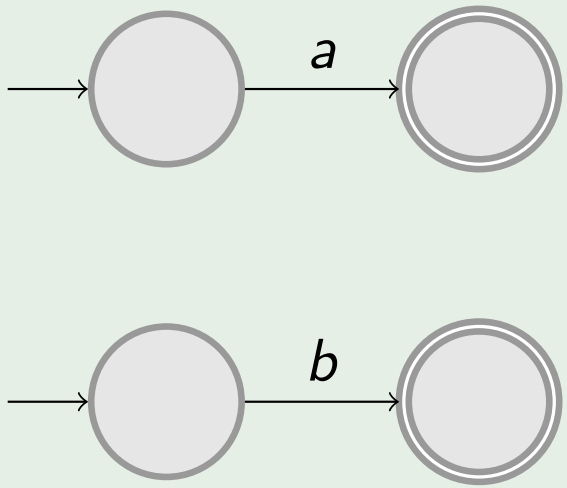
Soit $\mathcal{A}$ l' automate construit à partir de e.



Algorithme de Thompson

Exemple (Algorithme de Thompson)

Soit $e = (a + b)^* ab$

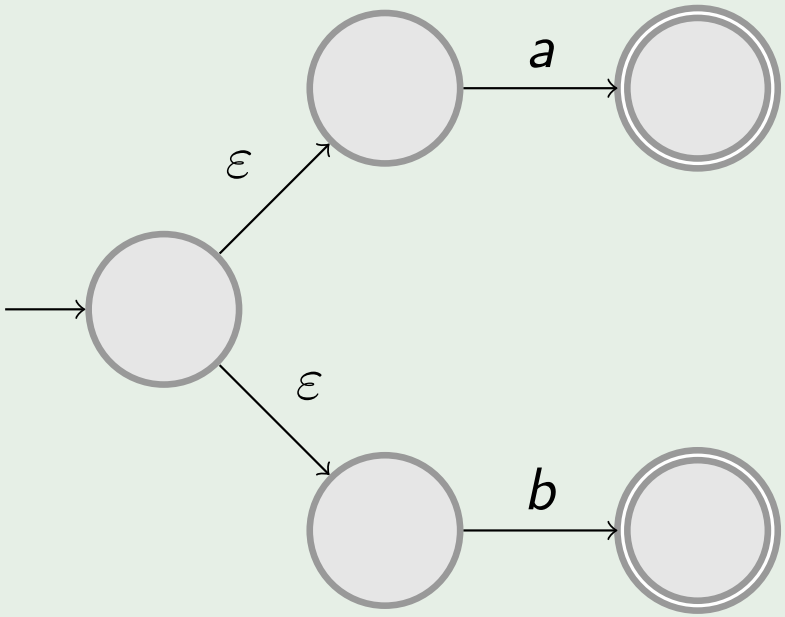


Automates pour a et b

Algorithme de Thompson

Exemple (Algorithme de Thompson)

Soit $e = (a + b)^* ab$

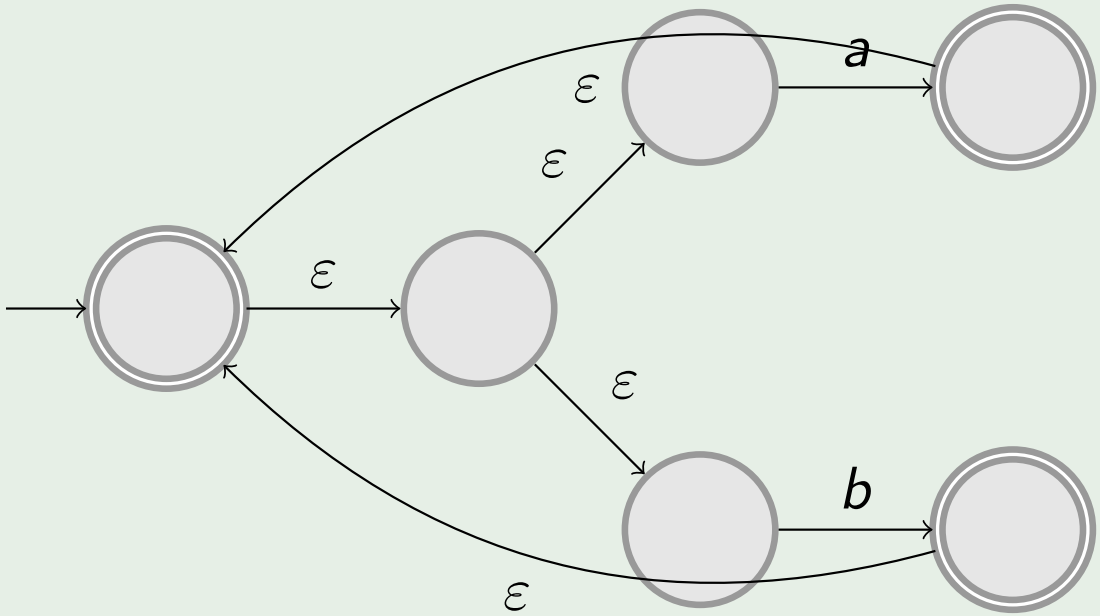


Automate pour $a + b$

Algorithme de Thompson

Exemple (Algorithme de Thompson)

Soit $e = (a + b)^* ab$

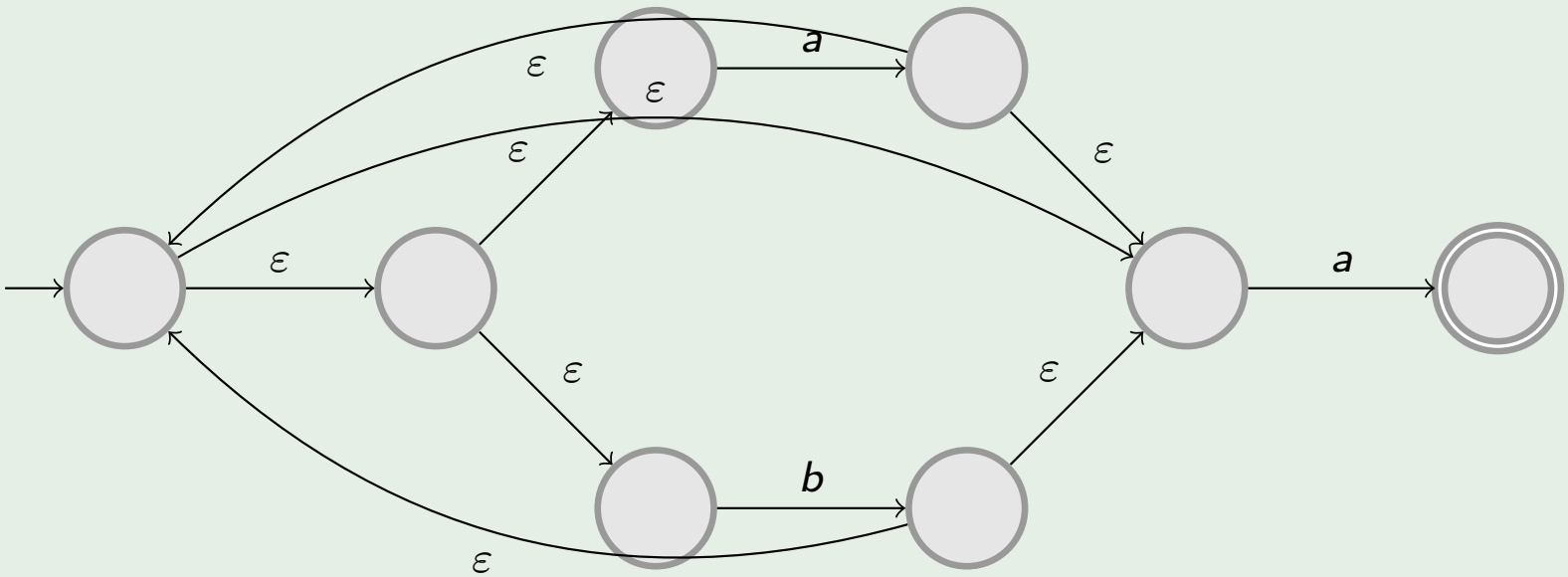


Automate pour $(a + b)^*$

Algorithme de Thompson

Exemple (Algorithme de Thompson)

Soit $e = (a + b)^* ab$

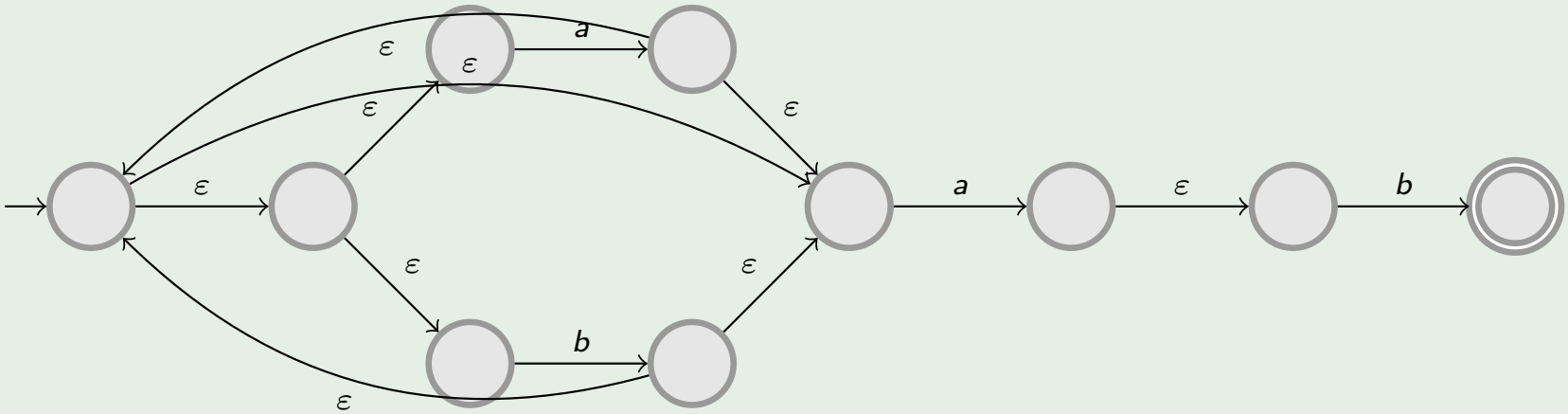


Automate pour $(a + b)^* a$

Algorithme de Thompson

Exemple (Algorithme de Thompson)

Soit $e = (a + b)^* ab$



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

Système d'équations ⋆⋆

Système d'équations

Pour trouver une expression régulière à partir d'un automate :

1. On écrit un système d'équations correspondant à l'automate
2. On résout le système pour trouver l'expression régulière

Système d'équations

Définition (Langage accepté à partir d'un état par un automate)

Le **langage accepté à partir d'un état** q par un automate $\mathcal{A}$, noté $L(q)$ ou L_q, est l'ensemble des mots acceptés à partir de l'état q.

$$L(q) = L_q = \{w \in \Sigma^*, (q, w) \mapsto^* (q', \varepsilon), q' \in Q_F\}$$

Remarque

$$\mathcal{L}(\mathcal{A}) = L(q_0)$$

Système d'équations ⋆⋆

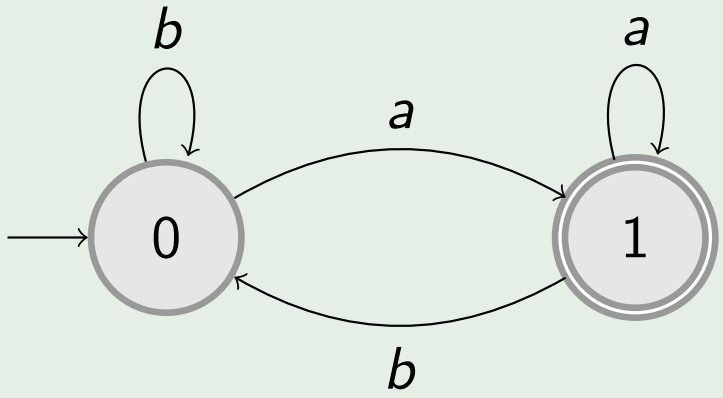
Équation définissant un langage accepté à partir d'un état

Le langage accepté à partir d'un état q par un automate $\mathcal{A}$ est défini par une équation de la forme :

$$L(q) = \bigcup_{(q,a,q')\in\Delta} aL(q') \cup \begin{cases} \varnothing & \text{si } q \notin Q_f \\ \{\varepsilon\} & \text{si } q \in Q_f \end{cases}$$

Système d'équations

Exemple (Équation définissant un langage accepté à partir d'un état)



$$\begin{cases} L_0 = bL_0 \cup aL_1 \\ L_1 = bL_0 \cup aL_1 \cup \{\varepsilon\} \end{cases}$$

Système d'équations

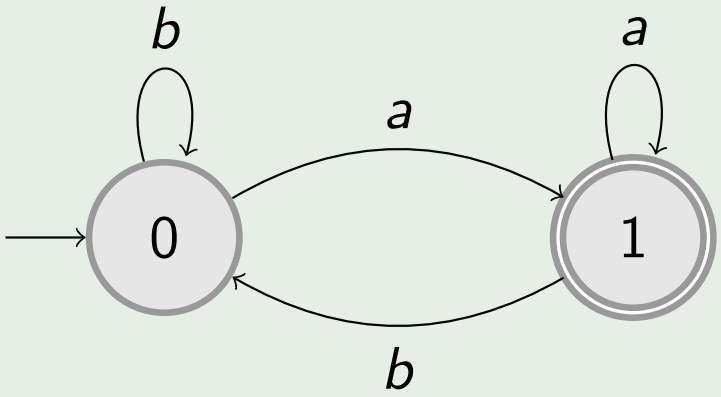
Équation d'une expression régulière à partir d'un état

Soit X_q l'expression régulière qui représente L_q, alors :

$$X_q = \sum_{(q,a,q')\in\Delta} aX_{q'} + \begin{cases} \varnothing & \text{si } q \notin Q_f \\ \varepsilon & \text{si } q \in Q_f \end{cases}$$

Système d'équations

Exemple (Équation d'une expression régulière à partir d'un état)



$$\begin{cases} X_0 = bX_0 + aX_1 \\ X_1 = bX_0 + aX_1 + \varepsilon \end{cases}$$

Système d'équations

★★★

Lemme (Lemme d'Arden)

Soit A et B deux langages tels que $\varepsilon \notin A$, alors l'équation :

$$X = AX + B$$

admet A^*B comme unique solution.

Lemme d'Arden ⋆

Lemme d'Arden.

1. $X = A^*B$ est solution. En effet :
   $AX + B = A(A^*B) + B = (AA^* + \varepsilon)B = A^*B = X$
2. $X = A^*B$ est la plus petite solution. Soit X une autre solution, alors :
   $X = AX + B = A(AX + B) + B = A^2X + AB + B$
   et en continuant :
   $\forall n > 0, X = A^{n+1}X + A^nB + \ldots + AB + B$
   donc, X contient tous les A^nB, donc A^*B, c'est-à-dire $A^*B \subset X$.
3. Si $\varepsilon \notin A$, la solution est unique. Soit X une solution, on sait déjà que $A^*B \subset X$. On sait aussi : $\forall n > 0, X = A^{n+1}X + A^nB + \ldots + AB + B$
   Soit w un mot de longueur n de X, alors $w \notin A^{n+1}X$ puisque les mots de $A^{n+1}X$ sont de longueur au moins $n+1$ car $\varepsilon \notin A$. Donc $w \in A^nB + \ldots + AB + B \subset A^*B$. Donc $X \subset A^*B$. Donc $X = A^*B$.

□

Système d'équations ⋆⋆

Système d'équations

On considère les langages réguliers A_i^j, B_i tels qu'aucun A_i^j ne contiennent ε, alors le système d'équations :

$$\begin{cases} X_1 = A_1^1 X_1 + A_2^1 X_2 + \ldots + A_n^1 X_n + B_1 \\ \ldots \\ X_n = A_1^n X_1 + A_2^n X_2 + \ldots + A_n^n X_n + B_n \end{cases}$$

admet une solution unique.

Résolution d'un système d'équations

Pour résoudre le système, on utilise le lemme d'Arden pour résoudre une des équations, puis on remplace dans les autres équations, et on itère comme pour la résolution par la méthode de Gauss.

Système d'équations

Exemple (Résolution d'un système d'équations)

$$\begin{cases} X_0 = bX_0 + aX_1 \\ X_1 = bX_0 + aX_1 + \varepsilon \end{cases}$$

On résout la seconde équation :

$$X_1 = aX_1 + (bX_0 + \varepsilon) = a^*(bX_0 + \varepsilon)$$

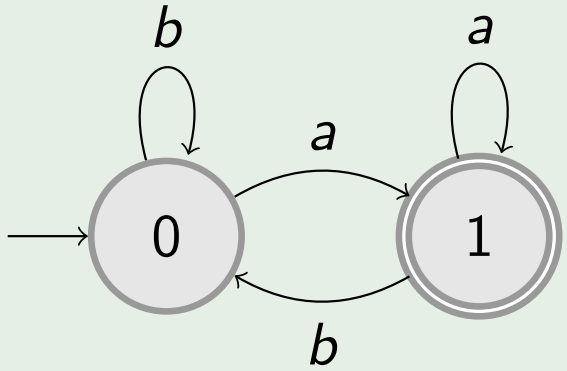
Puis on remplace X_1 dans la première équation :

$$\begin{aligned} X_0 &= bX_0 + a(a^*(bX_0 + \varepsilon)) \\ &= bX_0 + aa^*bX_0 + aa^* \\ &= (b + aa^*b)X_0 + aa^* \\ &= (b + aa^*b)^*aa^* \end{aligned}$$

Système d'équations

Exemple (Résolution d'un système d'équations)

$$\begin{aligned} X_0 &= (b + aa^*b)^*aa^* \\ &= ((\varepsilon + aa^*)b)^*aa^* \\ &= (a^*b)^*aa^* \\ &= (a^*b)^*a^*a \\ &= (a+b)^*a \end{aligned}$$



Méthode de Brzozowski et McCluskey ⋆⋆

Méthode de Brzozowski et McCluskey (1/2)

La **méthode de Brzozowski et McCluskey** est une méthode d'élimination d'états à partir d'un automate $\mathcal{A}$ qui est d'abord transformé en automate généralisé :

- il possède un seul état initial α et un seul état final ω
- les transitions sont étiquetés par des expressions rationnelles
- aucune transition n'entre dans α et aucune transition ne sort de ω

Pour transformer $\mathcal{A}$ en automate généralisé, on ajoute les deux état α et ω et on ajoute des ε-transitions depuis α vers les états initiaux de $\mathcal{A}$ et depuis les états finaux de $\mathcal{A}$ vers ω.
Puis on va réduire l'automate généralisé jusqu'à n'avoir que les états α et ω et une expression régulière entre les deux.

