

# Analyse syntaxique

## Langages

## Grammaires

- Arbres de dérivation

- Types de grammaires

## Reconnaisseurs

- Généralités

- Automates finis

- Automates à pile

- Machines de Turing

# Le paysage syntaxique

- Les **symboles** sont des éléments indivisibles qui vont servir de briques de base pour construire des mots.
- Un **alphabet** est un ensemble fini de symboles. On désigne conventionnellement un alphabet par la lettre grecque  $\Sigma$ .
- Une suite de symboles, appartenant à un alphabet  $\Sigma$ , mis bout à bout est appelé un **mot** (ou une *chaîne*) sur  $\Sigma$ . Le mot de longueur zéro est noté  $\epsilon$ .
- On note  $|m|$  la longueur du mot  $m$  (le nombre de symboles qui le composent) et  $|m|_s$  le nombre de symboles  $s$  que possède le mot  $m$ .
- L'ensemble de tous les mots que l'on peut construire sur un alphabet  $\Sigma$  est noté  $\Sigma^*$ .
- Un **langage** sur un alphabet  $\Sigma$  est un ensemble de mots construits sur  $\Sigma$ . Tout langage défini sur  $\Sigma$  est donc une partie de  $\Sigma^*$ .

# Exemples de langages

$$\Sigma = \{a\} \quad L_1 = \{\varepsilon, a, aa, aaa, \dots\}$$

$$\Sigma = \{a, b\} \quad L_2 = \{\varepsilon, ab, aabb, aaabbb, aaaabbbb, \dots\}$$

$$\Sigma = \{a, b\} \quad L_3 = \{\varepsilon, aa, bb, aaaa, abba, baab, bbbb, \dots\}$$

$$\Sigma = \{a, b, c\} \quad L_4 = \{\varepsilon, abc, aabbcc, aaabbbccc, \dots\}$$

# Opérations sur les langages

Union	$L_1 \cup L_2$	$\{x \mid x \in L_1 \text{ ou } x \in L_2\}$
Intersection	$L_1 \cap L_2$	$\{x \mid x \in L_1 \text{ et } x \in L_2\}$
Différence	$L_1 - L_2$	$\{x \mid x \in L_1 \text{ et } x \notin L_2\}$
Complément	$\bar{L}$	$\{x \in \Sigma^* \mid x \notin L\}$
Concaténation	$L_1 L_2$	$\{xy \mid x \in L_1 \text{ et } y \in L_2\}$
Auto concaténation	$\overbrace{L \dots L}^n$	$L^n$
Fermeture de Kleene	$L^*$	$\bigcup_{k \geq 0} L^k$

# Comment décrire un langage ?

- Enumération

$$L_2 = \{\varepsilon, ab, aabb, aaabbb, aaaabbbb, \dots\}$$

- Description littéraire

*Ensemble des mots construits sur l'alphabet  $\{a, b\}$ , commençant par des  $a$  et se terminant par des  $b$  et tel que le nombre de  $a$  et le nombre de  $b$  soit égal*

- Grammaire de réécriture

$$G = \langle \{S\}, \{a, b\}, \{S \rightarrow aSb \mid \varepsilon\}, S \rangle$$

# Grammaires de réécriture

Une grammaire de réécriture est un 4-uplet  $\langle N, \Sigma, P, S \rangle$  où :

- $N$  est un ensemble de **symboles non terminaux**, appelé l'**alphabet non-terminal**.
- $\Sigma$  est un ensemble de **symboles terminaux**, appelé l'**alphabet terminal**, tel que  $N$  et  $\Sigma$  soient disjoints.
- $P$  est un sous ensemble **fini** de :

$$(N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$$

un élément  $(\alpha, \beta)$  de  $P$ , que l'on note  $\alpha \rightarrow \beta$  est appelé une **règle de production** ou **règle de réécriture**.

$\alpha$  est appelé partie gauche de la règle

$\beta$  est appelé partie droite de la règle

- $S$  est un élément de  $N$  appelé l'**axiome** de la grammaire.

# Notation

Pour alléger les notations, on note :

$$\alpha \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

les  $n$  règles :

$$\alpha \rightarrow \beta_1, \alpha \rightarrow \beta_2, \dots, \alpha \rightarrow \beta_n$$



# Proto-phrases d'une grammaire

Les **proto-phrases** d'une grammaire  $G = \langle N, \Sigma, P, S \rangle$  sont des mots construits sur l'alphabet  $\Sigma \cup N$ , on les définit récursivement de la façon suivante :

- $S$  est une proto-phrase de  $G$
- si  $\alpha\beta\gamma$  est une proto-phrase de  $G$  et  $\beta \rightarrow \delta \in P$  alors  $\alpha\delta\gamma$  est une proto-phrase de  $G$ .

Une proto-phrase de  $G$  ne contenant aucun symbole non-terminal est appelé un mot engendré par  $G$ . Le **langage engendré par  $G$** , noté  $L(G)$  est l'ensemble des mots engendrés par  $G$ .

# Dérivation

- L'opération qui consiste à générer une proto-phrased  $\alpha\delta\gamma$  à partir d'une proto-phrased  $\alpha\beta\gamma$  et d'une règle de production  $r$  de la forme  $\beta \rightarrow \delta$  est appelée l'opération de **dérivation**. Elle se note à l'aide d'une double flèche :

$$\alpha\beta\gamma \Rightarrow \alpha\delta\gamma$$

- On note  $\alpha \xRightarrow{k} \beta$  pour indiquer que  $\beta$  se dérive de  $\alpha$  en  $k$  étapes.
- On définit aussi les deux notations  $\xRightarrow{+}$  et  $\xRightarrow{*}$  de la façon suivante :
  - $\alpha \xRightarrow{+} \beta \equiv \alpha \xRightarrow{k} \beta$  avec  $k > 0$
  - $\alpha \xRightarrow{*} \beta \equiv \alpha \xRightarrow{k} \beta$  avec  $k \geq 0$

# Langage engendré par une grammaire

- $L(G)$  est défini de la façon suivante :

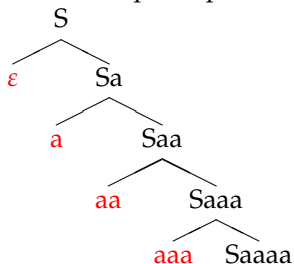
$$L(G) = \{m \in \Sigma^* \mid S \xRightarrow{+} m\}$$

- Deux grammaires  $G$  et  $G'$  sont équivalentes si  $L(G) = L(G')$ .

$$L_1 = \{\varepsilon, a, aa, aaa, \dots\}$$

$$G = \langle \{S\}, \{a\}, \{S \rightarrow Sa \mid \varepsilon\}, S \rangle$$

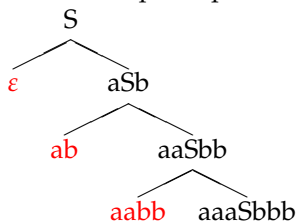
sous-ensemble des proto-phrases de  $G$



$$L_2 = \{\varepsilon, ab, aabb, aaabbb, aaaabbbb, \dots\}$$

$$G = \langle \{S\}, \{a, b\}, \{S \rightarrow aSb \mid \varepsilon\}, S \rangle$$

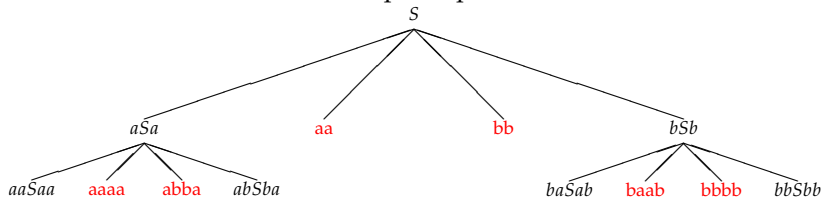
sous-ensemble des proto-phrases de  $G$



$$L_3 = \{aa, bb, aaaa, abba, baab, bbbb, \dots\}$$

$$G = \langle \{S\}, \{a, b\}, \{S \rightarrow aSa \mid bSb \mid aa \mid bb\}, S \rangle$$

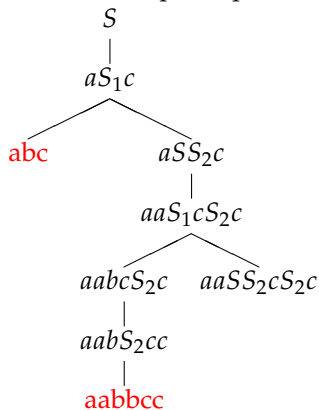
sous-ensemble des proto-phrases de  $G$



$$L_4 = \{\varepsilon, abc, aabbcc, aaabbbccc, \dots\}$$

$$G = \langle \{S, S_1, S_2\}, \{a, b, c\}, \left\{ \begin{array}{ll} S \rightarrow aS_1c, & S_1 \rightarrow b \mid SS_2, \\ cS_2 \rightarrow S_2c, & bS_2 \rightarrow bb \end{array} \right\}, S \rangle$$

sous-ensemble des proto-phrases de G



# Sens de dérivation

$$G = \langle \{E, T, F\}, \{+, *, a\}, \{E \rightarrow T + E \mid T, T \rightarrow F * T \mid F, F \rightarrow a\}, E \rangle$$

Les proto-phrases engendrées lors d'une dérivation peuvent comporter plus d'un symbole non-terminal :

$$\begin{aligned} E &\Rightarrow T + E \\ &\Rightarrow T + T \\ &\Rightarrow F + T \\ &\Rightarrow F + F * T \\ &\Rightarrow F + a * T \\ &\Rightarrow F + a * F \\ &\Rightarrow a + a * F \\ &\Rightarrow a + a * a \end{aligned}$$



# Sens de dérivation

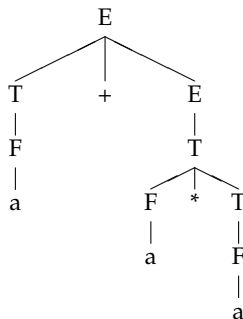
**Dérivation gauche** : on réécrit le non-terminal le plus à gauche :

$$\begin{aligned} E &\Rightarrow T + E \\ &\Rightarrow F + E \\ &\Rightarrow a + E \\ &\Rightarrow a + T \\ &\Rightarrow a + F * T \\ &\Rightarrow a + a * T \\ &\Rightarrow a + a * F \\ &\Rightarrow a + a * a \end{aligned}$$

**Dérivation droite** :  
...

$$\begin{aligned} E &\Rightarrow T + E \\ &\Rightarrow T + T \\ &\Rightarrow T + F * T \\ &\Rightarrow T + F * F \\ &\Rightarrow T + F * a \\ &\Rightarrow T + a * a \\ &\Rightarrow F + a * a \\ &\Rightarrow a + a * a \end{aligned}$$

# Arbre de dérivation



Un arbre de dérivation pour  $G$  ( $G = \langle N, \Sigma, P, S \rangle$ ) est un arbre ordonné et étiqueté dont les étiquettes appartiennent à l'ensemble  $N \cup \Sigma \cup \{\varepsilon\}$ . Si un nœud de l'arbre est étiqueté par le non terminal  $A$  et ses fils sont étiquetés  $X_1, X_2, \dots, X_n$  alors la règle  $A \rightarrow X_1, X_2, \dots, X_n$  appartient à  $P$ .

# Arbre de dérivation

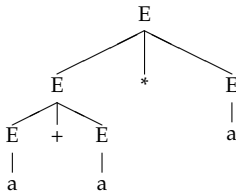
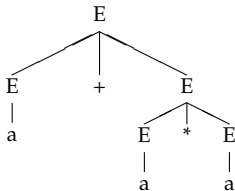
- Un arbre de dérivation indique les règles qui ont été utilisées dans une dérivation, mais pas l'ordre dans lequel elles ont été utilisées.
- À un arbre de dérivation correspondent une seule dérivation droite et une seule dérivation gauche.