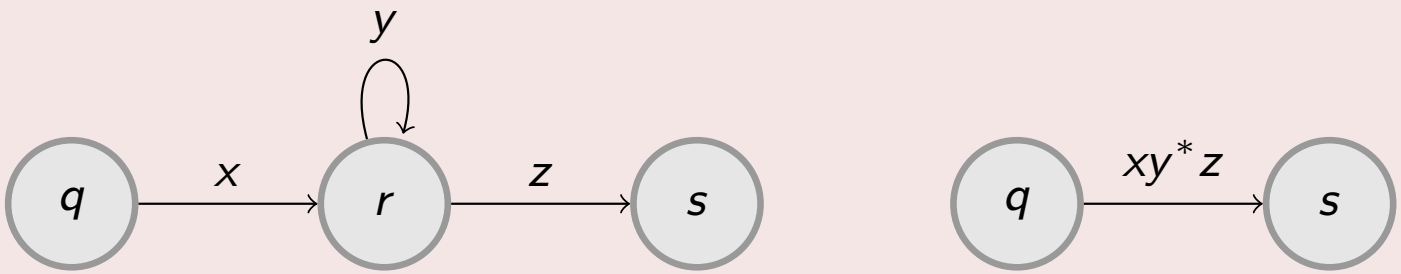
Méthode de Brzozowski et McCluskey ⋆⋆

Méthode de Brzozowski et McCluskey (2/2)

On itère les réductions suivantes :

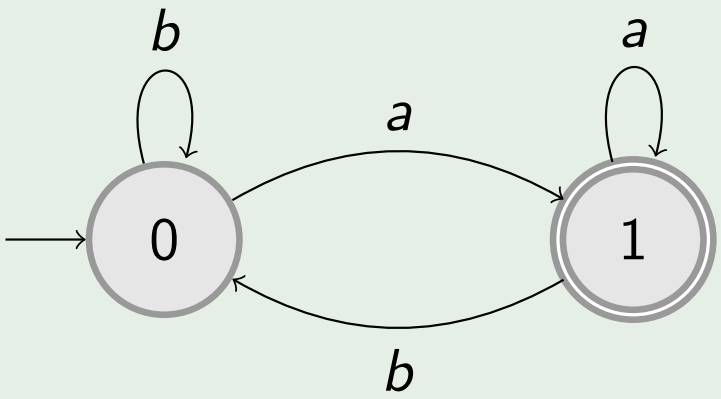
- Réduction des transitions : s'il existe deux transitions (q, x, r) et (q, y, r), on les remplace par une transition $(q, x + y, r)$

- Réduction des états : pour supprimer un état r, on considère tous les couples (q, s) tels qu'il existe des transitions (q, x, r), (r, y, r), (r, z, s) et on remplace ces transitions par (r, xy^*z, s)



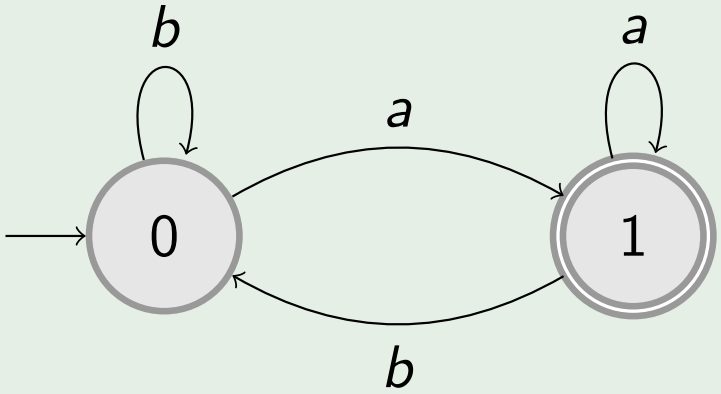
Méthode de Brzozowski et McCluskey

Exemple (Méthode de Brzozowski et McCluskey)



Méthode de Brzozowski et McCluskey

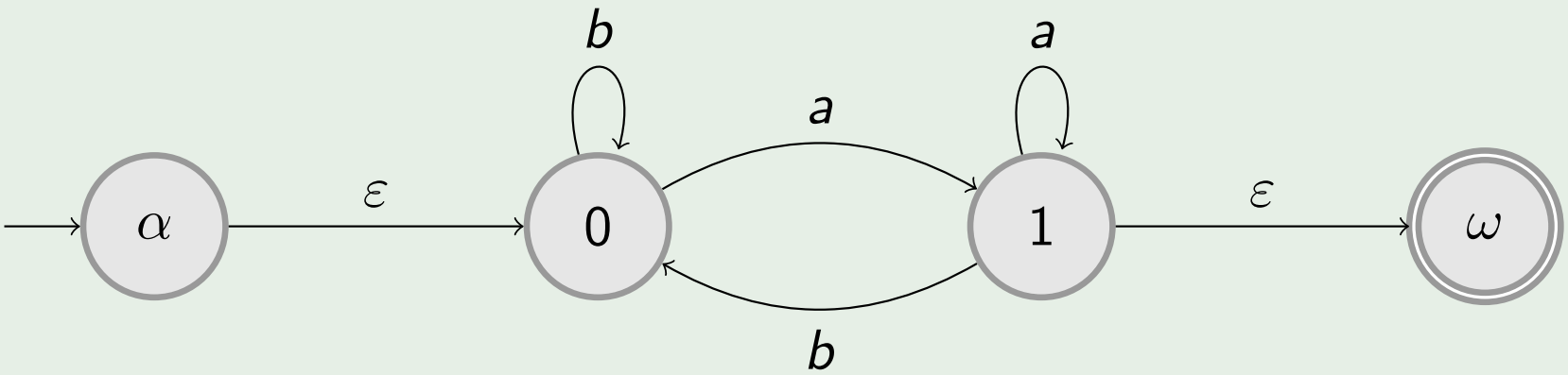
Exemple (Méthode de Brzozowski et McCluskey)



On transforme l'automate en automate généralisé

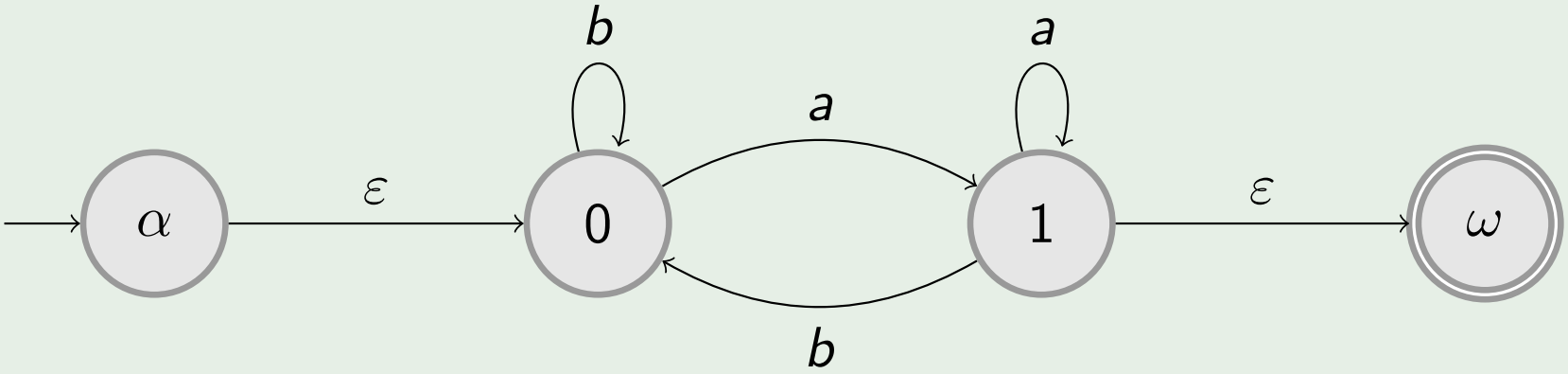
Méthode de Brzozowski et McCluskey

Exemple (Méthode de Brzozowski et McCluskey)



Méthode de Brzozowski et McCluskey

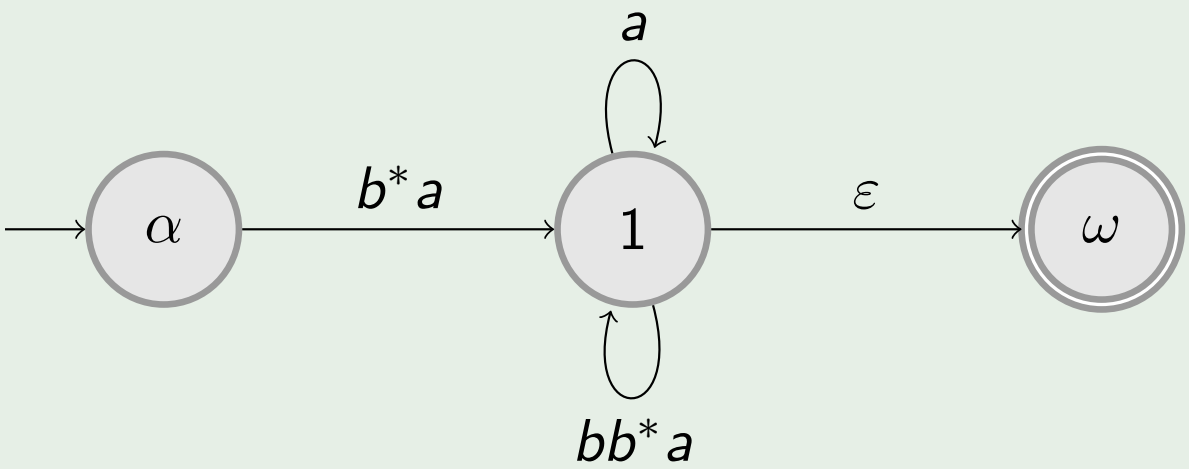
Exemple (Méthode de Brzozowski et McCluskey)



On supprime l'état 0

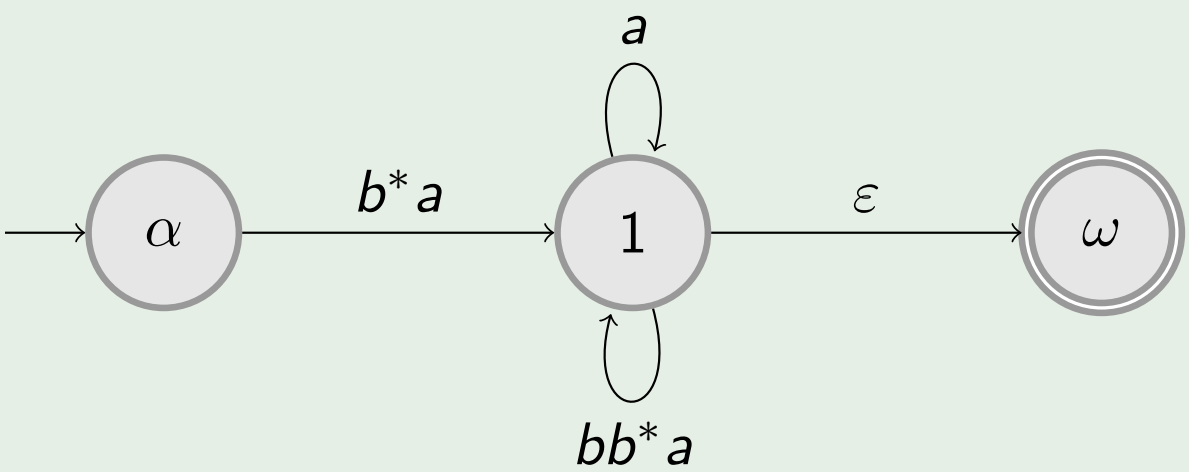
Méthode de Brzozowski et McCluskey

Exemple (Méthode de Brzozowski et McCluskey)



Méthode de Brzozowski et McCluskey

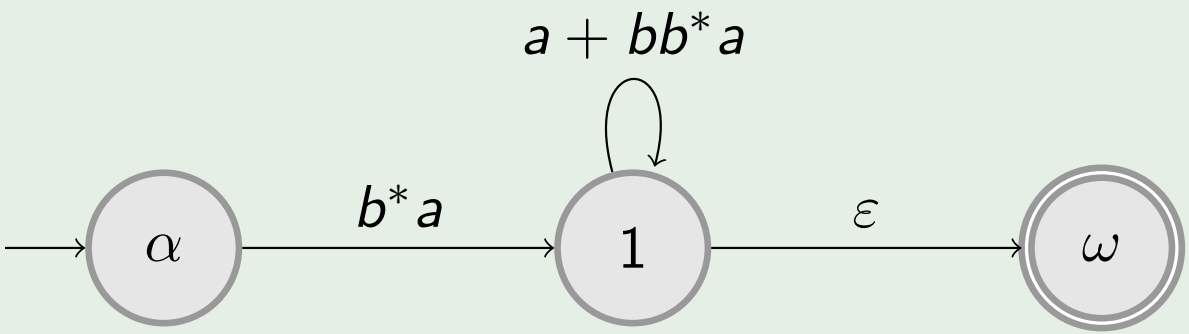
Exemple (Méthode de Brzozowski et McCluskey)



On supprime les transitions a et bb^*a

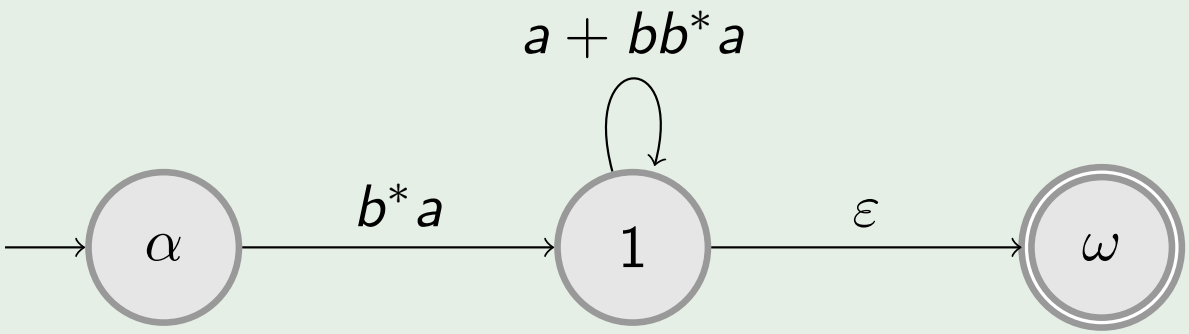
Méthode de Brzozowski et McCluskey

Exemple (Méthode de Brzozowski et McCluskey)



Méthode de Brzozowski et McCluskey

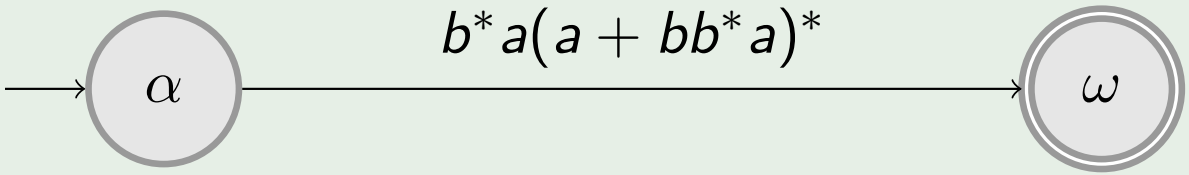
Exemple (Méthode de Brzozowski et McCluskey)



On supprime l'état 1

Méthode de Brzozowski et McCluskey

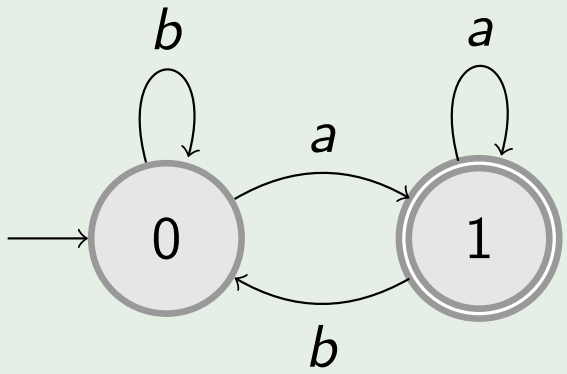
Exemple (Méthode de Brzozowski et McCluskey)



Méthode de Brzozowski et McCluskey

Exemple (Méthode de Brzozowski et McCluskey)

$$\begin{aligned} b^*a(a+bb^*a)^* &= b^*a((\varepsilon+bb^*)a)^* \\ &= b^*a(b^*a)^* \\ &= (b^*a)^*b^*a \\ &= (a+b)^*a \end{aligned}$$



Plan

Lemme de l'étoile

Idée

- Un langage fini est régulier
- Pour un langage infini, la taille des mots est non-bornée
- Un automate comporte un nombre fini d'états

$\rightarrow$ Si l'automate comporte n états, alors lorsqu'un mot w de longueur $|w| \geq n$ est reconnu, on passe nécessairement deux fois par le même état. . . et on peut itérer !

Lemme de l'étoile

★★★

Lemme (Lemme de l'étoile)

Soit L un langage régulier infini. Alors, il existe un entier N, tel que tout mot w de longueur $|w| \geq N$ possède une factorisation $w = xyz$ telle que :

- $y \neq \varepsilon$
- $|xy| \leq N$
- $xy^n z \in L, \forall n \geq 0$

Remarques

- Le lemme de l'étoile s'appelle également lemme de gonflement ou lemme de pompage
- Il existe plusieurs variantes de ce lemme

Lemme de l'étoile ⋆

Lemme de l'étoile.

Soit L un langage régulier infini et $\mathcal{A} = (Q, q_0, \Delta, Q_F)$ un automate fini reconnaissant L. On pose $N = |Q|$.

1. Soit $w \in L$ tel quel $|w| \geq N$. On peut écrire $w = w_1 \dots w_n w'$
2. Alors $(q_0, w_1 \dots w_n w') \mapsto (q_1, w_2 \dots w_n w') \mapsto^* (q_N, w') \mapsto^* (q_f, \varepsilon)$
3. Or, comme $N = |Q|$, il existe $0 \leq i \neq j \leq N$ tel que $q_i = q_j = q$
4. On note $x = w_1 \dots w_i$, $y = w_{i+1} \dots w_j$ et $z = w_{j+1} \dots w_N w'$
5. Alors, $(q_0, xyz) \mapsto^* (q, yz) \mapsto^* (q, z) \mapsto^* (q_f, \varepsilon)$
6. Et donc, $\forall n \geq 0, (y^{n+1}, q) \mapsto^* (y^n, q)$, c'est-à-dire, $xy^n z \in L$
7. Enfin, $|xy| = j \leq N$.

□

Lemme de l'étoile ⋆

Utilisation du lemme de l'étoile

- Attention, ce n'est qu'une simple implication ! Il existe des langages non-réguliers qui ont également cette propriété.

→ On n'utilise jamais le lemme de l'étoile pour montrer qu'un langage est régulier. Question : comment fait-on alors ?

- En revanche, si un langage n'a pas cette propriété, on est certain qu'il n'est pas régulier, par contraposée.

→ C'est l'utilisation principale du lemme de l'étoile : montrer qu'un langage n'est pas régulier.

Langage $a^n b^n$

Proposition (Langage $a^n b^n$)

Le langage $L = \{a^n b^n, n \geq 0\}$ n'est pas régulier.

Langage $a^n b^n$ ⋆⋆

Démonstration.

1. Supposons que $L = \{a^n b^n, n \geq 0\}$ soit régulier.
2. On applique le lemme de l'étoile pour obtenir la constante N.
3. On considère le mot $w = a^N b^N$. On a bien $|w| = 2N > N$.
4. On peut donc décomposer $w = xyz$ avec $|y| > 0$
5. Comme $|xy| \leq N$, x et y ne sont composés que de a.
6. Soit $k = |y| > 0$, alors $xy^n z = a^{N+(n-1)k} b^N$ pour tout entier $n \geq 0$.
7. $xy^n z \in L$ donc $N + (n-1)k = N$ et donc $k = 0$. Contradiction.
8. Donc L n'est pas régulier.

□

Sixième partie

Minimisation d'un automate d'états fini

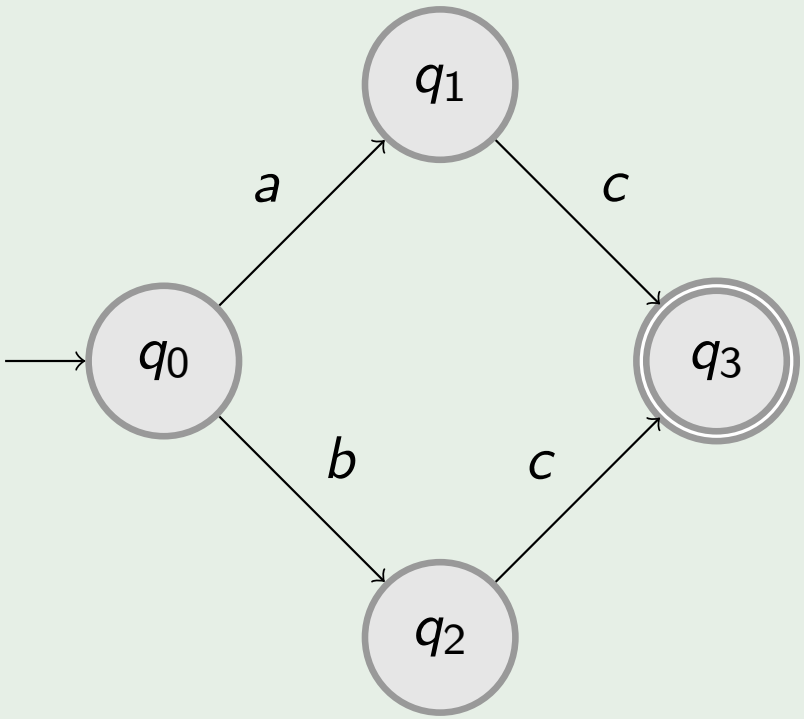
Plan de ce cours

8 Minimisation d'un automate
- Automate minimal
- Test de minimalité
- Automate des résiduels
- Algorithme de Moore

Plan

Exemple introductif

Exemple (Automate minimal)



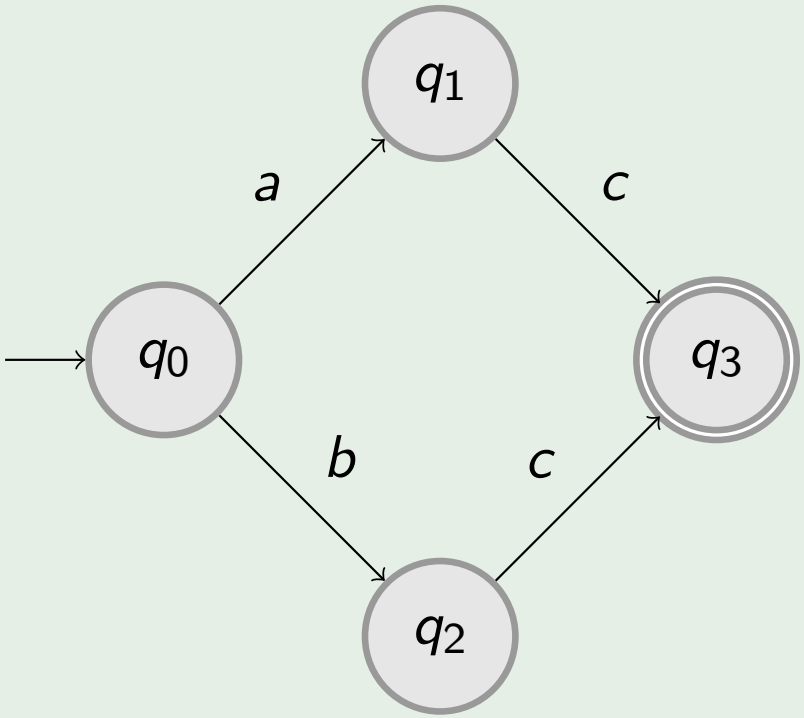
- L'automate est déterministe
- Le langage reconnu est $L = \{ab, ac\}$
- Il existe d'autres automates qui accepte L

$\rightarrow$ Comment trouver l'automate déterministe complet qui a le plus petit nombre d'états ?

$\rightarrow$ Existe-t-il un unique automate minimal ?