# Ambiguïté

Une grammaire  $G$  est **ambiguë** s'il existe au moins un mot  $m$  dans  $L(G)$  auquel correspond plus d'un arbre de dérivation.

*Exemple :*

$$E \rightarrow E + E \mid E * E \mid a$$



# Types de règles

Les grammaires peuvent être classées en fonction de la forme de leurs règles de production. On définit cinq types de règles de production :

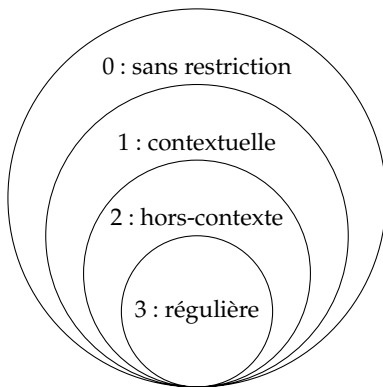
- Une règle est **régulière à gauche** si et seulement si elle est de la forme  $A \rightarrow xB$  ou  $A \rightarrow x$  avec  $A, B \in N$  et  $x \in \Sigma$ .
- Une règle est **régulière à droite** si et seulement si elle est de la forme  $A \rightarrow Bx$  ou  $A \rightarrow x$  avec  $A, B \in N$  et  $x \in \Sigma$ .
- Une règle  $A \rightarrow \alpha$  est un règle **hors-contexte** si et seulement si :  $A \in N$  et  $\alpha \in (N \cup \Sigma)^*$
- Une règle  $\alpha \rightarrow \beta$  est une règle **contextuelle** si et seulement si :  $\alpha = gAd$  et  $\beta = gBd$  avec  $g, d, B \in (N \cup \Sigma)^*$  et  $A \in N$ .  
*Le nom "contextuelle" provient du fait que  $A$  se réécrit  $B$  uniquement dans le contexte  $g_d$ .*
- Une règle  $\alpha \rightarrow \beta$  est une règle **sans restriction** si elle n'est pas contextuelle.

# Type d'une grammaire

Une grammaire est :

- **régulière** ou de type 3 si elle est régulière à droite ou régulière à gauche. Une grammaire est régulière à gauche si **toutes** ses règles sont régulières à gauche et une grammaire est régulière à droite si **toutes** ses règles sont régulières à droite.
- **hors contexte** ou de type 2 si toutes ses règles de production sont hors contexte.
- **dépendante du contexte** ou de type 1 si toutes ses règles de production sont dépendantes du contexte.
- **sans restrictions** ou de type 0 si toutes ses règles de production sont sans restrictions.

# Hiérarchie de Chomsky



# Type d'un langage

Un langage pouvant être engendré par une grammaire de type  $x$  et pas par une grammaire d'un type supérieur dans la hiérarchie, est appelé un **langage de type  $x$** .

Type	Nom
3	régulier
2	hors contexte
1	dépendant du contexte
0	rékursivement énumérable



# Exemples de langages réguliers

$$L = \{m \in \{a, b\}^*\}$$

$$L = \{m \in \{a, b\}^* \mid |m|_a \bmod 2 = 0\}$$

$$L = \{m \in \{a, b\}^* \mid m = xaaa \text{ avec } x \in \{a, b\}^*\}$$

$$L = \{m \in \{a, b\}^* \mid |m|_a \bmod 2 = 0 \text{ et } |m|_b \bmod 2 = 0\}$$

# Exemples de langages réguliers

$$\begin{aligned} L &= \{m \in \{a, b\}^*\} \\ G &= \langle \{S\}, \{a, b\}, \{S \rightarrow aS \mid bS \mid \varepsilon\}, S \rangle \end{aligned}$$

$$L = \{m \in \{a, b\}^* \mid |m|_a \bmod 2 = 0\}$$

$$L = \{m \in \{a, b\}^* \mid m = xaaa \text{ avec } x \in \{a, b\}^*\}$$

$$L = \{m \in \{a, b\}^* \mid |m|_a \bmod 2 = 0 \text{ et } |m|_b \bmod 2 = 0\}$$

# Exemples de langages réguliers

$$\begin{aligned} L &= \{m \in \{a, b\}^*\} \\ G &= \langle \{S\}, \{a, b\}, \{S \rightarrow aS \mid bS \mid \varepsilon\}, S \rangle \end{aligned}$$

$$\begin{aligned} L &= \{m \in \{a, b\}^* \mid |m|_a \bmod 2 = 0\} \\ G &= \langle \{S, T\}, \{a, b\}, \left\{ \begin{array}{l} S \rightarrow aT \mid bS \mid \varepsilon, \\ T \rightarrow aS \mid bT \end{array} \right\}, S \rangle \\ L &= \{m \in \{a, b\}^* \mid m = xaaa \text{ avec } x \in \{a, b\}^*\} \end{aligned}$$

$$L = \{m \in \{a, b\}^* \mid |m|_a \bmod 2 = 0 \text{ et } |m|_b \bmod 2 = 0\}$$

# Exemples de langages réguliers

$$\begin{aligned} L &= \{m \in \{a, b\}^*\} \\ G &= \langle \{S\}, \{a, b\}, \{S \rightarrow aS \mid bS \mid \varepsilon\}, S \rangle \end{aligned}$$

$$\begin{aligned} L &= \{m \in \{a, b\}^* \mid |m|_a \bmod 2 = 0\} \\ G &= \langle \{S, T\}, \{a, b\}, \left\{ \begin{array}{l} S \rightarrow aT \mid bS \mid \varepsilon, \\ T \rightarrow aS \mid bT \end{array} \right\}, S \rangle \end{aligned}$$

$$\begin{aligned} L &= \{m \in \{a, b\}^* \mid m = xaaa \text{ avec } x \in \{a, b\}^*\} \\ G &= \langle \{S, T, U\}, \{a, b\}, \left\{ \begin{array}{l} S \rightarrow aS \mid bS \mid aT, \\ T \rightarrow aU, \\ U \rightarrow a \end{array} \right\}, S \rangle \end{aligned}$$

$$L = \{m \in \{a, b\}^* \mid |m|_a \bmod 2 = 0 \text{ et } |m|_b \bmod 2 = 0\}$$

# Exemples de langages réguliers

$$L = \{m \in \{a, b\}^*\}$$

$$G = \langle \{S\}, \{a, b\}, \{S \rightarrow aS \mid bS \mid \varepsilon\}, S \rangle$$

$$L = \{m \in \{a, b\}^* \mid |m|_a \bmod 2 = 0\}$$

$$G = \langle \{S, T\}, \{a, b\}, \left\{ \begin{array}{l} S \rightarrow aT \mid bS \mid \varepsilon, \\ T \rightarrow aS \mid bT \end{array} \right\}, S \rangle$$

$$L = \{m \in \{a, b\}^* \mid m = xaaa \text{ avec } x \in \{a, b\}^*\}$$

$$G = \langle \{S, T, U\}, \{a, b\}, \left\{ \begin{array}{l} S \rightarrow aS \mid bS \mid aT, \\ T \rightarrow aU, \\ U \rightarrow a \end{array} \right\}, S \rangle$$

$$L = \{m \in \{a, b\}^* \mid |m|_a \bmod 2 = 0 \text{ et } |m|_b \bmod 2 = 0\}$$

$$G = \langle \{S, T, U, V\}, \{a, b\}, \left\{ \begin{array}{l} S \rightarrow aT \mid bU, \quad T \rightarrow aS \mid bV, \\ V \rightarrow aU \mid bT, \quad U \rightarrow aV \mid bS \mid \varepsilon \end{array} \right\}, S \rangle$$

# Exemples de langages hors-contexte

$$L = \{a^n b^n \mid n \geq 0\}$$

$$L = \{mm^{-1} \mid m \in \{a, b\}^*\} \quad (\text{langage miroir})$$

# Exemples de langages hors-contexte

$$L = \{a^n b^n \mid n \geq 0\}$$

$$G = \langle \{S\}, \{a, b\}, \{S \rightarrow aSb \mid \varepsilon\}, S \rangle$$

$$L = \{mm^{-1} \mid m \in \{a, b\}^*\} \quad (\text{langage miroir})$$

# Exemples de langages hors-contexte

$$L = \{a^n b^n \mid n \geq 0\}$$

$$G = \langle \{S\}, \{a, b\}, \{S \rightarrow aSb \mid \varepsilon\}, S \rangle$$

$$L = \{mm^{-1} \mid m \in \{a, b\}^*\} \quad (\text{langage miroir})$$

$$G = \langle \{S\}, \{a, b\}, \{S \rightarrow aSa \mid bSb \mid aa \mid bb\}, S \rangle$$



# Exemples de langages contextuels

$$L = \{a^n b^n c^n \mid n \geq 0\}$$

$$L = \{m\#m \mid \text{avec } m \in \{a,b\}^*\}$$

# Exemples de langages contextuels

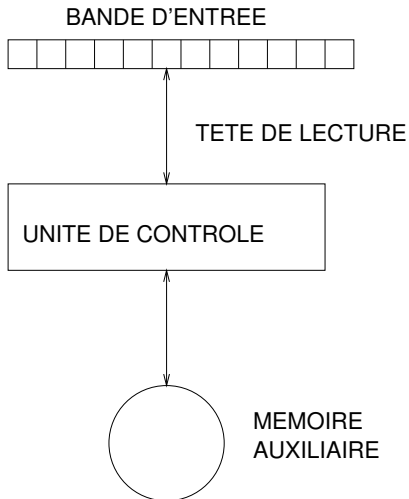
$$\begin{aligned} L &= \{a^n b^n c^n \mid n \geq 0\} \\ G &= \langle \{S, B, \bar{B}, C\}, \{a, b, c\}, \\ &\quad \left\{ \begin{array}{ll} S \rightarrow aSBC \mid \varepsilon, & CB \rightarrow \bar{B}B, \\ \bar{B}B \rightarrow \bar{B}C, & \bar{B}C \rightarrow BC, \\ aB \rightarrow ab, & bB \rightarrow bb, \\ bC \rightarrow bc, & cC \rightarrow cc \end{array} \right\}, S \rangle \\ L &= \{m\#m \mid \text{avec } m \in \{a, b\}^*\} \\ &\dots \end{aligned}$$

# Grammaire v/s Reconnaisseur

- Une **grammaire** d'un langage  $L$  permet de générer tous les mots appartenant à  $L$ .
- Un **reconnaisseur** pour un langage  $L$  est un programme qui prend en entrée un mot  $m$  et répond **oui** si  $m$  appartient à  $L$  et **non** sinon.
- Pour chaque classe de grammaire, il existe une classe de reconnaisseurs qui définit la même classe de langages.

Type de grammaire	Type de reconnaisseur
régulière	Automate fini
hors contexte	Automate à pile
contextuelle	Automate linéairement borné
sans restriction	Machine de Turing

# Représentation graphique d'un reconnaisseur



# Eléments d'un reconnaisseur

Un reconnaisseur est composé de quatre parties :

**1** une **bande de lecture**

- elle est composée d'une succession de cases.
- Chaque case pouvant contenir un seul symbole d'un alphabet d'entrée.
- C'est dans les cases de cette bande de lecture qu'est écrit le mot à reconnaître.

**2** une **tête de lecture**

- Elle peut lire une case à un instant donné.
- La case sur laquelle se trouve la tête de lecture à un moment donné s'appelle la **case courante**.
- La tête peut être déplacée par le reconnaisseur pour se positionner sur la case immédiatement à gauche ou à droite de la case courante.