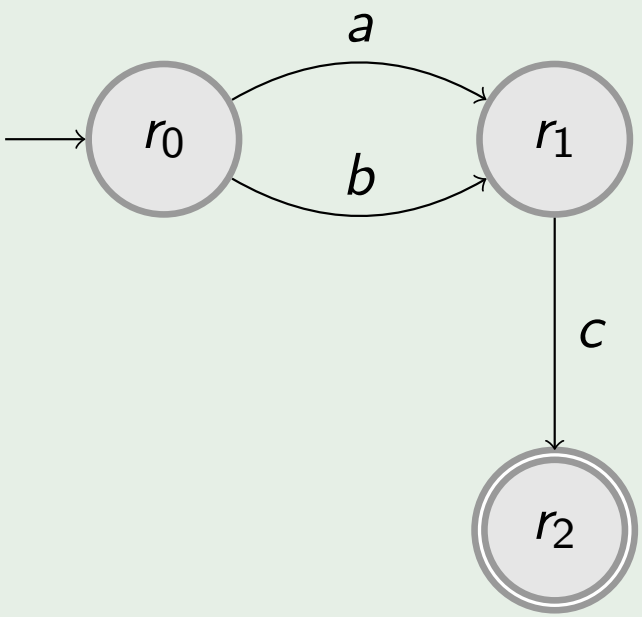
Exemple introductif

Exemple (Automate minimal)



- Idée : le langage reconnu depuis q_1 est le même que le langage reconnu depuis q_2

$\rightarrow$ On peut fusionner les états !



Morphisme d'automate

Définition (Morphisme d'automate)

Soit $\mathcal{A}_Q = (Q, q_0, \delta, Q_F)$ et $\mathcal{A}_R = (R, r_0, \delta, R_F)$ deux automates déterministes complets. Un **morphisme d'automate** est une application $\varphi : Q \to R$ telle que :

- $\varphi(q_0) = r_0$
- $\forall q \in Q, a \in \Sigma, \varphi(\delta(q, a)) = \delta(\varphi(q), a)$
- $\varphi^{-1}(R_F) = Q_F$, c'est-à-dire, $q \in Q_F \iff \varphi(q) \in R_F$

Remarque

Les morphismes d'automates sont définis uniquement pour les automates déterministes complets.

Morphisme d'automate

Proposition (Équivalence par morphisme)

Soient $\mathcal{A}_1$ et $\mathcal{A}_2$ deux automates et φ un morphisme d'automate de $\mathcal{A}_1$ vers $\mathcal{A}_2$. Alors, $\mathcal{A}_1$ et $\mathcal{A}_2$ accepte le même langage :

$$\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$$

Morphisme d'automate

Définition (Automates isomorphes)

Deux automates $\mathcal{A}_1$ et $\mathcal{A}_2$ sont **isomorphes** s'il existe un morphisme d'automate bijectif de $\mathcal{A}_1$ vers $\mathcal{A}_2$. Deux automates isomorphes diffèrent uniquement par le nom des états.

Définition (Automate plus petit)

Un automate $\mathcal{A}_2$ est **plus petit** que $\mathcal{A}_1$ s'il existe un morphisme surjectif $\mathcal{A}_1$ vers $\mathcal{A}_2$. $\mathcal{A}_2$ contient alors moins d'états que $\mathcal{A}_1$.

Remarque

Un morphisme induit une relation d'ordre partiel sur les automates.

Automate minimal

Théorème (Automate minimal)

Il existe un unique automate déterministe complet minimal.

Remarques

- La minimalité porte sur le nombre d'états d'un automate complet !
- L'unicité est à isomorphisme près.

Algorithmes de minimisation

Algorithmes de minimisation

Deux cas se présentent :

1. On dispose d'une expression régulière e
   - On veut construire l'automate minimal qui reconnaît le langage $\mathcal{L}(e)$ représenté par l'expression régulière

   $\rightarrow$ Automate des résiduels
2. On dispose d'un automate déterministe $\mathcal{A}$
   - On veut construire l'automate minimal qui reconnaît le langage $\mathcal{L}(\mathcal{A})$ accepté par l'automate

   $\rightarrow$ Algorithme de Moore

Plan

États séparables

Définition (États séparables)

Deux états q_1 et q_2 sont **séparables** s'il existe un mot w tel que, étant donné $q_1' = \delta(q_1, w)$ et $q_2' = \delta(q_2, w)$, alors $q_1' \in Q_F$ et $q_2' \notin Q_F$, c'est-à-dire le chemin depuis q_1 aboutit à un état final tandis que le chemin depuis q_2 aboutit à un état non-final.

Définition (États inséparables)

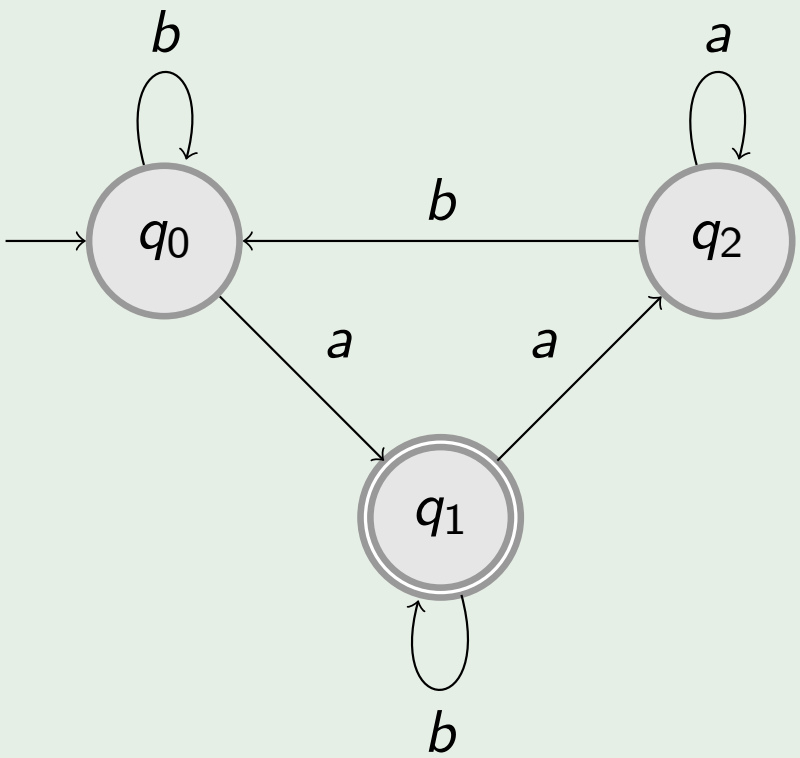
Deux états qui ne sont séparés par aucun mot sont dits **inséparables** ou **indistinguables**.

Remarque

Deux états inséparables reconnaissent le même langage.

États séparables

Exemple (États séparables)



Dans l'automate ci-contre :

- ε sépare q_1 et q_2
- aab ne sépare pas q_1 et q_2
- a sépare q_0 et q_2

Automate minimal

Proposition (Automate minimal)

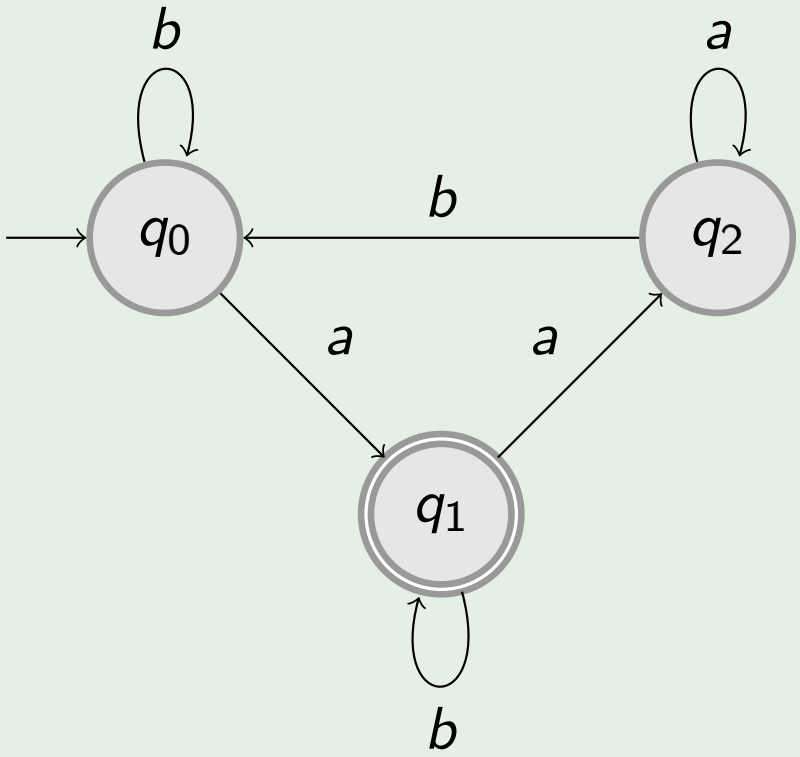
Un automate déterministe complet est minimal si et seulement si ses états sont deux à deux séparables

Remarques

- Un automate minimal ne contient pas deux états inséparables
- Ce résultat donne un moyen de montrer qu'un automate est minimal : il suffit d'exhiber pour chaque paire d'états un mot qui les sépare

Automate minimal

Exemple (Automate minimal)



Cet automate est minimal

Plan

Résiduel

Définition (Résiduel)

Soit un alphabet Σ, un langage L sur l'alphabet Σ et un mot $w \in \Sigma^*$.
Le **résiduel** de L par rapport à w est le langage :

$$w^{-1}L = \{u \in \Sigma^*, w \cdot u \in L\}$$

C'est l'ensemble des mots de L qui commencent par w auxquels on a retiré ce préfixe w

Remarque

Le résiduel est aussi parfois noté L/w

Résiduel

Exemple (Résiduel)

Soit $L = \{ab, ba, aab\}$:

- $a^{-1}L = \{b, ab\}$
- $b^{-1}L = \{a\}$
- $c^{-1}L = \varnothing$
- $(ab)^{-1}L = \varepsilon$

Soit $L = aa(bb + c)^*$:

- $a^{-1}L = a(bb + c)^*$
- $b^{-1}L = \varnothing$
- $c^{-1}L = \varnothing$
- $(aab)^{-1}L = b(bb + c)^*$

Propriétés des résiduels

Propriétés des résiduels

Soit E, F des expressions régulières sur un alphabet Σ, $u, v \in \Sigma^*$, $a \in \Sigma$:

- $\varepsilon^{-1}E = E$
- $\varnothing^{-1}E = \varnothing$
- $(uv)^{-1}E = v^{-1}(u^{-1}E)$
- $u^{-1}(E + F) = u^{-1}E + u^{-1}F$
- $a^{-1}(EF) = \begin{cases} (a^{-1}E)F & \text{si } \varepsilon \notin \mathcal{L}(E) \\ (a^{-1}E)F + a^{-1}F & \text{si } \varepsilon \in \mathcal{L}(E) \end{cases}$
- $a^{-1}E^* = (a^{-1}E)E^*$

Automate et résiduels

Proposition (Automate et résiduel)

Soit $\mathcal{A} = (Q, q_0, \delta, Q_F)$ un automate déterministe, et $L = \mathcal{L}(\mathcal{A})$, alors pour tout $w \in \Sigma^*$ et $q \in Q$:

$$(q_0, w) \mapsto^* (q, \varepsilon) \implies w^{-1}L = L_q$$

Démonstration.

$$\begin{aligned} x \in w^{-1}L &\iff wx \in L \\ &\iff \exists q_f \in Q_F, (q_0, wx) \mapsto^* (q_f, \varepsilon) \\ &\iff \exists q_f \in Q_F, (q_0, wx) \mapsto^* (q, x) \mapsto^* (q_f, \varepsilon) \\ &\iff \exists q_f \in Q_F, (q, x) \mapsto^* (q_f, \varepsilon) \iff x \in L_q \end{aligned}$$

□

Automate et résiduels

Proposition (Automate et résiduel)

L'ensemble des résiduels d'un langage reconnaissable est égal à ses langages L_q, c'est-à-dire si $\mathcal{A} = (Q, q_0, \delta, Q_F)$ est un automate déterministe complet alors :

$$\{w^{-1}\mathcal{L}(\mathcal{A}), w \in \Sigma^*\} = \{L_q, q \in Q\}$$

Conséquences

- Le nombre de résiduels de $\mathcal{L}(\mathcal{A})$ est fini
- Le nombre de résiduels de $\mathcal{L}(\mathcal{A})$ est toujours inférieur à $|Q|$ car il peut exister deux état q_1 et q_2 tels que $L_{q_1} = L_{q_2}$.

Automate des résiduels

Automate des résiduels

Soit L un langage avec un nombre fini de résiduels, alors l'**automate des résiduels** de L est l'automate déterministe $\mathcal{A}_L = \{Q, q_0, \delta, Q_F\}$:

- $Q = \{w^{-1}L, w \in \Sigma^*\}$
- $q_0 = L$
- $\delta(w^{-1}L, a) = (wa)^{-1}L$
- $Q_F = \{w^{-1}L, w \in L\}$

Automate des résiduels

Proposition (Automate des résiduels)

L'automate $\mathcal{A}_L$ des résiduels de L reconnaît le langage L.

Conséquence importante

L'automate des résiduels est minimal pour le langage L

Théorème (Théorème de Myhill-Nerode)

Un langage est reconnaissable si et seulement s'il a un nombre fini de résiduels

Construction de l'automate des résiduels

Construction de l'automate des résiduels

Soit un langage L avec n résiduels :

- On numérote les états de 0 à $n-1$
- Le résiduel L_i correspond à l'état i
- L'état initial 0 correspond à $L_0 = \varepsilon^{-1}L = L$
- Il existe une transition (i, a, j) si et seulement si $L_j = a^{-1}L_i$
- i est un état final si et seulement si $\varepsilon \in L_i$

Automate des résiduels

Exemple (Automate des résiduels)

$$L = 1^*0(0+1)$$

	0	1
L		

Automate des résiduels

Exemple (Automate des résiduels)

$$L = 1^*0(0+1)$$

- $0^{-1}L = (0+1) = L_1$
- $1^{-1}L = L$

	0	1
L	L_1	L
L_1		

Automate des résiduels

Exemple (Automate des résiduels)

$$L = 1^*0(0+1)$$

- $0^{-1}L = (0+1) = L_1$
- $1^{-1}L = L$
- $0^{-1}L_1 = \varepsilon = L_2$
- $1^{-1}L_1 = \varepsilon = L_2$

	0	1
L	L_1	L
L_1	L_2	L_2
L_2		

Automate des résiduels

Exemple (Automate des résiduels)

$$L = 1^*0(0+1)$$

- $0^{-1}L = (0+1) = L_1$
- $1^{-1}L = L$
- $0^{-1}L_1 = \varepsilon = L_2$
- $1^{-1}L_1 = \varepsilon = L_2$
- $0^{-1}L_2 = \varnothing = L_3$
- $1^{-1}L_2 = \varnothing = L_3$

	0	1
L	L_1	L
L_1	L_2	L_2
L_2	L_3	L_3
L_3		

Automate des résiduels

Exemple (Automate des résiduels)

$$L = 1^*0(0+1)$$

- $0^{-1}L = (0+1) = L_1$
- $1^{-1}L = L$
- $0^{-1}L_1 = \varepsilon = L_2$
- $1^{-1}L_1 = \varepsilon = L_2$
- $0^{-1}L_2 = \varnothing = L_3$
- $1^{-1}L_2 = \varnothing = L_3$
- $0^{-1}L_3 = \varnothing = L_3$
- $1^{-1}L_3 = \varnothing = L_3$

	0	1
L	L_1	L
L_1	L_2	L_2
L_2	L_3	L_3
L_3	L_3	L_3

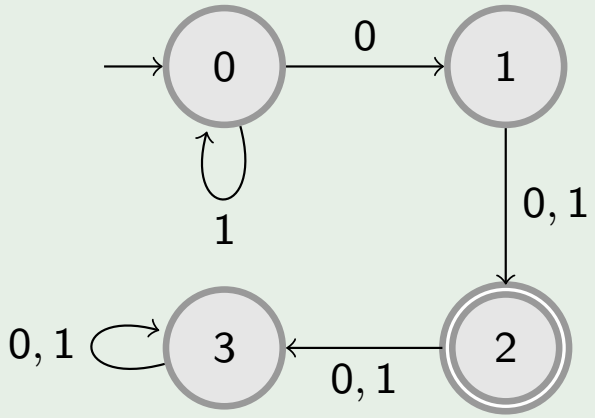
Automate des résiduels

Exemple (Automate des résiduels)

$$L = 1^*0(0+1)$$

- $0^{-1}L = (0+1) = L_1$
- $1^{-1}L = L$
- $0^{-1}L_1 = \varepsilon = L_2$
- $1^{-1}L_1 = \varepsilon = L_2$
- $0^{-1}L_2 = \varnothing = L_3$
- $1^{-1}L_2 = \varnothing = L_3$
- $0^{-1}L_3 = \varnothing = L_3$
- $1^{-1}L_3 = \varnothing = L_3$

	0	1
L	L_1	L
L_1	L_2	L_2
L_2	L_3	L_3
L_3	L_3	L_3



Automate des résiduels

Exemple (Automate des résiduels)

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

	a	b
L		

Automate des résiduels

Exemple (Automate des résiduels)

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

- $a^{-1}L = (a+b)^*ab + b = L_1$
- $b^{-1}L = (a+b)^*ab = L$

	a	b
L	L_1	L
L_1		

Automate des résiduels

Exemple (Automate des résiduels)

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

- $a^{-1}L = (a+b)^*ab + b = L_1$
- $b^{-1}L = (a+b)^*ab = L$
- $a^{-1}L_1 = (a+b)^*ab + b + \varnothing = L_1$
- $b^{-1}L_1 = (a+b)^*ab + \varepsilon = L_2$

	a	b
L	L_1	L
L_1	L_1	L_2
L_2		

Automate des résiduels

Exemple (Automate des résiduels)

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

- $a^{-1}L = (a+b)^*ab + b = L_1$
- $b^{-1}L = (a+b)^*ab = L$
- $a^{-1}L_1 = (a+b)^*ab + b + \varnothing = L_1$
- $b^{-1}L_1 = (a+b)^*ab + \varepsilon = L_2$
- $a^{-1}L_2 = (a+b)^*ab + b + \varnothing = L_1$
- $b^{-1}L_2 = (a+b)^*ab + \varnothing = L$

	a	b
L	L_1	L
L_1	L_1	L_2
L_2	L_1	L

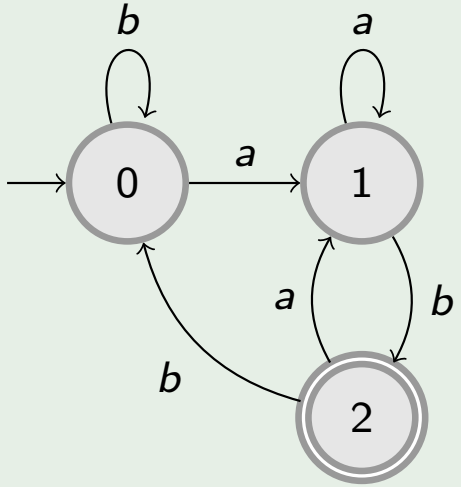
Automate des résiduels

Exemple (Automate des résiduels)

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

- $a^{-1}L = (a+b)^*ab + b = L_1$
- $b^{-1}L = (a+b)^*ab = L$
- $a^{-1}L_1 = (a+b)^*ab + b + \varnothing = L_1$
- $b^{-1}L_1 = (a+b)^*ab + \varepsilon = L_2$
- $a^{-1}L_2 = (a+b)^*ab + b + \varnothing = L_1$
- $b^{-1}L_2 = (a+b)^*ab + \varnothing = L$

	a	b
L	L_1	L
L_1	L_1	L_2
L_2	L_1	L



Plan

Congruence de Nérode

Définition (Congruence de Nérode)

Soit $\mathcal{A}$ un automate fini déterministe. Deux états q_1 et q_2 sont **équivalents par la congruence de Nérode**, noté $q_1 \cong q_2$, si les langages acceptés à partir de q_1 et q_2 sont identiques :

$$q_1 \cong q_2 \iff L(q_1) = L(q_2)$$

Autrement dit, soit $w \in \Sigma^*$, soient $q_1' = \delta(q_1, w)$ et $q_2' = \delta(q_2, w)$ alors :

$$q_1 \cong q_2 \iff \forall w \in \Sigma^*, \begin{cases} q_1' \in Q_F \text{ et } q_2' \in Q_F, \text{ ou} \\ q_1' \notin Q_F \text{ et } q_2' \notin Q_F \end{cases}$$

Remarque

$q_1 \cong q_2$ signifie que q_1 et q_2 sont inséparables.