**3** une **mémoire**

- Elle peut prendre des formes différentes.
- La mémoire permet de stocker des éléments d'un **alphabet de mémoire**.

# Éléments d'un reconnaisseur

## 4 une unité de contrôle

- Elle constitue le cœur d'un reconnaisseur.
- Elle peut être vue comme un programme qui dicte au reconnaisseur son comportement.
- Elle est définie par un ensemble fini d'états ainsi que par une fonction de transition qui décrit le passage d'un état à un autre en fonction du contenu de la case courante de la bande de lecture et du contenu de la mémoire.
- L'unité de contrôle décide aussi de la direction dans laquelle déplacer la tête de lecture et choisit quels symboles stocker dans la mémoire.
- Parmi les états d'un reconnaisseur, on distingue
  - des états initiaux, qui sont les états dans lesquels doit se trouver le reconnaisseur avant de commencer à reconnaître un mot
  - des états d'acceptation qui sont les états dans lequel doit se trouver le reconnaisseur après avoir reconnu un mot.

# Configuration et mouvement

- **Configuration** d'un reconnaisseur :
  - Etat de l'unité de contrôle
  - Contenu de la bande d'entrée et position de la tête
  - Contenu de la mémoire
- **Mouvement** : passage d'une configuration à une autre ( $C_1 \vdash C_2$ )

# Configurations

## ■ configuration initiale

- L'unité de contrôle est dans un état initial
- La tête est au début de la bande
- La mémoire contient un élément initial.

## ■ configuration d'acceptation

- L'unité de contrôle est dans un état d'acceptation
- La tête de lecture est à la fin de la bande
- La mémoire se trouve dans un état d'acceptation.



# Déterminisme

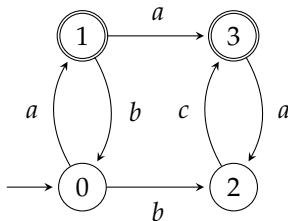
- L'unité de contrôle est dite **déterministe** si à toute configuration correspond au plus un mouvement. S'il peut exister plus d'un mouvement, elle est dite **non déterministe**.
- Le déterminisme est une propriété importante :
  - Un reconnaiseur déterministe reconnaît un mot de longueur  $n$  en  $O(n)$

# Reconnaissance

- Un mot  $m$  est **acceptée** par un reconnaisseur si, partant de l'état initial, avec  $m$  sur la bande d'entrée, le reconnaisseur peut faire une série de mouvements pour se retrouver dans un état d'acceptation.
- Le **langage accepté** par un reconnaisseur est l'ensemble de tous les mots qu'il accepte.

# Automates finis

- Le modèle le plus simple de reconnaisseur.
- Mémoire limitée aux seuls états.



Un état initial (0), un ou plusieurs états finaux (1 et 3).

# Définition

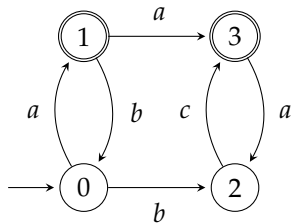
Un automate fini est un 5-uplet  $\langle Q, \Sigma, \delta, q_0, F \rangle$

- $Q$  est l'ensemble des états,
- $\Sigma$  est l'alphabet de l'entrée
- $\delta$  est la fonction de transition :

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$$

- $q_0 \in Q$  est l'état initial,
- $F \subseteq Q$  est l'ensemble des états d'acceptation.

## Exemple, suite



$$\langle Q, \Sigma, \delta, q_0, F \rangle$$

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

$$\Sigma = \{a, b, c\}$$

$$\delta(0, a) = \{1\}$$

$$\delta(0, b) = \{2\}$$

$$\delta(1, a) = \{3\}$$

$$\delta(1, b) = \{0\}$$

$$\delta(2, c) = \{3\}$$

$$\delta(3, a) = \{2\}$$

$$q_0 = 0$$

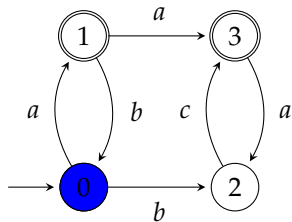
$$F = \{1, 3\}$$

# Configurations et mouvement

$$A = \langle Q, \Sigma, \delta, q_0, F \rangle$$

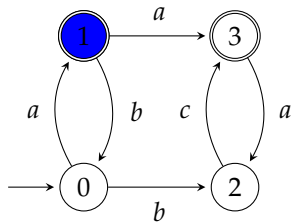
- **Configuration** :  $(q, m) \in Q \times \Sigma^*$  où :
  - $q$  représente l'état courant de l'unité de contrôle
  - $m$  est la partie du mot à reconnaître non encore lue. Le premier symbole de  $m$  (le plus à gauche) est celui qui se trouve sous la tête de lecture. Si  $m = \varepsilon$  alors tout le mot a été lu.
- **Configuration initiale** :  $(q_0, m)$  où  $m$  est le mot à reconnaître
- **Configuration d'acceptation** :  $(q, \varepsilon)$  avec  $q \in F$
- **Mouvement** :  $(q, aw) \vdash (q', w)$  si  $q' \in \delta(q, a)$ .