Propriétés de la congruence de Nérode

Propriétés de la congruence de Nérode

La congruence de Nérode est :

- une relation d'équivalence :
  - réflexive : $q \cong q$
  - symétrique : $q_1 \cong q_2 \iff q_2 \cong q_1$
  - transitive : $q_1 \cong q_2$ et $q_2 \cong q_3 \implies q_1 \cong q_3$
- une congruence sur les automates :
  - elle est compatible avec δ : $q_1 \cong q_2 \implies \forall a \in \Sigma, \delta(q_1, a) = \delta(q_2, a)$
  - elle sature Q_F : $q_1 \cong q_2 \implies (q_1 \in Q_F \iff q_2 \in Q_F)$

Classe d'équivalence

Définition (Classe d'équivalence)

Soit q un état d'un automate $\mathcal{A}$, note $[q]$ la **classe d'équivalence** de l'état q, c'est-à-dire l'ensemble des états qui sont équivalents à q par la congruence de Nérode.

$$[q] = \{q' \in Q, q' \cong q\}$$

Remarque

Ces classes d'équivalence forment une partition de l'ensemble des états Q

Automate quotient

Définition (Automate quotient)

Soit $\mathcal{A} = (Q, q_0, \delta, Q_F)$ un automate déterministe, l'**automate quotient** pour la relation $\cong$ est l'automate $\mathcal{A}' = (Q', q'_0, \delta', Q'_F)$ avec :

- $Q' = \{[q], q \in Q\}$
- $q'_0 = [q_0]$
- $\delta'([q], a) = [\delta(q, a)]$
- $Q'_F = \{[q], q \in Q_F\}$

Proposition (Automate minimal)

Soit $\mathcal{A}'$ l'automate quotient d'un automate $\mathcal{A}$ pour la relation $\cong$, alors :

- $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$
- *$\mathcal{A}'$ est minimal*

Congruence de Nérode

Définition (Congruence de Nérode pour les mots de longueur $\leq n$)

Soit $\mathcal{A}$ un automate fini déterministe. Deux états q_1 et q_2 sont **équivalents par la congruence de Nérode** pour les mots de longueur au plus n, noté $q_1 \cong_n q_2$, si les langages pour les mots de longueur au plus n acceptés à partir de q_1 et q_2 sont identiques :

$$q_1 \cong_n q_2 \iff \forall u \in \Sigma^*, |u| \leq n, (u \in L(q_1) \iff u \in L(q_2))$$

Propriétés

La relation $\cong_n$ est également une relation d'équivalence et une congruence sur les automates.

Congruence de Nérode

Propriété importante de $\cong_n$

$$q_1 \cong_{n+1} q_2 \iff q_1 \cong_n q_2 \text{ et } \forall a \in \Sigma, \delta(q_1, a) = \delta(q_2, a)$$

Proposition (Congruence de Nérode)

Il existe n tel que $\cong_n$ est identique à $\cong_{n+1}$ et alors $\cong_n$ est la congruence de Nérode $\cong$.

Conséquence

Cette propriété nous permet de concevoir un algorithme pour minimiser pas à pas un automate : l'algorithme de Moore

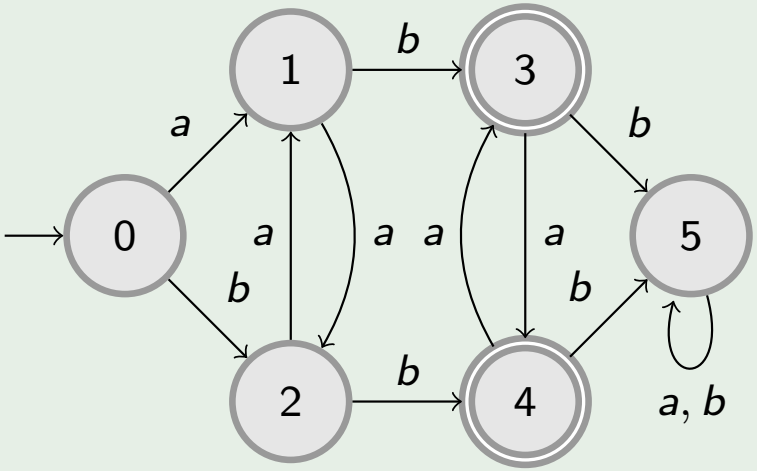
Algorithme de Moore

Algorithme de Moore

1. On contruit un tableau avec les états dans chaque colonne
2. On construit $\cong_0$: $q_1 \cong_0 q_2 \iff (q_1 \in Q_F \iff q_2 \in Q_F)$
   - On numérote I la classe des états non-finaux
   - On numérote II la classe des états finaux
3. On construit $\cong_{n+1}$ à partir de $\cong_n$:
   1. On contruit une ligne pour chaque lettre $a \in \Sigma$
   2. On associe à chaque lette a et à chaque état q le numéro de la classe correspondant à l'état $\delta(q, a)$ dans $\cong_n$
   3. On numérote les colonnes en fonction de $\cong_n$ et du résultat obtenu à l'étape précédente pour obtenir les classes de $\cong_{n+1}$
   4. Si $\cong_{n+1}$ est identique à $\cong_n$, on s'arrête

Algorithme de Moore

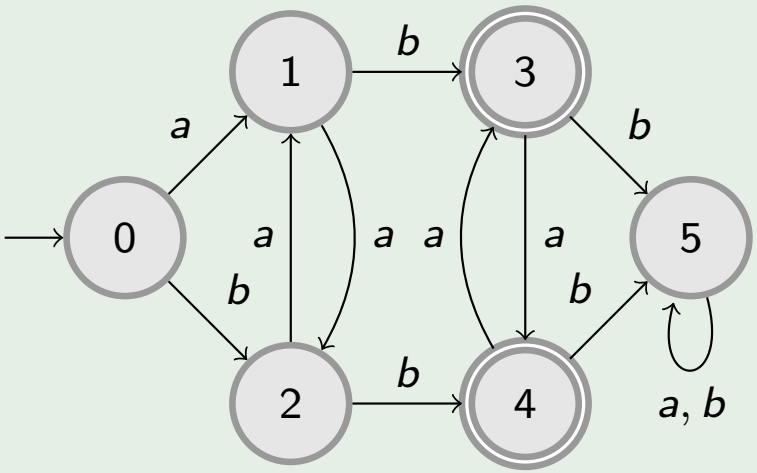
Exemple (Algorithme de Moore)



	0	1	2	3	4	5

Algorithme de Moore

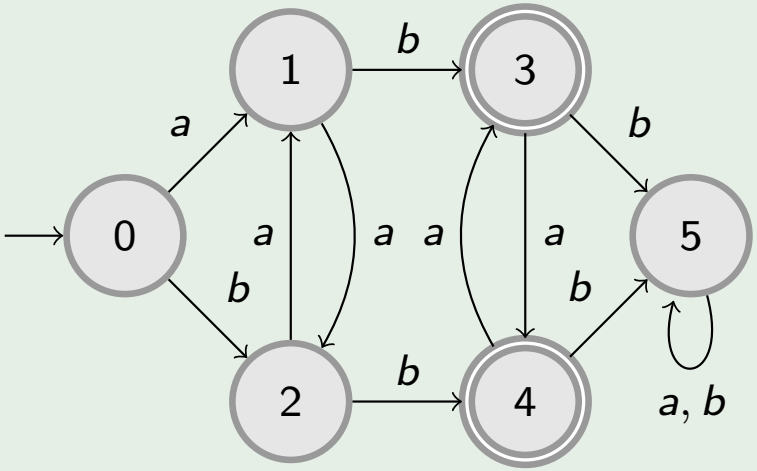
Exemple (Algorithme de Moore)



	0	1	2	3	4	5
$\cong_0$	I	I	I	II	II	I

Algorithme de Moore

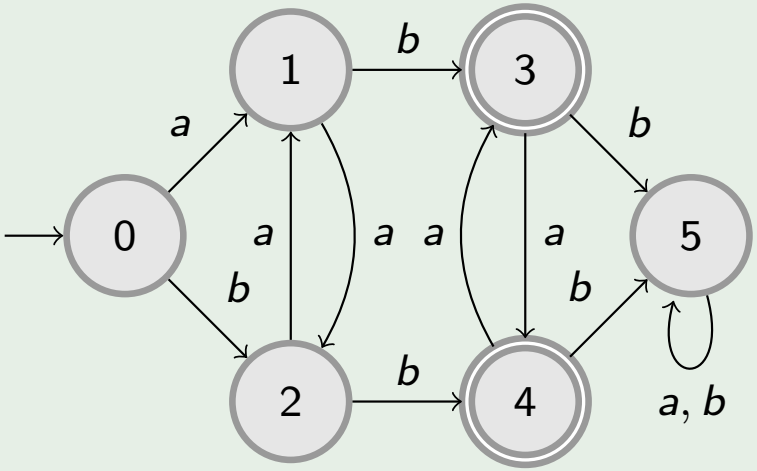
Exemple (Algorithme de Moore)



	0	1	2	3	4	5
$\cong_0$	I	I	I	II	II	I
a	I	I	I	II	II	I
b	I	II	II	I	I	I

Algorithme de Moore

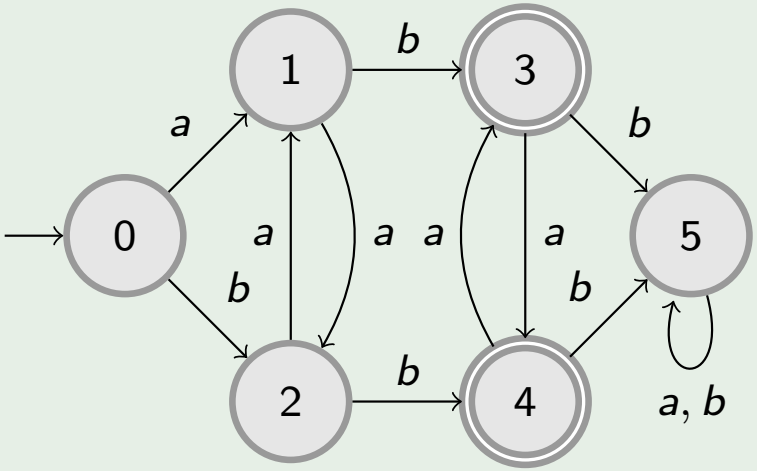
Exemple (Algorithme de Moore)



	0	1	2	3	4	5
$\cong_0$	I	I	I	II	II	I
a	I	I	I	II	II	I
b	I	II	II	I	I	I
$\cong_1$	I	II	II	III	III	I

Algorithme de Moore

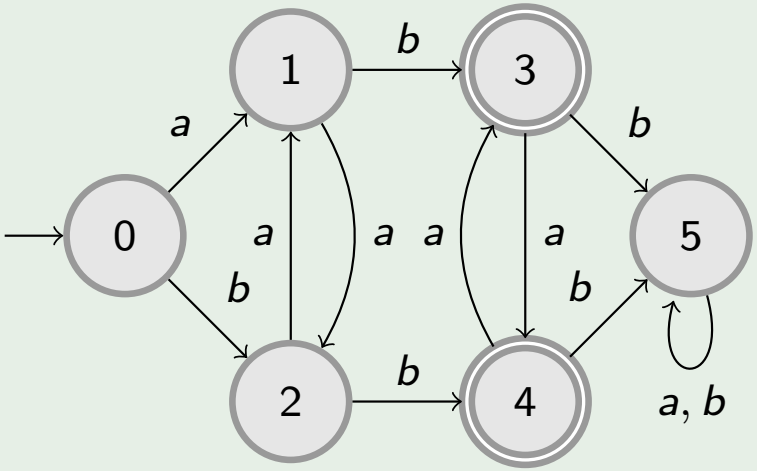
Exemple (Algorithme de Moore)



	0	1	2	3	4	5
$\cong_0$	I	I	I	II	II	I
a	I	I	I	II	II	I
b	I	II	II	I	I	I
$\cong_1$	I	II	II	III	III	I
a	II	II	II	III	III	I
b	II	III	III	I	I	I

Algorithme de Moore

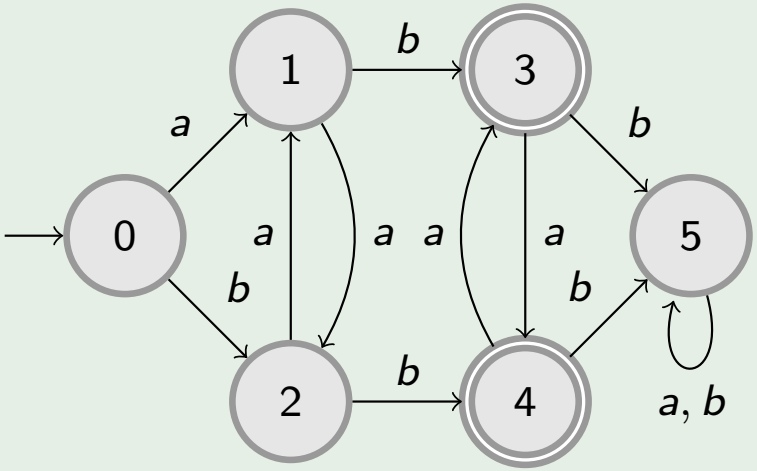
Exemple (Algorithme de Moore)



	0	1	2	3	4	5
$\cong_0$	I	I	I	II	II	I
a	I	I	I	II	II	I
b	I	II	II	I	I	I
$\cong_1$	I	II	II	III	III	I
a	II	II	II	III	III	I
b	II	III	III	I	I	I
$\cong_2$	I	II	II	III	III	IV

Algorithme de Moore

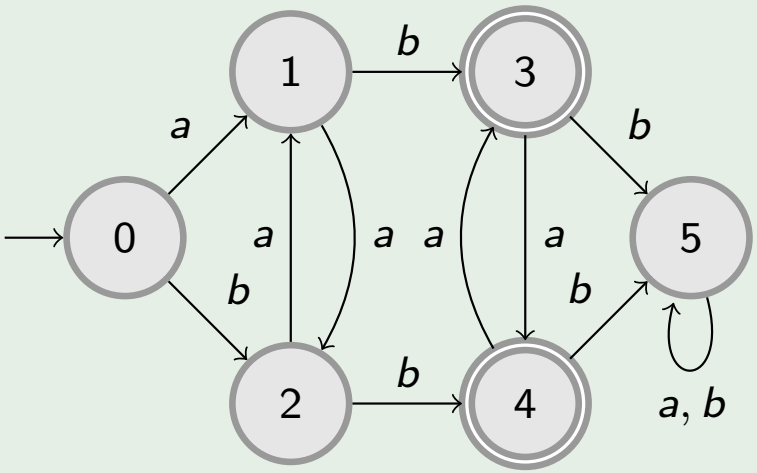
Exemple (Algorithme de Moore)



	0	1	2	3	4	5
$\cong_0$	I	I	I	II	II	I
a	I	I	I	II	II	I
b	I	II	II	I	I	I
$\cong_1$	I	II	II	III	III	I
a	II	II	II	III	III	I
b	II	III	III	I	I	I
$\cong_2$	I	II	II	III	III	IV
a	II	II	II	III	III	IV
b	II	III	III	IV	IV	IV

Algorithme de Moore

Exemple (Algorithme de Moore)

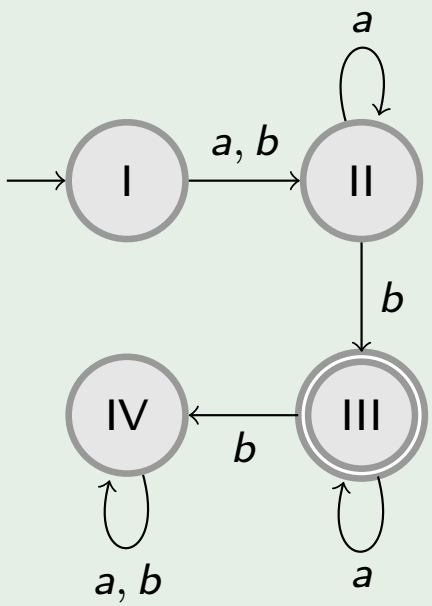


	0	1	2	3	4	5
$\cong_0$	I	I	I	II	II	I
a	I	I	I	II	II	I
b	I	II	II	I	I	I
$\cong_1$	I	II	II	III	III	I
a	II	II	II	III	III	I
b	II	III	III	I	I	I
$\cong_2$	I	II	II	III	III	IV
a	II	II	II	III	III	IV
b	II	III	III	IV	IV	IV
$\cong_3$	I	II	II	III	III	IV

Algorithme de Moore

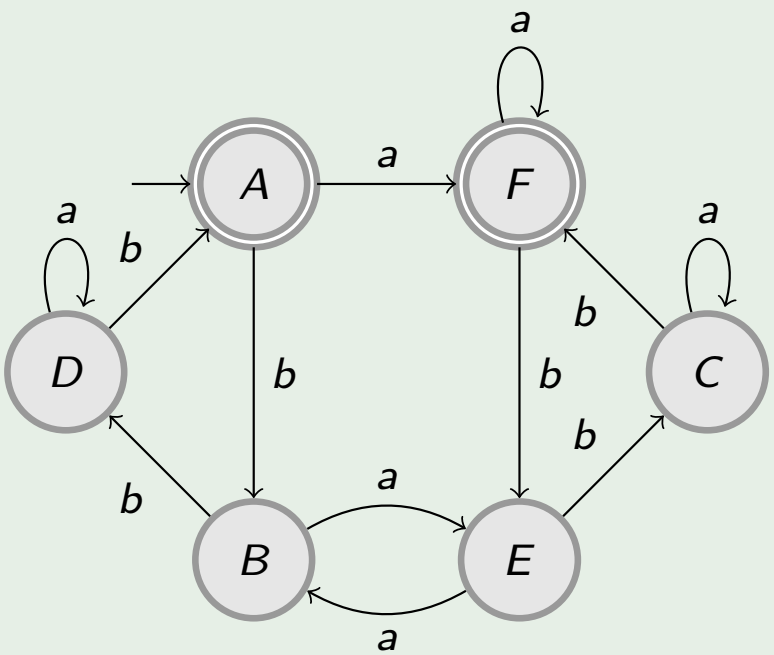
Exemple (Algorithme de Moore)

	0	1	2	3	4	5
$\cong_0$	I	I	I	II	II	I
a	I	I	I	II	II	I
b	I	II	II	I	I	I
$\cong_1$	I	II	II	III	III	I
a	II	II	II	III	III	I
b	II	III	III	I	I	I
$\cong_2$	I	II	II	III	III	IV
a	II	II	II	III	III	IV
b	II	III	III	IV	IV	IV
$\cong_3$	I	II	II	III	III	IV



Algorithme de Moore

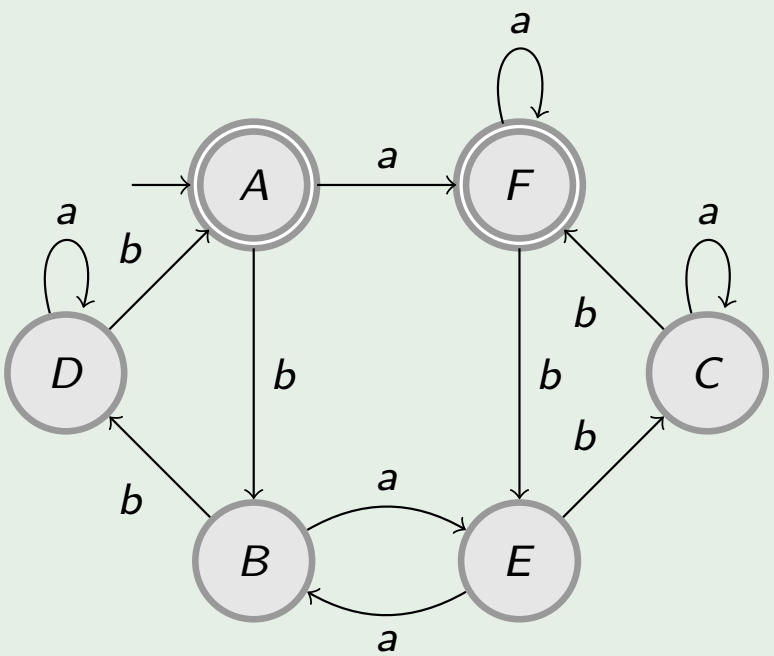
Exemple (Algorithme de Moore)



	A	B	C	D	E	F

Algorithme de Moore

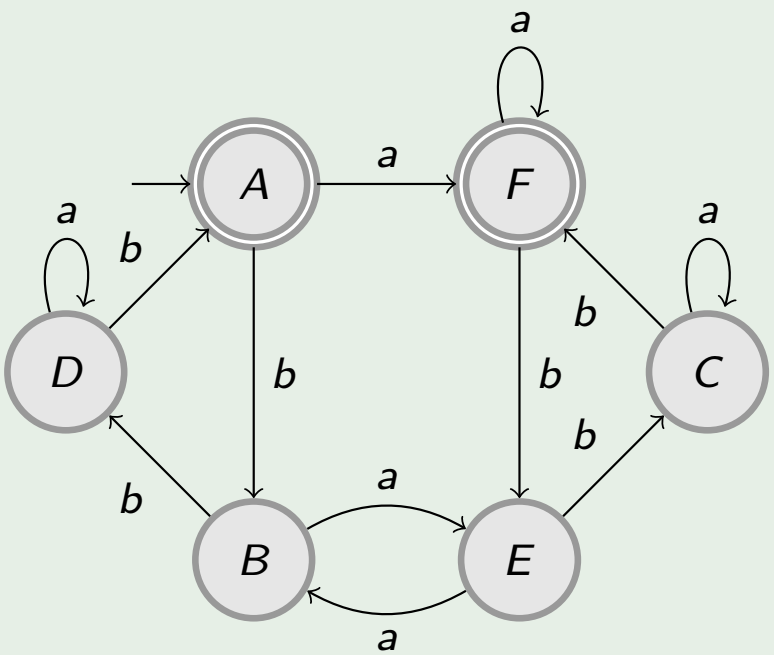
Exemple (Algorithme de Moore)



	A	B	C	D	E	F
$\cong_0$	I	II	II	II	II	I

Algorithme de Moore

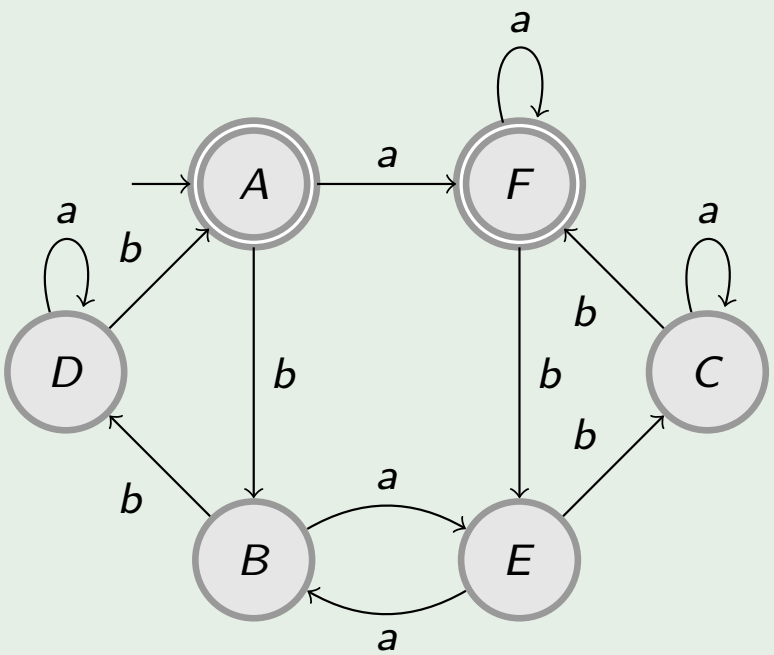
Exemple (Algorithme de Moore)



	A	B	C	D	E	F
$\cong_0$	I	II	II	II	II	I
a	I	II	II	II	II	I
b	II	II	I	I	II	II

Algorithme de Moore

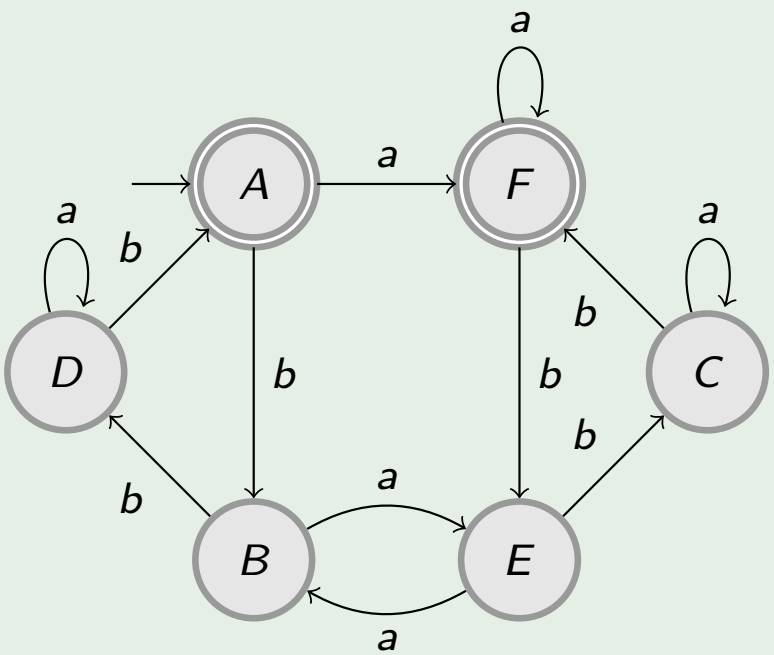
Exemple (Algorithme de Moore)



	A	B	C	D	E	F
$\cong_0$	I	II	II	II	II	I
a	I	II	II	II	II	I
b	II	II	I	I	II	II
$\cong_1$	I	II	III	III	II	I

Algorithme de Moore

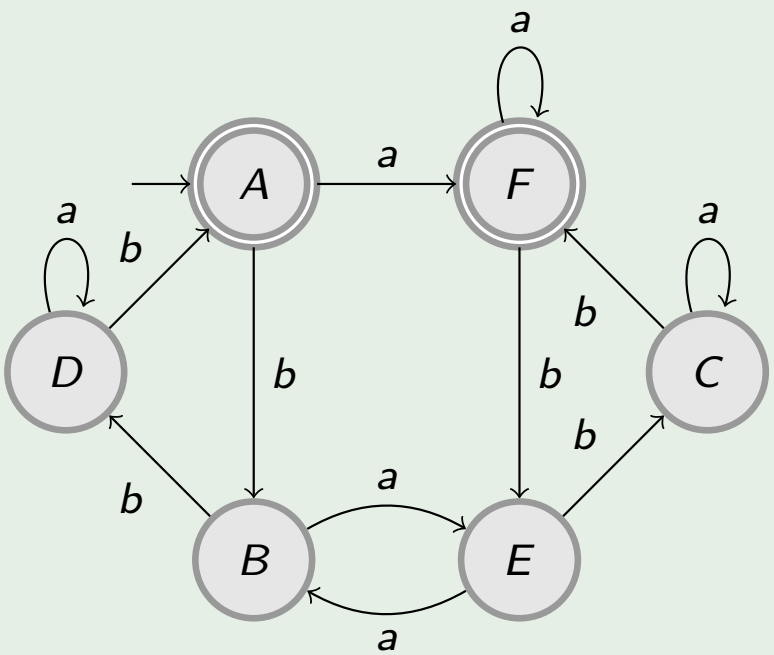
Exemple (Algorithme de Moore)



	A	B	C	D	E	F
$\cong_0$	I	II	II	II	II	I
a	I	II	II	II	II	I
b	II	II	I	I	II	II
$\cong_1$	I	II	III	III	II	I
a	I	II	III	III	II	I
b	II	III	I	I	III	II

Algorithme de Moore

Exemple (Algorithme de Moore)

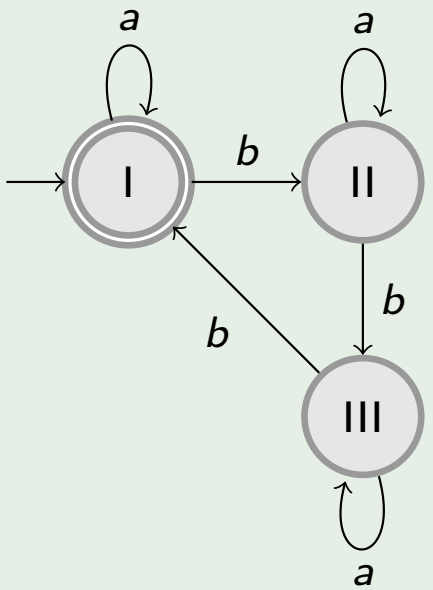


	A	B	C	D	E	F
$\cong_0$	I	II	II	II	II	I
a	I	II	II	II	II	I
b	II	II	I	I	II	II
$\cong_1$	I	II	III	III	II	I
a	I	II	III	III	II	I
b	II	III	I	I	III	II
$\cong_2$	I	II	III	III	II	I

Algorithme de Moore

Exemple (Algorithme de Moore)

	A	B	C	D	E	F
$\cong_0$	I	II	II	II	II	I
a	I	II	II	II	II	I
b	II	II	I	I	II	II
$\cong_1$	I	II	III	III	II	I
a	I	II	III	III	II	I
b	II	III	I	I	III	II
$\cong_2$	I	II	III	III	II	I



Septième partie

Grammaires algébriques

Plan de ce cours

Plan

Motivation

Motivation

- Le pouvoir d'expression des langages réguliers est limité !
  - On ne sait pas définir un langage régulier pour décrire les expressions bien parenthésées ou pour reconnaître $a^n b^n$

  $\rightarrow$ On a besoin de langages plus expressifs mais avec moins de propriétés
- Les langages de programmation sont algébriques !

Rappel : Grammaire algébrique (type 2) ⋆⋆

Définition (Grammaire algébrique (type 2))

Une **grammaire de type 2** ou **grammaire algébrique** ou **grammaire hors contexte** est une grammaire où les règles sont de la forme :

$$A \to \alpha$$

avec $A \in N$ et $\alpha \in V^*$, la partie gauche est réduite à un non-terminal.

Définition (Langage algébrique (type 2))

Un **langage algébrique** est un langage engendré par une grammaire algébrique.

Rappel : Grammaire algébrique (type 2)

Exemple (Grammaire algébrique (type 2))

La grammaire $G = (\{S\}, \{a, b\}, S, R)$ avec R :

- $S \to \varepsilon \mid aSb$

engendre le langage $L = \{a^n b^n, n \geq 0\}$

Plan

Langage régulier ⋆⋆

Proposition (Langage régulier)

Tout langage régulier est algébrique

Remarques

- La réciproque est fausse ! Il existe des langages algébriques qui ne sont pas réguliers !
  - Exemple : $L = \{a^n b^n, n \geq 0\}$
- Un langage algébrique non-régulier :
  - n'est pas reconnu par un automate fini
  - n'est pas représenté par une expression régulière
  - n'est pas engendré par une grammaire régulière

Propriétés des langages algébriques

Union

★★

Proposition (Union de langages algébriques)

Soient L_1 et L_2 deux langages algébriques, alors l'union $L_1 \cup L_2$ est un langage algébrique.

Démonstration.

Soit $G_1 = (N_1, T_1, S_1, R_1)$ et $G_2 = (N_2, T_2, S_2, R_2)$ des grammaires engendrant L_1 et L_2, alors la grammaire

$$G = (N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, S, R_1 \cup R_2 \cup \{ S \to S_1 \mid S_2\})$$

engendre $L_1 \cup L_2$ □

Propriétés des langages algébriques

Produit

★★

Proposition (Produit de langages algébriques)

Soient L_1 et L_2 deux langages algébriques, alors le produit $L_1.L_2$ est un langage algébrique.

Démonstration.

Soit $G_1 = (N_1, T_1, S_1, R_1)$ et $G_2 = (N_2, T_2, S_2, R_2)$ des grammaires engendrant L_1 et L_2, alors la grammaire

$$G = (N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, S, R_1 \cup R_2 \cup \{ S \to S_1 S_2\})$$

engendre $L_1.L_2$ □

Propriétés des langages algébriques

Itération

★★

Proposition (Itération d'un langage algébrique)

Soient L un langage algébriques, alors l'itération L^* est un langage algébrique.

Démonstration.

Soit $G = (N, T, S, R)$ une grammaire engendrant L, alors la grammaire

$$G' = (N \cup \{S'\}, T, S', R \cup \{\ S' \to SS' \mid \varepsilon\})$$

engendre L^* □

Propriétés des langages algébriques

Intersection

★★

Proposition (Intersection de langages algébriques)

Soient L_1 et L_2 deux langages algébriques, alors l'intersection $L_1 \cap L_2$ ***n'est pas forcément*** *un langage algébrique.*

Contre-exemple

Soient $L_1 = \{a^n b^n c^m, n > 0, m > 0\}$ et $L_2 = \{a^n b^m c^m, n > 0, m > 0\}$.
Ces deux langages sont algébriques. Leur intersection

$$L = L_1 \cap L_2 = \{a^n b^n c^n, n > 0\}$$

n'est pas un langage algébrique (voir plus loin).

Propriétés des langages algébriques

Complémentaire

★★

Proposition (Complémentaire d'un langage algébrique)

Soient L un langage algébriques, alors le complémentaire $\overline{L}$ ***n'est pas forcément*** *un langage algébrique.*

Démonstration.

Si $\overline{L}$ était algébrique, alors l'intersection de deux langages algébriques serait algébrique car $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ □

Propriétés des langages algébriques ⋆⋆

Décidabilité de l'appartenance et du test du vide

Il existe des algorithmes qui permettent de savoir si :

- un mot w appartient à un langage algébrique L
- un langage algébrique L est vide

Indécidabilité

Soient L_1, L_2, L des langages algébriques sur un alphabet Σ, R un langage régulier. Il n'existe pas d'algorithmes qui permettent de savoir si :

- $L_1 \cap L_2 = \varnothing$
- $L = \Sigma^*$
- $L_1 = L_2$
- $L_1 \subset L_2$
- $L = R$
- $R \subset L$
- $\overline{L}$ est algébrique
- $L_1 \cap L_2$ est algébrique
- L est régulier

Plan

Symbole non-terminal utile

Définition (Symbole non-terminal utile)

Soit $G = (N, T, S, R)$. Un symbole non-terminal $X \in N$ est :

- **productif** si $\mathcal{L}_G(X) \neq \varnothing$
- **accessible** s'il existe α et β tels que $S \rightarrow^* \alpha X \beta$
- **utile** s'il est productif et qu'il existe α et β tels que $S \rightarrow^* \alpha X \beta$, et α et β ne contiennent que des symboles non-terminaux productifs

Grammaire réduite

Définition (Grammaire réduite)

Une grammaire est **réduite** si tous ses symboles non-terminaux sont utiles.

Remarques

- On ne change pas le langage engendré en supprimant les non-terminaux inutiles.
- Si l'axiome est inutile, la grammaire engendre le langage vide.

Calcul des symboles non-terminaux productifs ⋆⋆

Calcul des symboles non-terminaux productifs

1. Calculer l'ensemble V_0 des symboles non-terminaux X pour lesquels il existe une règle $X \to \alpha, \alpha \in A^*$
2. Calculer l'ensemble V_{i+1} formé de V_i et des symboles non-terminaux X pour lesquels il existe une règle $X \to \alpha, \alpha \in (A \cup V_i)^*$
3. Arrêter lorsque $V_{i+1} = V_i$. Cet ensemble est l'ensemble des symboles non-terminaux productifs.

Calcul des symboles non-terminaux accessibles ⋆⋆

Calcul des symboles non-terminaux accessibles

1. Poser $W_0 = \{S\}$
2. Calculer l'ensemble W_{i+1} formé de W_i et des symboles non-terminaux X pour lesquels il existe une règle $Y \to \alpha X \beta, Y \in W_i$
3. Arrêter lorsque $W_{i+1} = W_i$. Cet ensemble est l'ensemble des symboles non-terminaux accessibles.

Algorithme de réduction d'une grammaire ⋆⋆

Algorithme de réduction d'une grammaire

1. Déterminer l'ensemble des symboles non-terminaux productifs
2. Supprimer l'ensemble des symboles non-terminaux non-productifs, ainsi que les règles dans lesquels ils apparaissent
3. Si l'axiome S est improductif, la grammaire réduite a pour seul symbole non-terminal S et un ensemble de règles vide
4. Si l'axiome S est productif, déterminer tous les symboles non-terminaux accessibles à partir de S. Ceci permet d'obtenir l'ensemble des symboles non-terminaux utiles.
5. Supprimer l'ensemble des symboles non-terminaux inutiles, ainsi que les règles dans lesquels ils apparaissent

$\rightarrow$ La grammaire ainsi obtenue est réduite

Réduction d'une grammaire

Exemple (Réduction d'une grammaire)

1. Soit $G = (\{S, X, Y\}, \{a, b\}, S, R)$ avec R :
   - $S \to a \mid X$
   - $X \to XY$
   - $Y \to b$
2. Symboles non-terminaux productifs : $V_0 = \{S, Y\} = V_1$
3. On supprime X qui est improductif :
   - $S \to a$
   - $Y \to b$
4. Symboles non-terminaux accessibles : $W_0 = \{S\} = W_1$
5. On supprime Y qui est non-accessible :
   - $S \to a$

$\to$ La grammaire est réduite

Réduction d'une grammaire

Exemple (Réduction d'une grammaire)

1. Soit $G = (\{S, W, X, Y, Z\}, \{a, b, c\}, S, R)$ avec R :
   - $S \to XYZW$
   - $X \to cX$
   - $Y \to ab$
   - $Z \to cYa \mid WSW$
   - $W \to \varepsilon$
2. Symboles non-terminaux productifs :
   1. $V_0 = \{W, Y\}$
   2. $V_1 = V_0 \cup \{Z\} = V_2$
3. S est improductif

$\to$ La grammaire réduite est $G = (\{S\}, \{a, b, c\}, S, \varnothing)$

Plan

Grammaire propre

Définition (Grammaire propre)

Une **grammaire propre** ne contient :

- pas de règle unitaire (ou renommage), c'est-à-dire la grammaire ne contient pas de règle de la forme :

$$A \to B$$

- pas de production vide, c'est-à-dire la grammaire ne contient pas de règle de la forme :

$$A \to \varepsilon$$

avec $A, B \in N$ des non-terminaux

Grammaire propre

Proposition (Langage engendré par une grammaire propre)

Tout langage algébrique ne contenant pas le mot vide ε peut être engendré par une grammaire propre.

Remarque importante

Si le langage L contient le mot vide ε :

1. On contruit une grammaire propre $G = (N, T, S, R)$ qui engendre le langage $L \setminus \{\varepsilon\}$
2. La grammaire $G' = (N \cup \{S'\}, T, S', R \cup \{S' \to S \mid \varepsilon\})$ engendre le langage L.
   - $\rightarrow$ l'axiome S' n'apparaît jamais en partie droite d'une règle
   - $\rightarrow$ le seul non-terminal qui peut dériver ε est S'

Symbole non-terminal annulable ⋆⋆

Définition (Symbole non-terminal annulable)

Un symbole non-terminal X est **annulable** si $X \to^* \varepsilon$

Remarque

X est annulable si :

- il existe une règle $X \to \varepsilon$
- il existe une règle $X \to Y_1 Y_2 \dots Y_n$, avec $Y_1, Y_2, \dots, Y_n$ des symboles non-terminaux annulables

Cette propriété nous donne un algorithme pour trouver l'ensemble N_ε des symboles non-terminaux annulables

Calcul de l'ensemble N_ε

Calcul de l'ensemble des symboles non-terminaux annulables

1. Calculer l'ensemble N_0 des symboles non-terminaux X tels qu'il existe une règle $X \to \varepsilon$
2. Calculer l'ensemble N_{i+1} formé de l'ensemble N_i et des symboles non-terminaux X tels qu'il existe une règle $X \to \alpha$ avec $\alpha \in N_i^*$
3. Arrêter lorsque $N_{i+1} = N_i$. Cet ensemble est l'ensemble N_ε des symboles non-terminaux annulables.

Algorithme d'élimination des productions vides ⋆⋆

Algorithme d'élimination des productions vides

1. Calculer l'ensemble N_ε des symboles non-terminaux annulables
2. Remplacer chaque règle $X \to \alpha$ par toutes les règles obtenues en remplaçant, de toutes les façons possibles, les occurrences de symboles non-terminaux annulables par le mot vide.
   - S'il y a n occurrences de symboles non-terminaux annulables dans α, 2^n règles sont créées.
3. Supprimer les productions vides.

$\to$ La grammaire obtenue est équivalente à la grammaire de départ au mot vide près, et est sans production vide.

Algorithme d'élimination des productions vides

Exemple (Algorithme d'élimination des productions vides)

1. Soit $G = (\{S\}, \{a, b\}, S, R)$ avec R :
   - $S \to aSbS \mid \varepsilon$
2. $N_\varepsilon = \{S\}$
3. On remplace S par le mot vide de toutes les façons possibles dans $aSbS$: 4 mots $aSbS$, abS, aSb, ab.
4. Grammaire propre obtenue :
   - $S \to aSbS \mid abS \mid aSb \mid ab$

Algorithme d'élimination des productions vides

Exemple (Algorithme d'élimination des productions vides)

1. Soit $G = (\{S\}, \{a, b\}, S, R)$ avec R :
   - $S \to aSb \mid SS \mid \varepsilon$
2. $N_\varepsilon = \{S\}$
3. On remplace S par le mot vide de toutes les façons possibles :
   - dans aSb : 2 mots aSb, ab
   - dans SS : 2 mots SS, S
4. Grammaire propre obtenue :
   - $S \to aSb \mid ab \mid SS$

Algorithme d'élimination des règles unitaires ⋆⋆

Algorithme d'élimination des règles unitaires

1. Calculer la relation $\geq$ définie par : $X \geq Y \iff X \to^* Y$
2. Calculer la relation $\sim$ définie par : $X \sim Y \iff X \geq Y$ et $Y \geq X$
3. Pour chaque classe d'équivalence :
   1. Choisir un symbole représentant
   2. Remplacer tous les autres symboles équivalents par ce représentant
   3. Éliminer toutes les règles unitaires entre symboles équivalents
4. En commençant par les symboles minimaux dans la relation $\geq$, remplacer les règles unitaires $X \to Y$ par toutes les règles $X \to \alpha$ pour tous les α tels que $Y \to \alpha$

$\to$ La grammaire obtenue est équivalente à la grammaire de départ et est sans règle unitaire.

Algorithme d'élimination des règles unitaires

Exemple (Algorithme d'élimination des règles unitaires)

1. Soit $G = (\{E, T, F\}, \{+, \times, (,), a, b\}, E, R)$ avec R :
   - $E \to E + T \mid T$
   - $T \to T \times F \mid F$
   - $F \to (E) \mid a \mid b$
2. Il y a deux règles unitaires : $E \to T$ et $T \to F$
   $\to$ L'ordre obtenu est $E \geq T \geq F$ (et pas de symboles équivalents)
3. On commence par les symboles minimaux dans la relation $\geq$:
   - On substitue à $T \to F$ les règles $T \to (E) \mid a \mid b$
   - On substitue à $E \to T$ les règles $E \to T \times F \mid (E) \mid a \mid b$
4. On obtient :
   - $E \to E + T \mid T \times F \mid (E) \mid a \mid b$
   - $T \to T \times F \mid (E) \mid a \mid b$
   - $F \to (E) \mid a \mid b$

Plan

Grammaire récursive à gauche

Définition (Grammaire récursive à gauche)

Une grammaire algébrique est **récursive à gauche** si elle contient un symbole $A \in N$ tel qu'il existe une dérivation $A \rightarrow^* A\alpha, \alpha \in V^*$

Remarque

Ce type de grammaire ne permet pas de construire des analyseurs syntaxiques descendants. C'est ce qui motive l'élimination des récursivité à gauche.