# Reconnaissance



$(0, ababaa)$

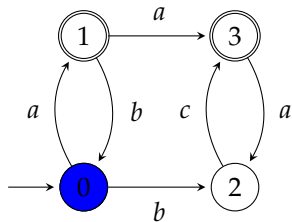
# Reconnaissance



$(0, ababaa)$   
 $\vdash (1, babbc)$

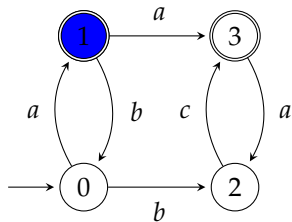


# Reconnaissance



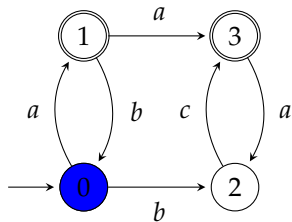
$(0, ababaa)$   
 $\vdash (1, babbc)$   
 $\vdash (0, abbc)$

# Reconnaissance



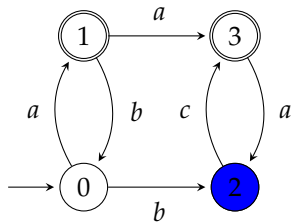
$(0, ababaa)$   
 $\vdash (1, babbc)$   
 $\vdash (0, abbc)$   
 $\vdash (1, bbc)$

# Reconnaissance



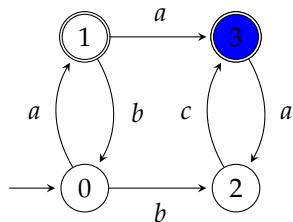
$(0, ababaa)$   
 $\vdash (1, babbc)$   
 $\vdash (0, abbc)$   
 $\vdash (1, bbc)$   
 $\vdash (0, bc)$

# Reconnaissance



$(0, ababaa)$   
 $\vdash (1, babbc)$   
 $\vdash (0, abbc)$   
 $\vdash (1, bbc)$   
 $\vdash (0, bc)$   
 $\vdash (2, c)$

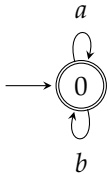
# Reconnaissance



$(0, ababaa)$   
 $\vdash (1, babbc)$   
 $\vdash (0, abbc)$   
 $\vdash (1, bbc)$   
 $\vdash (0, bc)$   
 $\vdash (2, c)$   
 $\vdash (3, \varepsilon)$

$$L = \{m \in \{a,b\}^*\}$$

$$L = \{m \in \{a,b\}^*\}$$

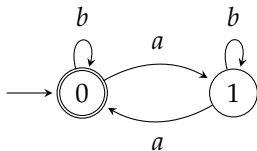


$$G = \langle \{S\}, \{a,b\}, \{S \rightarrow aS \mid bS \mid \varepsilon\}, S \rangle$$

$$L = \{m \in \{a,b\}^* \mid |m|_a \bmod 2 = 0\}$$



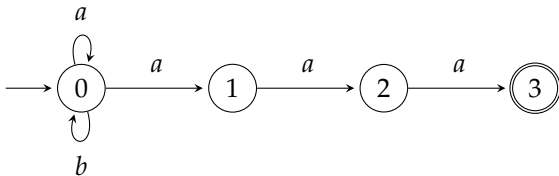
$$L = \{m \in \{a,b\}^* \mid |m|_a \bmod 2 = 0\}$$



$$G = \langle \{S, T\}, \{a, b\}, \{S \rightarrow aT \mid bS \mid \varepsilon, T \rightarrow aS \mid bT\}, S_0 \rangle$$

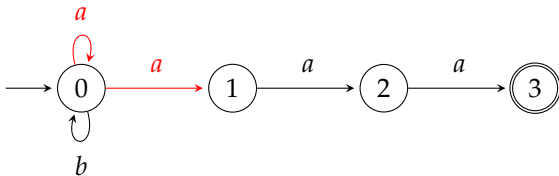
$$L = \{xaaa \mid x \in \{a,b\}^*\}$$

$$L = \{xaaa \mid x \in \{a,b\}^*\}$$



$$G = \langle \{S, T, U\}, \{a, b\}, \{S \rightarrow aS \mid bS \mid aT, T \rightarrow aU, U \rightarrow a\}, S \rangle$$

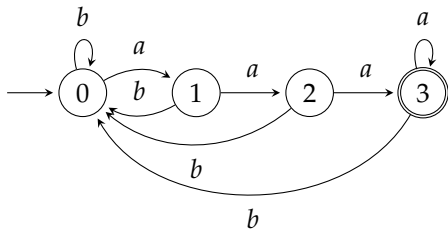
$$L = \{xaaa \mid x \in \{a,b\}^*\}$$



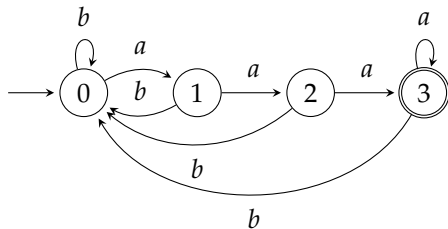
$$G = \langle \{S, T, U\}, \{a, b\}, \{S \rightarrow aS \mid bS \mid aT, T \rightarrow aU, U \rightarrow a\}, S \rangle$$

Non-déterminisme !

$$L = \{xaaa \mid x \in \{a,b\}^*\}$$



$$L = \{xaaa \mid x \in \{a,b\}^*\}$$



$(0, abaabaaa)$   
 $\vdash (1, baabaaa)$   
 $\vdash (0, aabaaa)$   
 $\vdash (1, abaaa)$   
 $\vdash (2, baaa)$   
 $\vdash (0, aaa)$   
 $\vdash (1, aa)$   
 $\vdash (2, a)$   
 $\vdash (3, \epsilon)$

# Déterminisme

- Tout langage régulier peut être reconnu par un automate fini déterministe
- Pour tout automate fini non déterministe  $A$ , on peut construire un automate déterministe  $A'$  avec  $L(A) = L(A')$
- Prix à payer : dans le pire des cas,  $|Q(A')| = 2^{|Q(A)|}$

# Limite des automates finis

- Certains langages ne peuvent pas être reconnus par les automates finis (ne peuvent être engendrés par une grammaire régulière)
- Exemple :  $L = \{a^n b^n \mid n \geq 0\}$
- Il faut mémoriser le nombre de  $a$  que l'on a lu pour vérifier que le mot possède autant de  $b$ .
- Pour mémoriser un nombre potentiellement infini de  $a$ , il faut un ensemble infini d'états !