Algorithme d'élimination des récursivité à gauche ⋆⋆

Algorithme d'élimination des récursivité à gauche immédiate

Soit une grammaire avec des dérivations à gauche *immédiates* pour le symbole non-terminal A :

- $A \to A\alpha_1 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m$

Alors, on transforme la règle précédente en ajoutant le non-terminal A' :

- $A \to \beta_1 A' \mid \ldots \mid \beta_m A'$
- $A' \to \alpha_1 A' \mid \ldots \mid \alpha_n A' \mid \varepsilon$

Algorithme d'élimination des récursivité à gauche

Exemple (Algorithme d'élimination des récursivité à gauche immédiate)

Soit $G = (\{E, T, F\}, \{+, \times, (,), a, b\}, E, R)$ avec R :

- $E \to E + T \mid T$
- $T \to T \times F \mid F$
- $F \to (E) \mid a \mid b$

On a deux récursivité à gauche immédiates à éliminer. La grammaire devient $G' = (\{E, E', T, T', F\}, \{+, \times, (,), a, b\}, E, R')$ avec R' :

- $E \to TE'$
- $E' \to +TE' \mid \varepsilon$
- $T \to FT'$
- $T' \to \times FT' \mid \varepsilon$
- $F \to (E) \mid a \mid b$

Algorithme d'élimination des récursivité à gauche ⋆⋆

Algorithme d'élimination des récursivité à gauche

1. Numéroter les non-terminaux : $N = \{A_1, \ldots, A_n\}$
2. Pour tous les non-terminaux A_i
   1. Pour toutes les règles $A_i \to A_j\alpha_i$ avec $j < i$
      1. Pour chaque règle $A_j \to \beta_j$, ajouter la règle $A_i \to \beta_j\alpha_i$
      2. Enlever la règle $A_i \to A_j\alpha_i$
   2. Éliminer les récursions à gauche immédiates pour A_i

Algorithme d'élimination des récursivité à gauche

Exemple (Algorithme d'élimination des récursivité à gauche)

Soit $G = (\{A_1, A_2\}, \{a, b, c, d\}, A_1, R)$ avec R :

- $A_1 \to A_2 a \mid b$
- $A_2 \to A_2 c \mid A_1 d$

On déroule l'algorithme :

1. Pour A_1, rien à faire.
2. Pour A_2, la règle $A_2 \to A_1 d$ est concerné, on la remplace par :
   - $A_2 \to A_2 ad \mid bd$

   On élimine la récursivité à gauche immédiate pour A_2 :
   - $A_2 \to bdA_2'$
   - $A_2' \to cA_2' \mid adA_2' \mid \varepsilon$

Plan

Forme normale de Chomsky ⋆⋆

Définition (Forme normale de Chomsky)

Une grammaire algébrique $G = (N, T, S, R)$ est sous **forme normale de Chomsky** (ou forme normale quadratique) si toutes les règles de production sont sous la forme :

$$A \to a$$
$$A \to BC$$

avec $A, B, C \in N$ et $a \in A$

Remarque

L'arbre de dérivation d'une grammaire sous forme normale de Chomsky est un arbre binaire

Forme normale de Chomsky ⋆⋆

Proposition (Forme normale de Chomsky)

Pour tout langage algébrique propre, il existe une grammaire sous forme normale de Chomsky qui l'engendre.

Corollaire (Décidabilité de l'appartenance d'un mot)

Il existe un algorithme qui permet de savoir si un mot appartient à un langage algébrique

Forme normale de Chomsky ⋆⋆

Algorithme de mise en forme normale de Chomsky

Soit $G = (N, T, S, R)$ une grammaire propre.

1. Ajouter $\{Z_a | a \in A\}$ à l'ensemble des symboles non-terminaux
2. Ajouter $Z_a \to a$ à l'ensemble des règles pour tout $a \in A$
3. Conserver toutes les règles de la forme $X \to \alpha$ avec $|\alpha| = 1$
4. Transformer toutes les règles de la forme $X \to \alpha$ avec $|\alpha| \geq 2$:
   1. Remplacer chaque symbole terminal a par Z_a
   2. Si $|\alpha| \geq 3$, c'est-à-dire $\alpha = Y_1 \dots Y_p$, ajouter $p-2$ nouveaux symboles $T_1, \dots, T_{p-2}$ et remplacer la règle $X \to Y_1 \dots Y_p$ par :

$$X \to Y_1 T_1, T_1 \to Y_2 T_2, \dots, T_{p-3} \to Y_{p-2} T_{p-2}, T_{p-2} \to Y_{p-1} Y_p$$

Forme normale de Chomsky

Exemple (Forme normale de Chomsky)

1. Soit $G = (\{S\}, \{a, b\}, S, R)$ avec R :
   - $S \to aSbS \mid abS \mid aSb \mid ab$
2. On ajoute Z_a et Z_b et on transforme les règles :
   - $S \to Z_aSZ_bS \mid Z_aZ_bS \mid Z_aSZ_b \mid Z_aZ_b$
   - $Z_a \to a$, $Z_b \to b$
3. On ajoute des nouveaux symboles T_i et on transforme les règles :
   - $S \to Z_aT_1$, $T_1 \to ST_2$, $T_2 \to Z_bS$
   - $S \to Z_aT_3$, $T_3 \to Z_bS$
   - $S \to Z_aT_4$, $T_4 \to SZ_b$
   - $S \to Z_aZ_b$
   - $Z_a \to a$, $Z_b \to b$
4. La grammaire obtenue est sous forme normale de Chomsky

Forme normale de Greibach

Définition (Forme normale de Greibach)

Une grammaire algébrique $G = (N, T, S, R)$ est sous **forme normale de Greibach** si toutes les règles de production sont sous la forme :

$$A \to a\alpha$$

avec $A \in N, a \in T, \alpha \in N^*$

Forme normale de Greibach ⋆⋆

Proposition (Forme normale de Greibach)

Pour tout langage algébrique propre, il existe une grammaire sous forme normale de Greibach qui l'engendre.

Plan

Intersection avec un langage régulier ⋆⋆

Proposition (Intersection avec un langage régulier)

Soit L_A un langage algébrique et L_R un langage régulier, alors $L_A \cap L_R$ est un langage algébrique

Intersection avec un langage régulier ⋆⋆

Intersection avec un langage régulier.

Soient $G = (N, T, S, R)$ une grammaire sous forme normale de Chomsky engendrant L_A et $\mathcal{A} = (Q, q_0, \delta, Q_F)$ un automate reconnaissant L_R. On définit la grammaire $G' = (N', T, S', R')$ par :

- $N' = \{(q, U, q'), q \in Q, U \in N, q' \in Q\} \cup \{S'\}$
- R' :
  - $(p, A, q) \to (p, B, r)(r, C, q)$ pour $r \in Q$ et $A \to BC$ de R
    On génère des non-terminaux en simulant le fonctionnement de G
  - $(p, A, q) \to (p, a, q)$ pour $A \to a$
    On génère des non-terminaux en simulant le fonctionnement de G
  - $(p, a, q) \to a$ si $\delta(p, a) = q$
    On simule l'automate, la suite d'états est un chemin dans l'automate
  - $S' \to (q_0, S, q_f)$ pour tout $q_f \in Q_F$
    On commence bien dans l'état initial et on finit dans un état final

□

Huitième partie

Automates à pile

Plan de ce cours

Plan

Lemme d'itération

Théorème (Lemme d'itération pour les langages algébriques)

Soit L un langage algébrique. Alors il existe un entier N tel que tout mot w de longueur $|w| \geq N$ possède une factorisation $w = xuyvz$ telle que :

- $|uv| > 0$
- $|uyv| \leq N$
- $xu^n yv^n z \in L, \forall n \geq 0$

Remarque

Intuitivement, il existe un non-terminal A tel que :

$$S \rightarrow^* xAz, A \rightarrow^* uAv, A \rightarrow^* y$$

Langage $a^n b^n c^n$

Proposition (Langage $a^n b^n c^n$)

Le langage $L = \{a^n b^n c^n, n \geq 0\}$ n'est pas algébrique.

Langage $a^n b^n c^n$.

1. Supposons que $L = \{a^n b^n c^n, n \geq 0\}$ soit algébrique.
2. On applique le lemme d'itération pour obtenir la constante N.
3. On considère le mot $w = a^N b^N c^N$. On a bien $|w| = 3N > N$.
4. Comme $|uyv| \leq N$, u et v ne contiennent pas de a ou pas de c.
5. Au moment d'itérer, on n'aura plus autant de a que de b que de c puisqu'une des lettres ne sera pas itérée.
6. Donc L n'est pas algébrique.

□

Plan

Automate à pile simple

Définition (Automate à pile simple)

Un **automate à pile simple** $\mathcal{A}$ est un quadruplet $(\Sigma, \Gamma, Z_0, \Delta)$ avec :

- Σ est l'alphabet d'entrée
- Γ est l'alphabet de la pile
- $Z_0 \in \Gamma$ est le symbole de fond de pile
- Δ est une partie finie de $\Sigma \times \Gamma \times \Gamma^*$

Automate à pile simple

Fonctionnement d'un automate à pile simple

On représente le fonctionnement d'un automate à pile simple à l'aide d'une pile. $(a, Z, Z') \in \Delta$ avec $Z' = Z_1 \dots Z_p$ signifie que, si le symbole courant du mot d'entrée est a, alors on doit :

1. Dépiler Z
   - Si $Z = \varepsilon$, la transition a lieu pour tous les symboles en haut de la pile, la pile reste inchangée
2. Empiler $Z_1, \dots, Z_p$
   - Si $Z' = \varepsilon$, aucun symbole n'est empilé
3. Lire le symbole a

Le mot est reconnu si, à la fin de la lecture du mot, la pile est vide

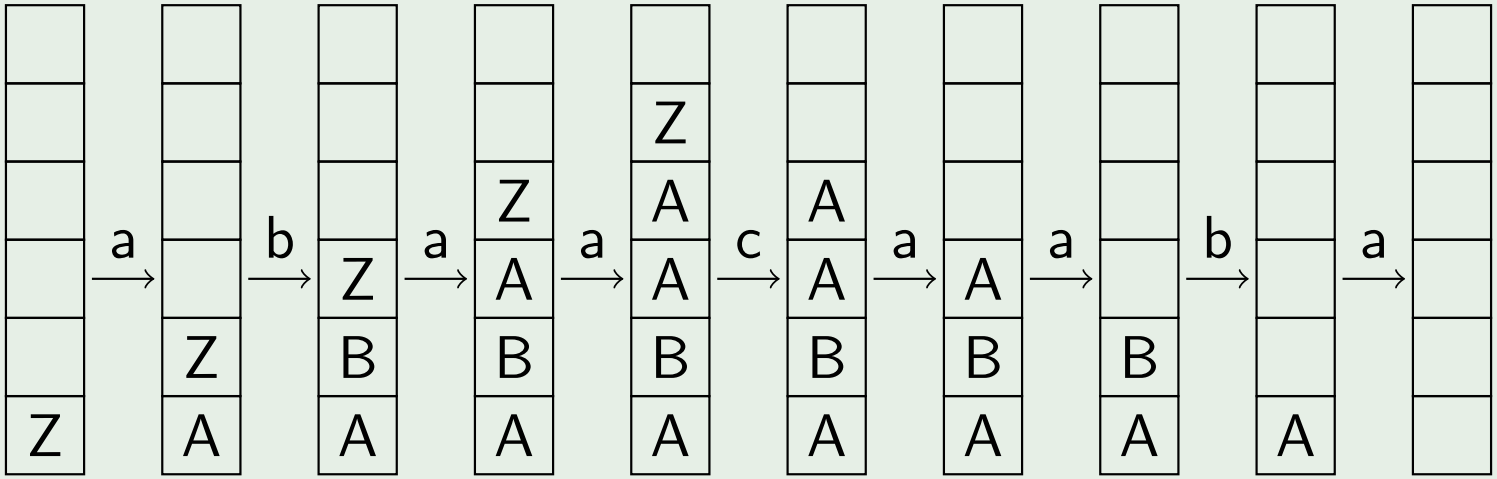
Automate à pile simple

Exemple (Automate à pile simple)

Soit $\mathcal{A} = (\Sigma, \Gamma, Z, \Delta)$ avec :

- $\Sigma = \{a, b, c\}$
- $\Gamma = \{A, B, Z\}$
- $\Delta = \{(a, Z, AZ), (a, A, \varepsilon), (b, Z, BZ), (b, B, \varepsilon), (c, Z, \varepsilon)\}$

On analyse le mot *abaacaaba* :



Plan

Grammaire algébrique et automate à pile simple

Proposition (Grammaire algébrique et automate à pile simple)

Toute grammaire sous forme normale de Greibach peut être transformée en un automate à pile simple

Algorithme de transformation

Soit $G = (N, T, S, R)$ une grammaire algébrique sous forme normale de Greibach. On définit l'automate à pile simple $\mathcal{A} = (\Sigma, \Gamma, Z_0, \Delta)$ avec :

- $\Sigma = T$
- $\Gamma = N$
- $Z_0 = S$
- Δ : à chaque règle $A \to a\alpha$ de R, on associe la transition (a, A, α^R) où α^R est le mot miroir de α

Grammaire algébrique et automate à pile simple

Exemple (Grammaire algébrique et automate à pile simple)

Soit $G = (\{S, A, B, C\}, \{a, b\}, S, R)$ avec R :

- $S \to aACSB \mid aSB \mid bB$
- $A \to aACS \mid aS \mid b$
- $B \to aAC \mid a$
- $C \to a$

G est bien en forme normale de Greibach. L'automate à pile simple correspondant est $\mathcal{A} = (\{a, b\}, \{S, A, B, C\}, S, \Delta)$ avec

$$\Delta = \{(a, S, BSCA), (a, S, BS), (b, S, B),$$
$$(a, A, SCA), (a, A, S), (b, A, \varepsilon),$$
$$(a, B, CA), (a, B, \varepsilon), (a, C, \varepsilon)\}$$

Plan

Automate à pile

Définition (Automate à pile)

Un **automate à pile** $\mathcal{A}$ est un septuplet $(\Sigma, \Gamma, Z_0, Q, q_0, Q_F, \Delta)$ avec :

- Σ est l'alphabet d'entrée
- Γ est l'alphabet de la pile
- $Z_0 \in \Gamma$ est le symbole de fond de pile
- Q est un ensemble fini d'états
- $q_0 \in Q$ est un état initial
- $Q_F \subseteq Q$ est un ensemble d'états finaux
- Δ est une partie finie de $(\Sigma \cup \{\varepsilon\}) \times Q \times \Gamma \times Q \times \Gamma^*$

Automate à pile

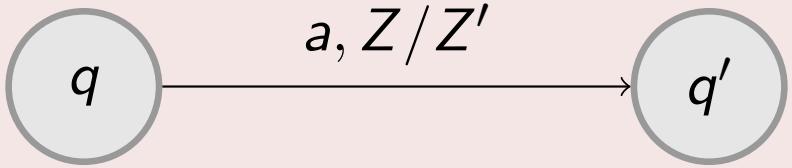
Fonctionnement d'un automate à pile

$(a, q, Z, q', Z') \in \Delta$ avec $Z' = Z_1 \dots Z_p$ signifie que, si l'automate est dans l'état q et que le symbole courant du mot d'entrée est a, alors on doit :

1. Dépiler Z
   - Si $Z = \varepsilon$, la transition a lieu pour tous les symboles en haut de la pile, la pile reste inchangée
2. Empiler $Z_1, \dots, Z_p$
   - Si $Z' = \varepsilon$, aucun symbole n'est empilé
3. Lire le symbole a
4. Transiter dans l'état q'

Automate à pile

Représentation de $(q, a, Z, q', Z') \in \Delta$



Automate à pile

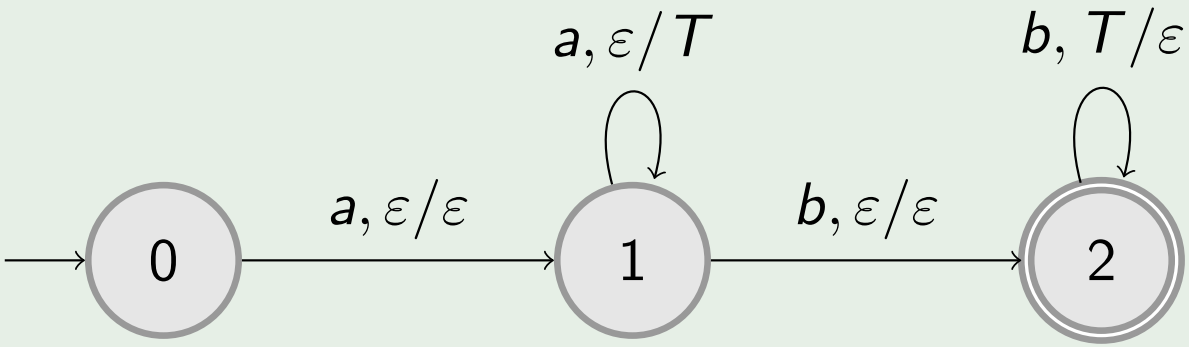
Remarques

- Un automate à pile est une combinaison d'un automate et d'une pile. Par rapport à un automate fini, la fonction de transition prend en compte l'état de la pile et agit sur la pile.
- Un automate fini est un automate à pile avec un alphabet de pile vide ($\Gamma = \varnothing$) et dont toutes les transitions laissent la pile inchangée
- Un automate à pile simple est un automate à pile avec un seul état.

Automate à pile

Exemple (Automate à pile)

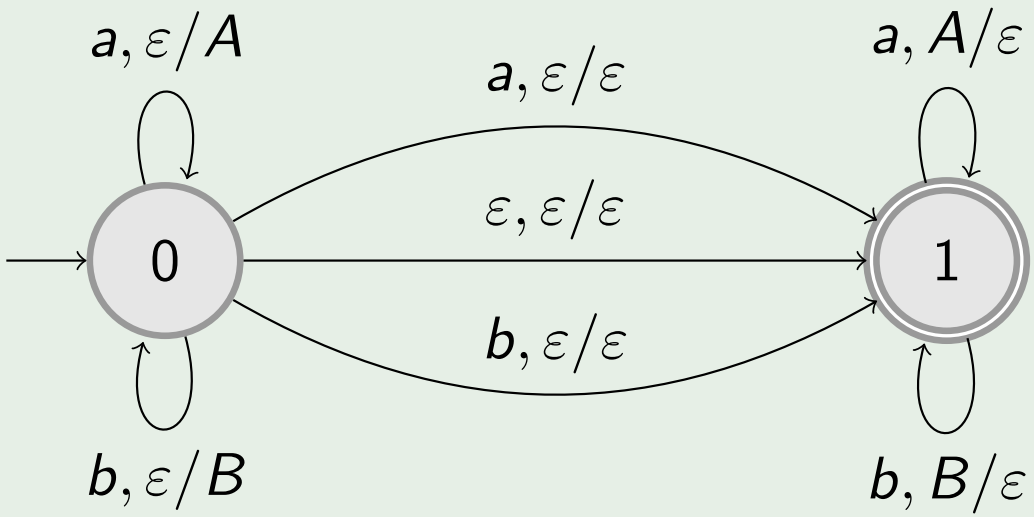
L'automate à pile suivant reconnaît le langage $\{a^n b^n, n \geq 1\}$



Automate à pile

Exemple (Automate à pile)

L'automate à pile suivant reconnaît le langage des palindromes :



Plan

Configuration

Définition (Configuration)

Une **configuration** est un triplet (q, w, h) avec :

- $q \in Q$ est l'état courant de l'automate
- $w \in \Sigma^*$ est le mot restant à lire
- $h \in \Gamma^*$ est le mot sur la pile lu depuis le sommet

Remarque

La configuration initiale de l'automate à pile pour un mot w est (q_0, w, Z_0)

Transition et calcul

Définition (Transition)

Il existe une **transition** entre la configuration (q, aw, zh) et la configuration $(q', w, z'h)$, notée :

$$(q, aw, zh) \vdash (q', w, z'h)$$

s'il existe $(a, q, z, q', z') \in \Delta$

Définition (Calcul)

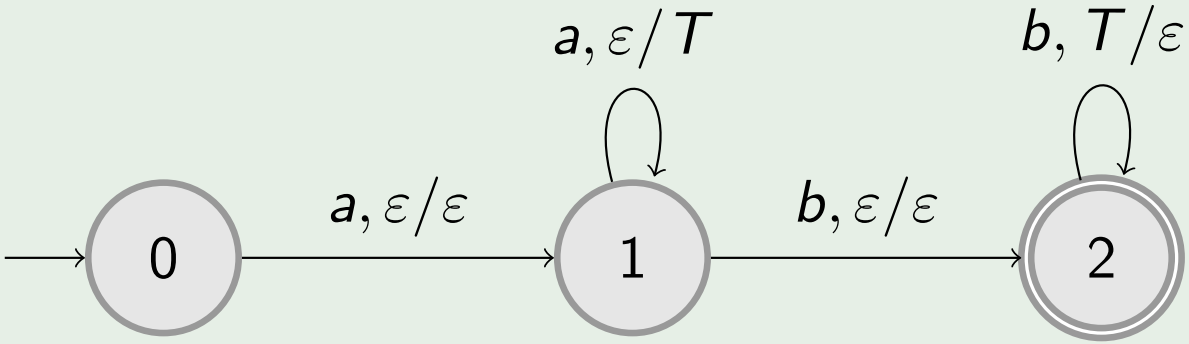
Un **calcul** valide de l'automate à pile sur un mot w est une suite de configurations $c_0, \ldots, c_n$ si :

- $c_0 = (q_0, w, Z_0)$ est la configuration initiale
- $c_0 \vdash c_1 \vdash \ldots \vdash c_n$, noté $c_0 \vdash^* c_n$

Transition et calcul

Exemple (Transition et calcul)

Soit l'automate à pile suivant qui reconnaît le langage $\{a^n b^n, n \geq 1\}$

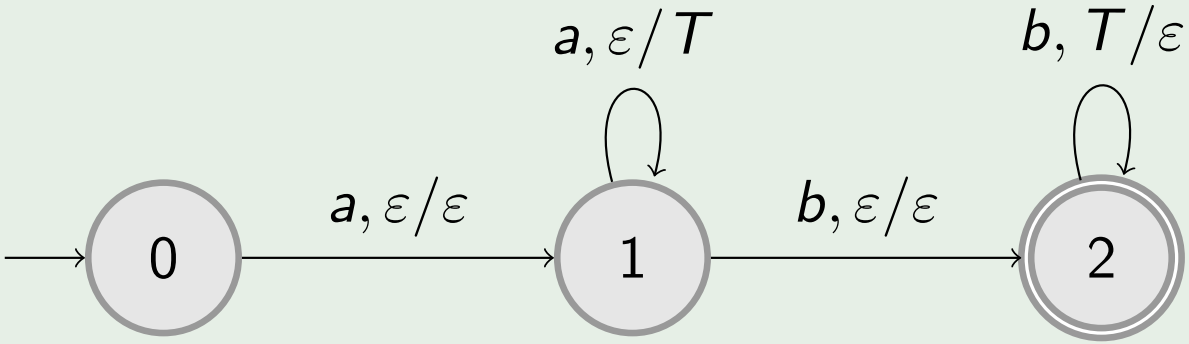


$(0, aabb, Z_0)$

Transition et calcul

Exemple (Transition et calcul)

Soit l'automate à pile suivant qui reconnaît le langage $\{a^n b^n, n \geq 1\}$

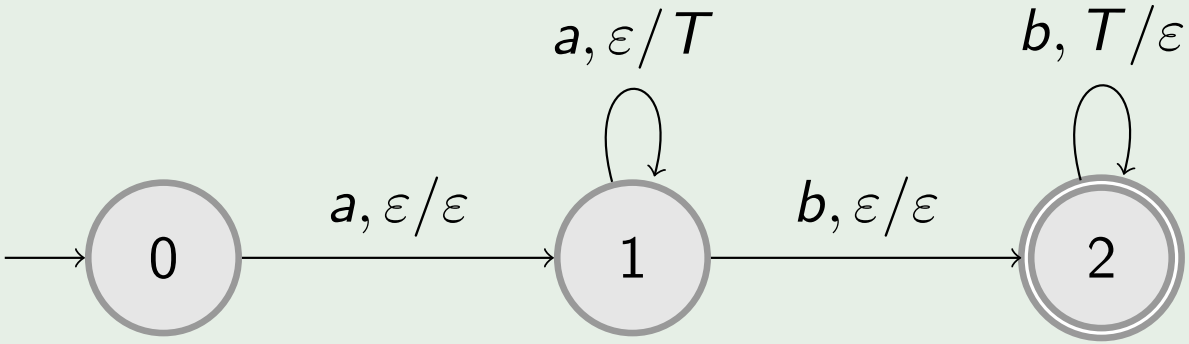


$$(0, aabb, Z_0) \vdash (1, abb, Z_0)$$

Transition et calcul

Exemple (Transition et calcul)

Soit l'automate à pile suivant qui reconnaît le langage $\{a^n b^n, n \geq 1\}$

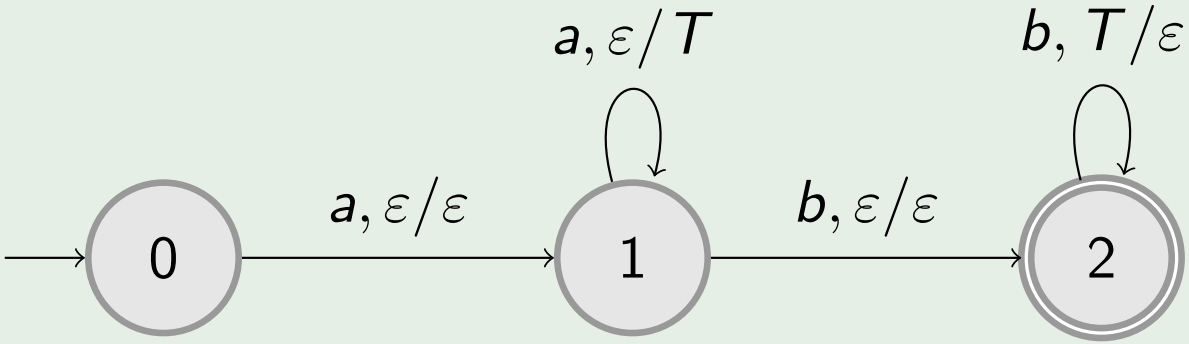


$$(0, aabb, Z_0) \vdash (1, abb, Z_0) \vdash (1, bb, TZ_0)$$

Transition et calcul

Exemple (Transition et calcul)

Soit l'automate à pile suivant qui reconnaît le langage $\{a^n b^n, n \geq 1\}$

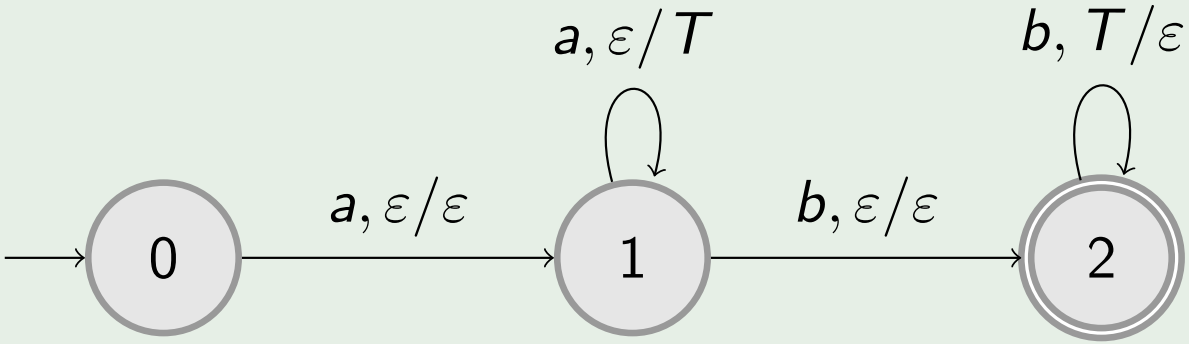


$$(0, aabb, Z_0) \vdash (1, abb, Z_0) \vdash (1, bb, TZ_0)$$
$$\vdash (2, b, TZ_0)$$

Transition et calcul

Exemple (Transition et calcul)

Soit l'automate à pile suivant qui reconnaît le langage $\{a^n b^n, n \geq 1\}$

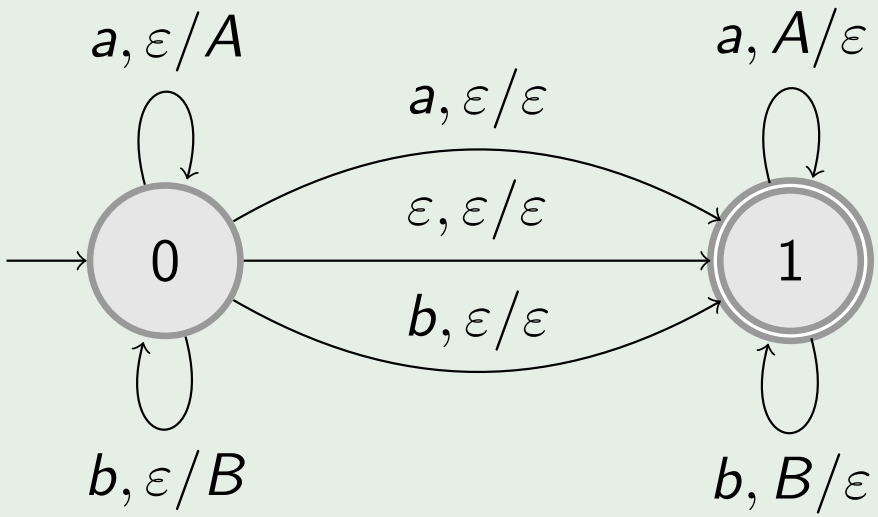


$$(0, aabb, Z_0) \vdash (1, abb, Z_0) \vdash (1, bb, TZ_0)$$
$$\vdash (2, b, TZ_0) \vdash (2, \varepsilon, Z_0)$$

Transition et calcul

Exemple (Transition et calcul)

Soit l'automate à pile suivant qui reconnaît le langage des palindromes :

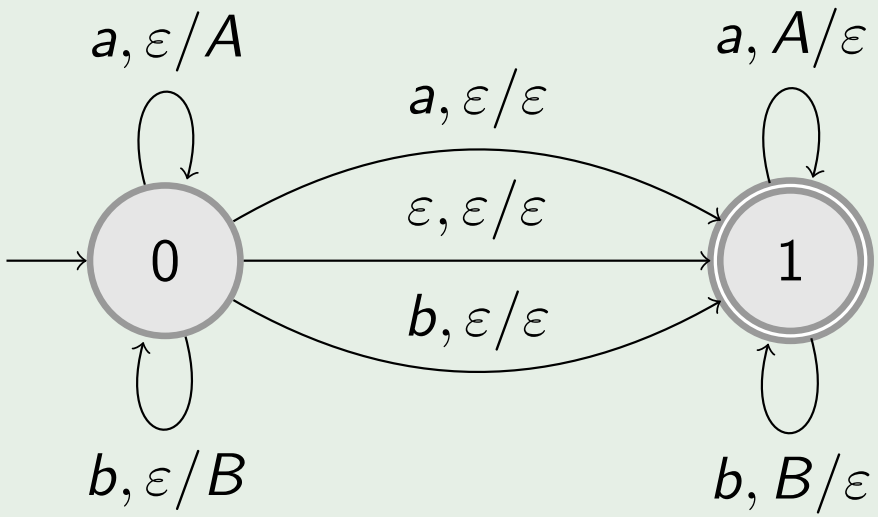


$(0, baaab, Z_0)$

Transition et calcul

Exemple (Transition et calcul)

Soit l'automate à pile suivant qui reconnaît le langage des palindromes :

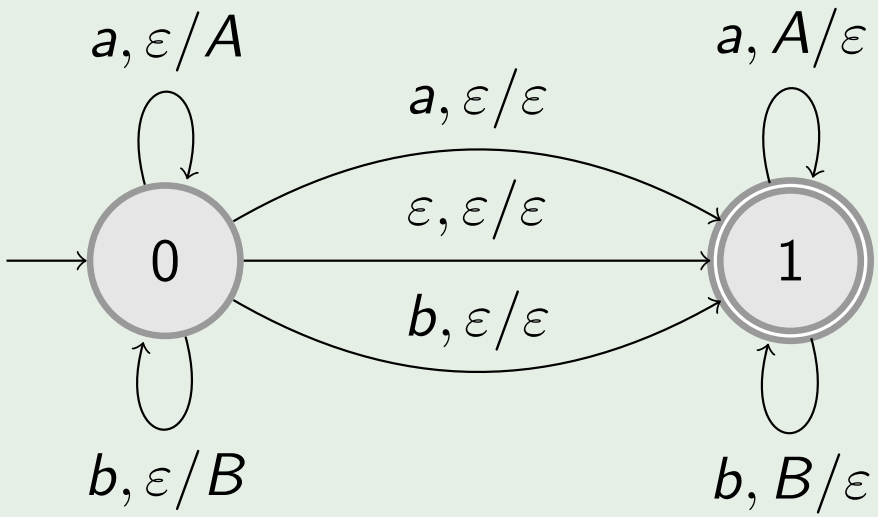


$$(0, baaab, Z_0) \vdash (0, aaab, BZ_0)$$

Transition et calcul

Exemple (Transition et calcul)

Soit l'automate à pile suivant qui reconnaît le langage des palindromes :

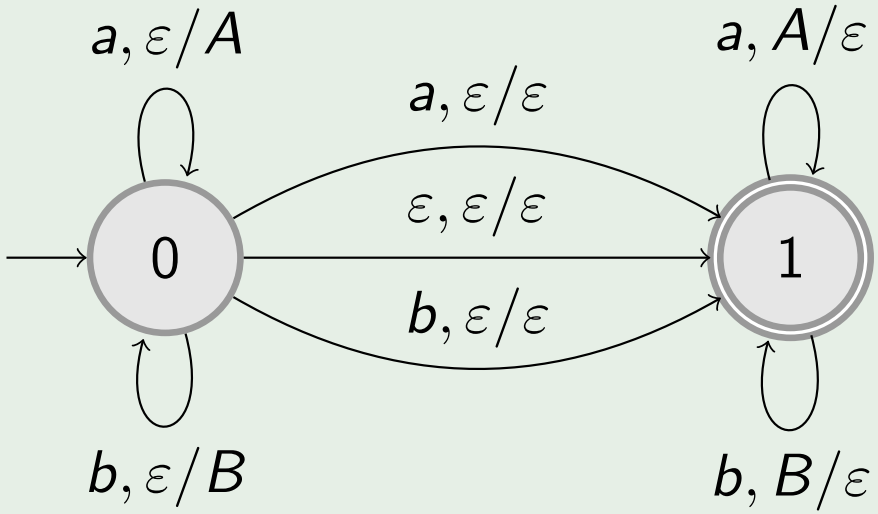


$$(0, baaab, Z_0) \vdash (0, aaab, BZ_0) \vdash (0, aab, ABZ_0)$$

Transition et calcul

Exemple (Transition et calcul)

Soit l'automate à pile suivant qui reconnaît le langage des palindromes :

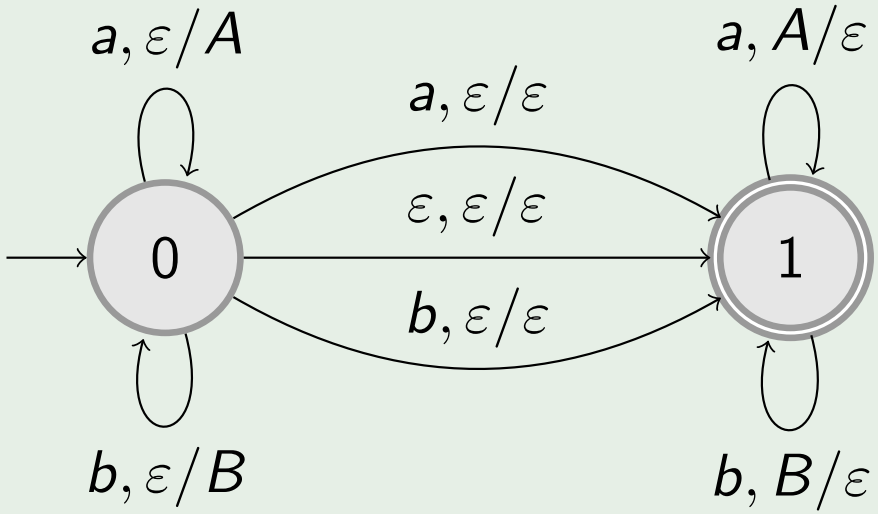


$$(0, baaab, Z_0) \vdash (0, aaab, BZ_0) \vdash (0, aab, ABZ_0)$$
$$\vdash (1, ab, ABZ_0)$$

Transition et calcul

Exemple (Transition et calcul)

Soit l'automate à pile suivant qui reconnaît le langage des palindromes :

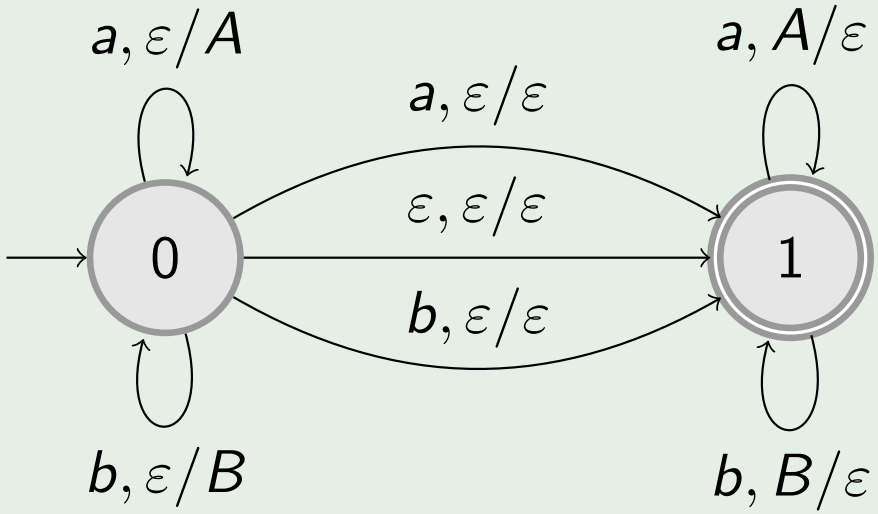


$$(0, baaab, Z_0) \vdash (0, aaab, BZ_0) \vdash (0, aab, ABZ_0)$$
$$\vdash (1, ab, ABZ_0) \vdash (1, b, BZ_0)$$

Transition et calcul

Exemple (Transition et calcul)

Soit l'automate à pile suivant qui reconnaît le langage des palindromes :



$$(0, baaab, Z_0) \vdash (0, aaab, BZ_0) \vdash (0, aab, ABZ_0)$$
$$\vdash (1, ab, ABZ_0) \vdash (1, b, BZ_0) \vdash (1, \varepsilon, Z_0)$$

Plan

Mot accepté par un automate par un automate à pile

Définition (Mot accepté par un automate par un automate à pile)

Un mot w est **accepté** par un automate à pile si :

$$(q_0, w, Z_0) \vdash^* c_F$$

où c_F est une configuration acceptante. Plusieurs cas se présentent :

- acceptation par pile vide
- acceptation par état final
- acceptation par pile vide et état final

Acceptation par pile vide

Définition (Acceptation par pile vide)

Un mot w est **accepté par pile vide** par un automate à pile $(\Sigma, \Gamma, Z_0, Q, q_0, Q_F, \Delta)$ s'il existe un état $q \in Q$ tel que :

$$(q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)$$

Définition (Langage accepté par pile vide)

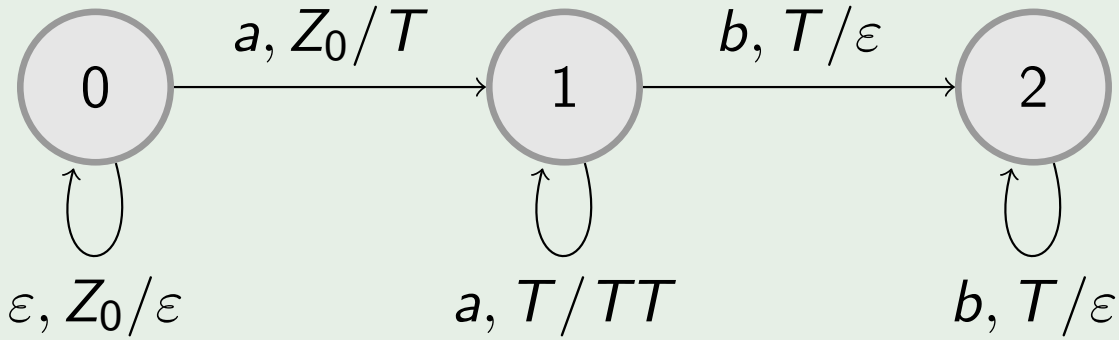
Le **langage accepté par pile vide** par un automate à pile $\mathcal{A}$ est défini par :

$$\mathcal{L}^V(\mathcal{A}) = \{w \in \Sigma^*, (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)\}$$

Acceptation par pile vide

Exemple (Acceptation par pile vide)

L'automate à pile suivant reconnaît le langage $\{a^n b^n, n \geq 0\}$



Acceptation par état final

Définition (Acceptation par état final)

Un mot w est **accepté par état final** par un automate à pile $(\Sigma, \Gamma, Z_0, Q, q_0, Q_F, \Delta)$ s'il existe un état $q_f \in Q_F$ et un mot $z \in \Gamma^*$ tels que :

$$(q_0, w, Z_0) \vdash^* (q_f, \varepsilon, z)$$

Définition (Langage accepté par état final)

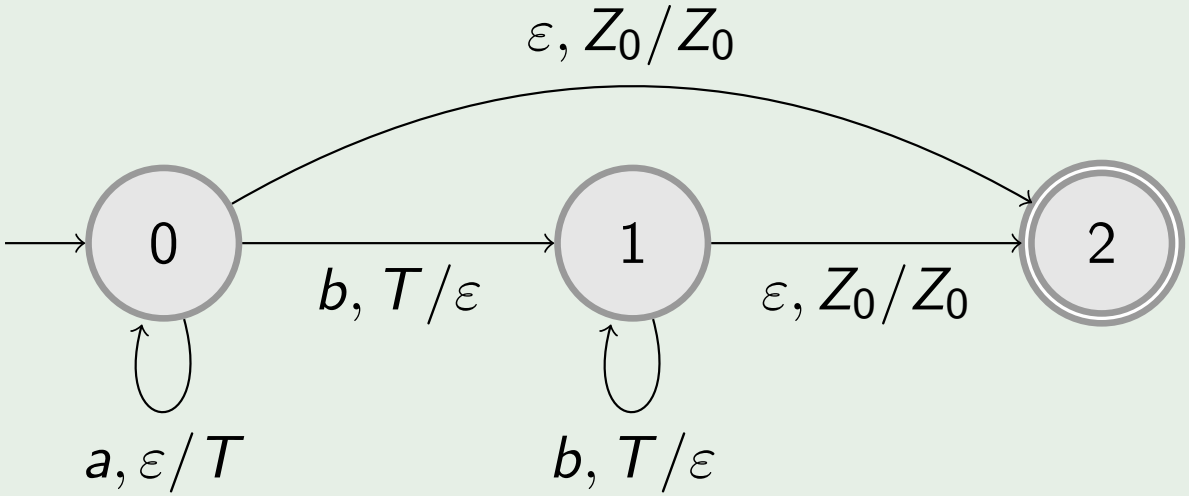
Le **langage accepté par état final** par un automate à pile $\mathcal{A}$ est défini par :

$$\mathcal{L}^F(\mathcal{A}) = \{w \in \Sigma^*, (q_0, w, Z_0) \vdash^* (q_f, \varepsilon, z)\}$$

Acceptation par état final

Exemple (Acceptation par état final)

L'automate à pile suivant reconnaît le langage $\{a^n b^n, n \geq 0\}$



Critères d'acceptation

Proposition (Équivalence des critères d'acceptation)

Les critères d'acceptation par pile vide et par état final sont équivalents, c'est-à-dire que les langages acceptés par pile vide $\mathcal{L}^V$ sont exactement les langages acceptés par état final $\mathcal{L}^F$.