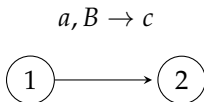


# Automates à pile

- Forme simple de mémoire : une pile.
- Mode de stockage *Last In First Out*.
- On ne peut accéder qu'à l'élément se trouvant au sommet de la pile.
- Deux opérations possibles :
  - **empiler** : ajouter un élément au sommet.
  - **dépiler** : enlever l'élément se trouvant au sommet.
- Elle commence avec un *symbole de fond de pile*  $\perp$ , que l'on ne peut pas dépiler.

La pile permet de stocker de l'information sans forcément multiplier le nombre d'états.

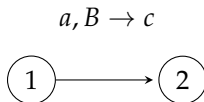
# Représentation graphique



Si l'automate est en 1, et que la tête de lecture est sur  $a$ , l'automate :

- décale la tête de lecture d'une case vers la droite
- dépile  $b$  ( $b$  doit être présent au sommet de la pile)
- empile  $c$
- va en 2

# Représentation graphique



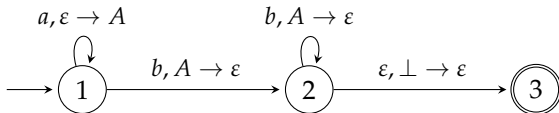
Si l'automate est en 1, et que la tête de lecture est sur  $a$ , l'automate :

- décale la tête de lecture d'une case vers la droite
- dépile  $b$  ( $b$  doit être présent au sommet de la pile)
- empile  $c$
- va en 2

cas particuliers

- si  $a = \varepsilon$ , l'automate peut franchir cet arc sans lire de symbole.
- si  $b = \varepsilon$ , l'automate peut franchir cet arc indépendamment du symbole se trouvant en sommet de pile (et ne le dépile pas).
- si  $c = \varepsilon$ , l'automate peut franchir cet arc sans rien empiler.

# Représentation graphique



# Définition formelle

Un automate à pile est un 6-uplet  $\langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$

- $Q$  est l'ensemble des états
- $\Sigma$  est l'alphabet d'entrée
- $\Gamma$  est l'alphabet de symboles de pile  
(en particulier,  $\perp \in \Gamma$ )
- $\delta$  est la fonction de transition :

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q \times \Gamma^*)$$

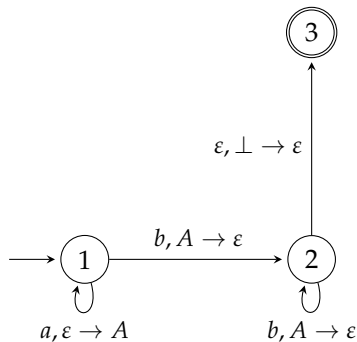
- $q_0 \in Q$  est l'état initial
- $F \subseteq Q$  est l'ensemble des états d'acceptation

# Configurations et mouvement

$$A = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$$

- **Configuration** :  $(q, m, \alpha) \in Q \times \Sigma^* \times \Gamma^*$  où :
  - $q$  représente l'état courant de l'unité de contrôle
  - $m$  est la partie du mot à reconnaître non encore lue. Le premier symbole de  $m$  (le plus à gauche) est celui qui se trouve sous la tête de lecture. Si  $m = \varepsilon$  alors tout le mot a été lu.
  - $\alpha$  représente le contenu de la pile. Le symbole le plus à gauche est le sommet de la pile. Si  $\alpha = \varepsilon$  alors la pile est vide.
- **Configuration initiale** :  $(q_0, m, \varepsilon)$  où  $m$  est le mot à reconnaître
- **Configuration d'acceptation** :  $(q, \varepsilon, \varepsilon)$  avec  $q \in F$
- **Mouvement** :  $(q, aw, Z\alpha) \vdash (q', w, \gamma\alpha)$  si  $(q', \gamma) \in \delta(q, a, Z)$ .

# Equivalence



$\langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$

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

$$\Sigma = \{a, b\}$$

$$\Gamma = \{A, \perp\}$$

$$\delta(0, \varepsilon, \varepsilon) = \{(1, \perp)\}$$

$$\delta(1, a, \varepsilon) = \{(1, A)\}$$

$$\delta(1, b, A) = \{(2, \varepsilon)\}$$

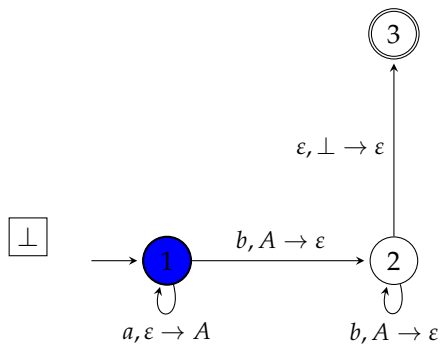
$$\delta(2, b, A) = \{(2, \varepsilon)\}$$

$$\delta(2, \varepsilon, \perp) = \{(3, \varepsilon)\}$$

$$q_0 = 0$$

$$F = \{3\}$$

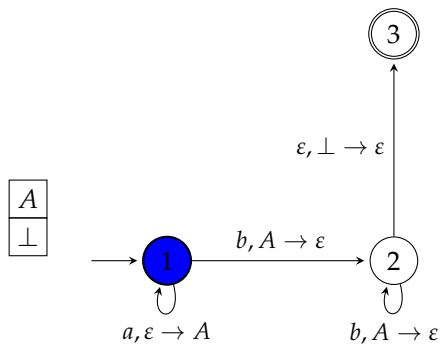
# Reconnaissance



$(1, aaabbb, \perp)$

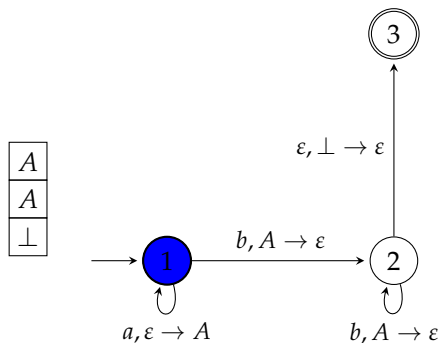


# Reconnaissance



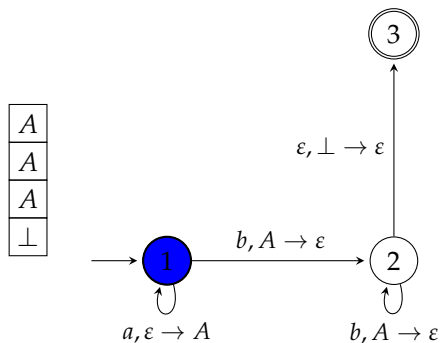
$(1, aaabbb, \perp)$   
 $\vdash (1, aabbb, A\perp)$