Équivalence des critères d'acceptation.

- $\mathcal{L}^V \subset \mathcal{L}^F$: chaque transition dans laquelle Z_0 est dépilé est remplacée par une transition vers un nouvel état final
- $\mathcal{L}^F \subset \mathcal{L}^V$: après avoir atteint l'état final, on vide entièrement la pile

□

Plan

Automate à pile déterministe

Définition (Automate à pile déterministe)

Un automate à pile $(\Sigma, \Gamma, Z_0, Q, q_0, Q_F, \delta)$ est **déterministe** si, quelle que soit la configuration, une seule transition au plus est applicable. Dit autrement :

- δ est une fonction partielle $(\Sigma \cup \{\varepsilon\}) \times Q \times \Gamma \to Q \times \Gamma^*$
- lorsque $\delta(\varepsilon, q, z)$ est définie, alors $\delta(a, q, z)$ n'est définie pour aucune lettre $a \in \Sigma$

Remarque

Pour les automates à pile déterministes, les modes d'acceptation par pile vide et par état final ne sont plus équivalents !

Langage algébrique déterministe

Définition (Langage algébrique déterministe)

Un **langage algébrique déterministe** est un langage algébrique L pour lequel il existe un automate à pile déterministe $\mathcal{A}$ acceptant par état final tel que :

$$\mathcal{L}^F(\mathcal{A}) = L$$

Remarque importante

Il existe des langages algébriques non-déterministes ! Par exemple, le langage des palindromes est non-déterministe (on ne sait pas deviner où est le milieu du mot).

Langage algébrique et automate à pile

Proposition (Langage algébrique et automate à pile)

Les langages algébrique sont les langages acceptés par un automate à pile

Remarques

- Les automates à pile (non-déterministes) sont nécessaires pour reconnaître les langages algébriques
- Est-il possible de passer d'une grammaire algébrique à un automate à pile et vice-versa ?
  - $\rightarrow$ Oui ! On a déjà fait un sens !

Neuvième partie

Machines de Turing

Plan de ce cours

Plan

Comment reconnaître $a^n b^n c^n$?

Le langage $a^n b^n c^n$

- Le langage $a^n b^n c^n$ n'est pas régulier
  - Il ne peut pas être reconnu par un automate d'états finis
- Le langage $a^n b^n c^n$ n'est pas algébrique
  - Il ne peut pas être reconnu par un automate à pile
- Pourtant, il semble «facile» de reconnaître $a^n b^n c^n$

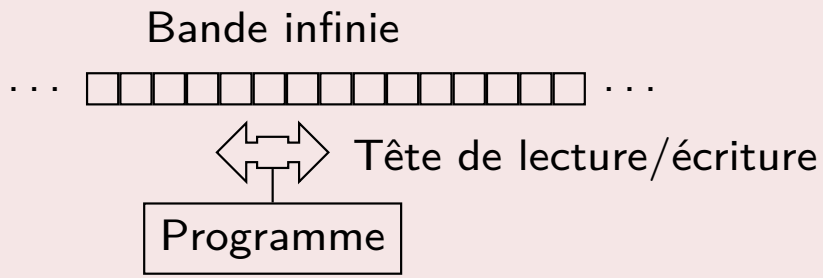
  $\rightarrow$ Machines de Turing !

Machine d'Alan Turing

Machine d'Alan Turing

La machine d'Alan Turing modélise la façon dont un être humain effectue des calculs :

- On peut mémoriser une quantité limitée d'information
  - Nombre fini d'états
- On peut manipuler une quantité infinie d'information sur du papier
  - Bande infinie avec une tête de lecture/écriture



Plan

Machine de Turing

Définition (Machine de Turing)

Une **machine de Turing** $\mathcal{M}$ est un septuplet $(Q, \Gamma, \square, \Sigma, q_0, Q_F, \delta)$ avec :

- Q est un ensemble fini d'états
- Γ est l'alphabet de la bande
- $\square \in \Gamma$ est le symbole blanc (aussi noté B)
- $\Sigma \subset \Gamma \setminus \{\square\}$ est l'alphabet d'entrée
- $q_0 \in Q$ est un état initial
- $Q_F \subseteq Q$ est un ensemble d'états finaux
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\triangleleft, \triangleright\}$ est la fonction de transition

Remarque

On peut aussi définir des états d'acceptations et des états de rejets

Machine de Turing

Fonctionnement d'une machine de Turing

$\delta(q, a) = (q', b, \mathcal{D})$ signifie que, si la machine est dans l'état q et que le symbole sous la tête de lecture/écriture est a, alors on doit :

- Écrire la lettre b
- Transiter dans l'état q'
- Bouger la tête de lecture/écriture dans la direction $\mathcal{D}$ ($\triangleleft$ ou $\triangleright$)

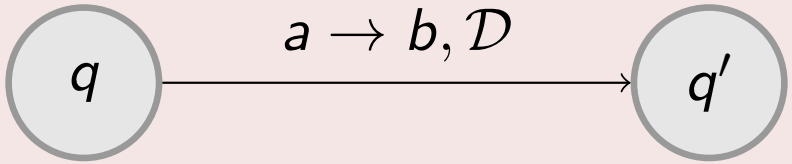
État initial

Au départ :

- Le mot à reconnaître est écrit sur la bande
- La tête de lecture/écriture est placée sur la première lettre
- La bande est complétée par le symbole blanc $\square$

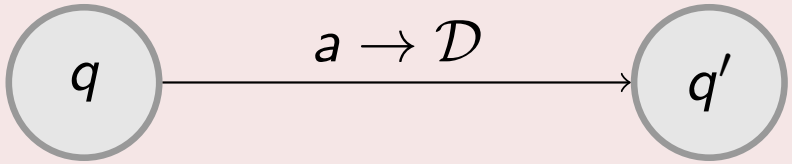
Machine de Turing

Représentation de la fonction de transition

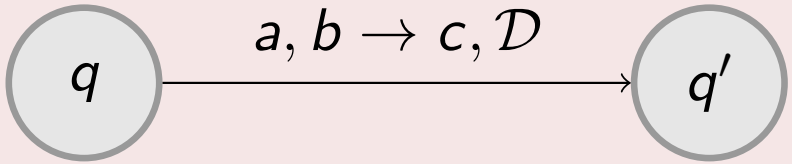


Cas particuliers :

- La machine n'écrit rien



- La machine lit un a ou b



Plan

Configuration

Définition (Configuration)

Une **configuration** est un triplet (u, q, v) avec :

- $u \in \Gamma^*$ est la partie de la bande à gauche de la tête (exclue)
- $q \in Q$ est l'état courant de la machine
- $v \in \Gamma^*$ est la partie de la bande à droite de la tête (inclue)

Remarque

La configuration initiale de la machine pour un mot w est (ε, q_0, w)

Transition

Définition (Transition)

Il existe une **transition** entre la configuration (ua, q, bv) et la configuration (u', q', v'), notée :

$$(ua, q, bv) \vdash (u', q', v')$$

avec $a, b \in \Gamma$, $q, q' \in Q$, $u, v, u', v' \in \Gamma^*$ si :

- $\delta(b, q) = (c, q', \triangleright)$ et alors :
  - $u' = uac$
  - $v' = v$
- $\delta(b, q) = (c, q', \triangleleft)$ et alors :
  - $u' = u$
  - $v' = acv$

Calcul

Définition (Calcul)

Un **calcul** valide de la machine de Turing sur un mot w est une suite de configurations $c_0, \ldots, c_n$ si :

- $c_0 = (\varepsilon, q_0, w)$ est la configuration initiale
- $c_0 \vdash c_1 \vdash \ldots \vdash c_n$, noté $c_0 \vdash^* c_n$

Arrêt d'une machine de Turing

Définition (Arrêt d'une machine de Turing)

Une machine de Turing s'arrête sur une configuration (u, q, av) s'il n'existe pas de transition possible, c'est-à-dire s'il n'existe pas de transition $\delta(q, a)$.

Remarque

De manière équivalente, on peut définir un état de rejet q_R et pour toutes les transitions qui n'existent pas, on définit une transition vers q_R.

Arrêt d'une machine de Turing

Exemple (Arrêt d'une machine de Turing)

Soit $\mathcal{M}$ une machine de Turing avec les règles de transition suivantes :

1. $\delta(q_1, a) = (q_2, a, \triangleright)$
2. $\delta(q_2, b) = (q_1, b, \triangleleft)$

Alors :

- $\mathcal{M}$ s'arrête en partant de la configuration (ε, q_1, aa)
- $\mathcal{M}$ ne s'arrête pas en partant de la configuration (ε, q_1, ab)

Plan

Mot accepté, langage accepté

Définition (Mot accepté)

Un mot w est **accepté** par une machine de Turing si :

$$(\varepsilon, q_0, w) \vdash^* (u, q_f, v), q_f \in Q_F$$

Définition (Langage accepté par une machine de Turing)

Le **langage accepté** par la machine de Turing $\mathcal{M}$ est défini par :

$$\mathcal{L}(\mathcal{M}) = \{w \in \Sigma^*, (\varepsilon, q_0, w) \vdash^* (u, q_f, v), q_f \in Q_F\}$$

Langage récursivement énumérable

Définition (Langage récursivement énumérable)

Un **langage récursivement énumérable** ou **langage semi-décidable** est un langage accepté par une machine de Turing.

Définition (Langage récursivement énumérable)

Un **langage récursivement énumérable** ou **langage semi-décidable** est un langage qui peut être énuméré par une machine de Turing (c'est-à-dire que la machine écrit tous les mots du langage sur le ruban les uns à la suite des autres), éventuellement au bout d'un temps infini.

Remarque

Ces deux définitions sont équivalentes.

Lecture d'un mot

Lecture d'un mot

Quand on lit un mot w avec une machine de Turing, trois cas peuvent se présenter :

1. L'exécution se termine et le mot est accepté
   - La machine est dans un état final
2. L'exécution se termine et le mot est rejeté
   - La machine s'arrête et n'est pas dans un état final
3. L'exécution ne se termine pas
   - La machine ne s'arrête pas et ne passe jamais dans un état final

Langage récursif

Définition (Langage récursif)

Un **langage récursif** ou **langage décidable** est un langage L tel qu'il existe une machine de Turing $\mathcal{M}$ pour laquelle :

- si $w \in L$, $\mathcal{M}$ accepte w
- si $w \notin L$, $\mathcal{M}$ rejette w

$\rightarrow$ $\mathcal{M}$ n'a pas d'exécution infinie

On dit alors que la machine de Turing **décide** L.

Remarque

Un langage décidé par une machine de Turing peut être reconnu par une procédure effective (algorithme).

Langage récursif et langage récursivement énumérable

Langage récursif et langage récursivement énumérable

- Un langage récursif est récursivement énumérable
- Un langage récursivement énumérable n'est pas forcément récursif
- Un langage est récursif si et seulement s'il est récursivement énumérable et que son complémentaire l'est aussi

Plan

$\{w\#w, w \in \{0,1\}^*\}$
Présentation

Exemple (Présentation)

On considère le langage suivant :

$$L = \{w\#w, w \in \{0,1\}^*\}$$

Ce langage est-il :

- régulier ? Non.
- algébrique ? Non.

$\rightarrow$ Nécessité d'une machine de Turing

$\{w\#w, w \in \{0,1\}^*\}$
Stratégie

Exemple (Stratégie)

- Faire des zig-zag entre les places correspondantes des deux côtés du symbole # et déterminer si elles sont identiques
- Placer les marques sur la bande pour garder une trace à propos des places correspondantes identiques

$\rightarrow$ Chaque passe identifie un symbole identique de chaque côté du #. Si tous les symboles sont marqués, le mot est dans L. Si on découvre une incohérence, le mot est rejeté.

$\{w\#w, w \in \{0,1\}^*\}$
Définition de la machine

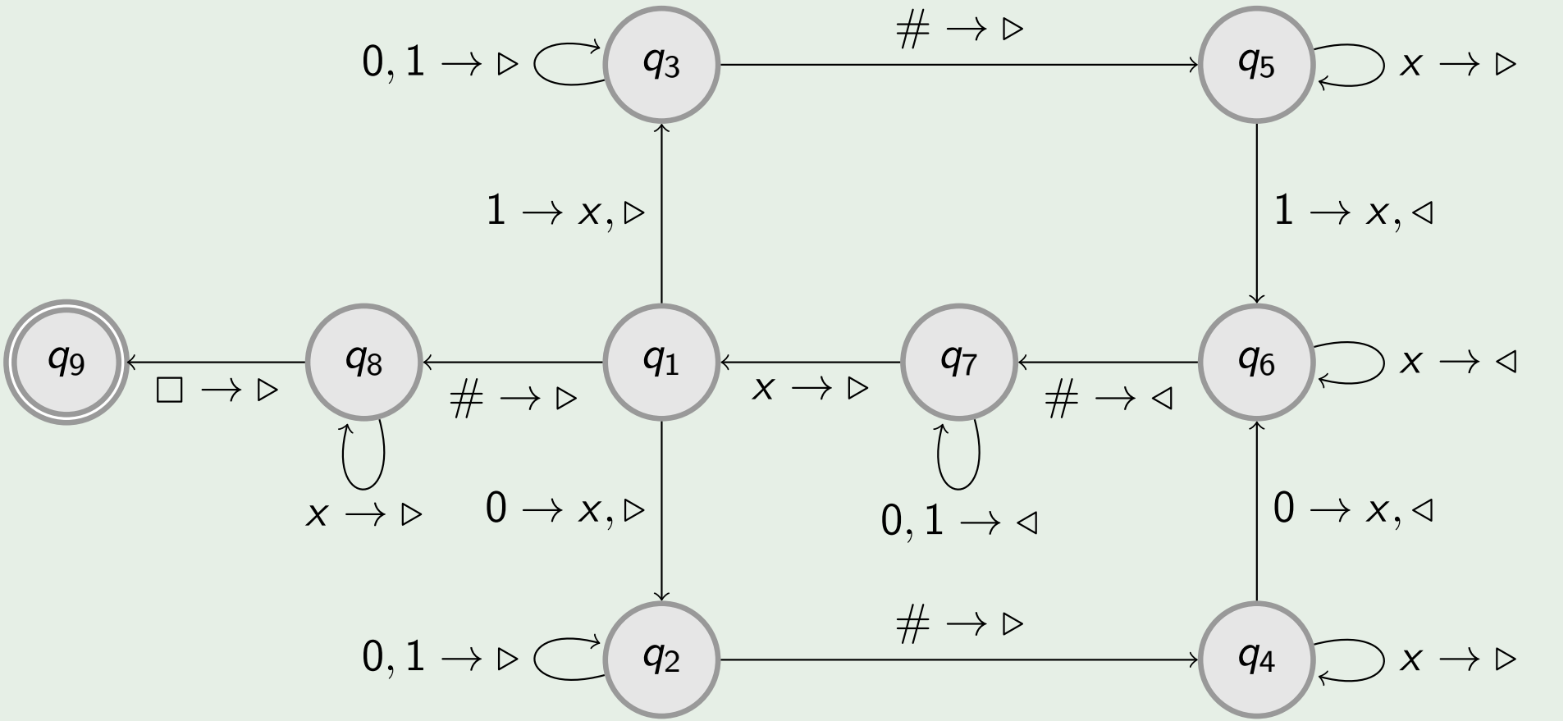
Exemple (Définition de la machine)

On considère la machine $\mathcal{M} = (Q, \Gamma, \square, \Sigma, q_1, Q_F, \delta)$ avec :

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9\}$
- $\Sigma = \{0, 1, \#\}$
- $\Gamma = \Sigma \cup \{\square, x\}$
- $Q_F = \{q_9\}$
- δ donné par le diagramme suivant.

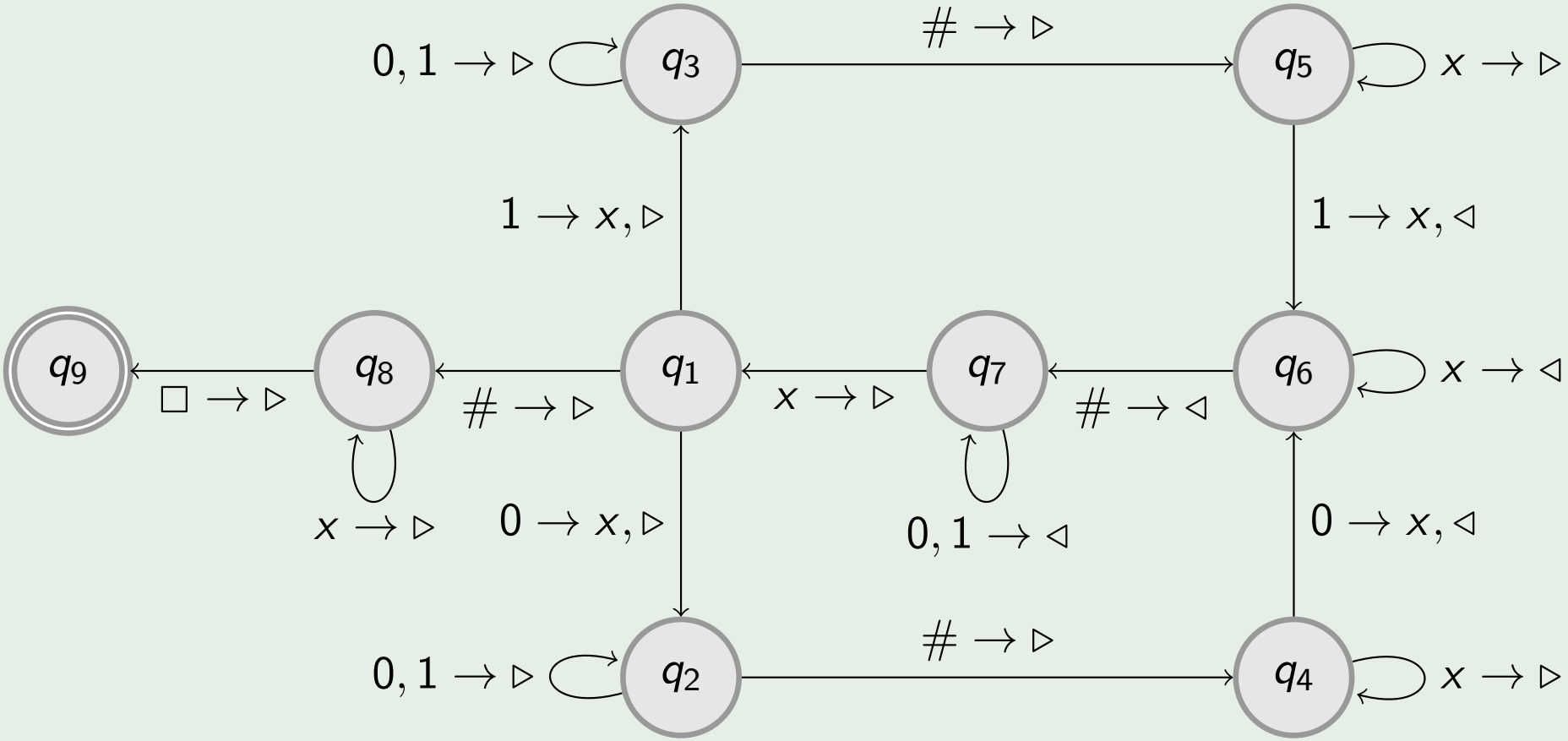
$\{w\#w, w \in \{0,1\}^*\}$

Exemple (Programme)



$\{w\#w, w \in \{0,1\}^*\}$

Exemple (Programme)



Exemple : 0110#0110

Plan

Machine de Turing non-déterministe

Machine de Turing non-déterministe

Dans une machine de Turing non-déterministe, la fonction de transition δ est remplacée par une relation de transition Δ :

$$\Delta \subset Q \times (\Gamma \cup \{\varepsilon\}) \times Q \times \Gamma \times \{\triangleleft, \triangleright\}$$

On peut avoir des ε-transitions.

Question

Est-ce que les machines de Turing non-déterministes reconnaissent les mêmes langages que les machines de Turing déterministes ?

Machine de Turing non-déterministe

Proposition (Machine de Turing non-déterministe)

Pour toute machine de Turing non-déterministe $\mathcal{M}_N$, il existe une machine de Turing déterministe $\mathcal{M}_D$ telle que :

$$\mathcal{L}(\mathcal{M}_N) = \mathcal{L}(\mathcal{M}_D)$$

Idée de la preuve

On construit une machine $\mathcal{M}_D$ qui va parcourir les branches de l'arbre des configuration de $\mathcal{M}_N$. Quand on tombe sur une configuration d'acceptation, on accepte.

C'est tout pour le moment...

Des questions ?