# Reconnaissance



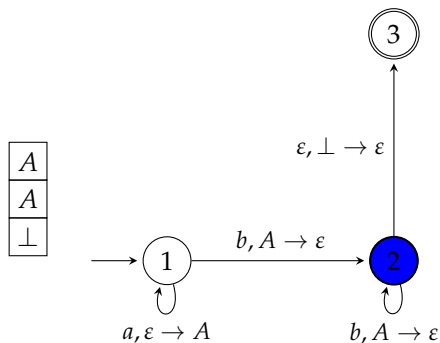
$(1, aaabbb, \perp)$   
 $\vdash (1, aabbb, A\perp)$   
 $\vdash (1, abbb, AA\perp)$

# Reconnaissance



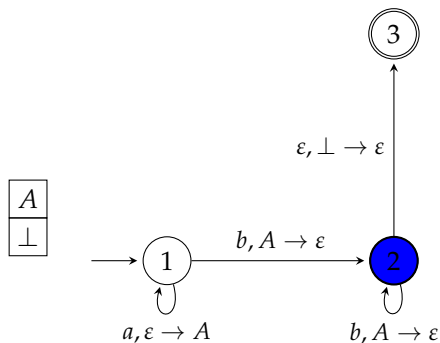
$(1, aaabbb, \perp)$   
 $\vdash (1, aabbb, A\perp)$   
 $\vdash (1, abbb, AA\perp)$   
 $\vdash (1, bbb, AAA\perp)$

# Reconnaissance



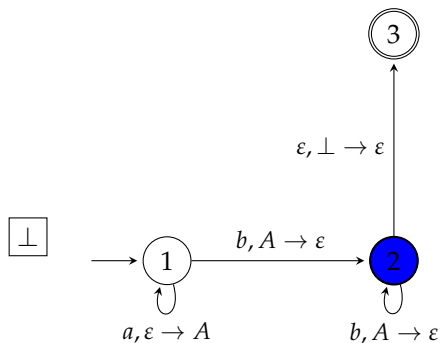
$(1, aaabbb, \perp)$   
 $\vdash (1, aabbb, A\perp)$   
 $\vdash (1, abbb, AA\perp)$   
 $\vdash (1, bbb, AAA\perp)$   
 $\vdash (2, bb, AA\perp)$

# Reconnaissance



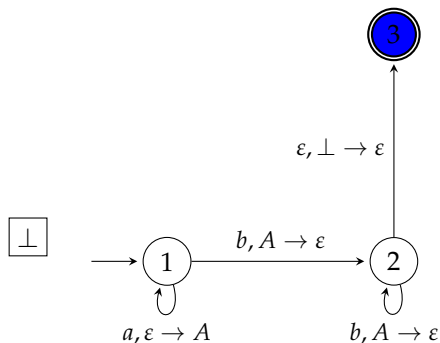
$(1, aaabbb, \perp)$   
 $\vdash (1, aabbb, A\perp)$   
 $\vdash (1, abbb, AA\perp)$   
 $\vdash (1, bbb, AAA\perp)$   
 $\vdash (2, bb, AA\perp)$   
 $\vdash (2, b, A\perp)$

# Reconnaissance



$(1, aaabbb, \perp)$   
 $\vdash (1, aabbb, A\perp)$   
 $\vdash (1, abbb, AA\perp)$   
 $\vdash (1, bbb, AAA\perp)$   
 $\vdash (2, bb, AA\perp)$   
 $\vdash (2, b, A\perp)$   
 $\vdash (2, \epsilon, \perp)$

# Reconnaissance



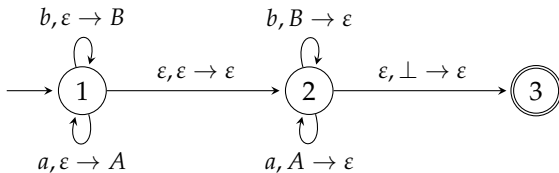
$(1, aaabbb, \perp)$   
 $\vdash (1, aabbb, A\perp)$   
 $\vdash (1, abbb, AA\perp)$   
 $\vdash (1, bbb, AAA\perp)$   
 $\vdash (2, bb, AA\perp)$   
 $\vdash (2, b, A\perp)$   
 $\vdash (2, \varepsilon, \perp)$   
 $\vdash (3, \varepsilon, \perp)$

$$L = \{mm^{-1} \mid m \in \{a,b\}^*\}$$

$$G = \langle \{S\}, \{a,b\}, \{S \rightarrow aSa \mid bSb \mid \varepsilon\}, S \rangle$$



$$L = \{mm^{-1} \mid m \in \{a,b\}^*\}$$

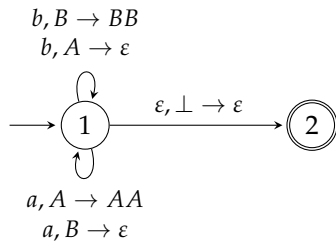


$$G = \langle \{S\}, \{a, b\}, \{S \rightarrow aSa \mid bSb \mid \varepsilon\}, S \rangle$$

$$L = \{m \in \{a,b\}^*, |m|_a = |m|_b\}$$

$$G = \langle \{S\}, \{a,b\}, \{S \rightarrow aSb \mid bSa \mid SS \mid \varepsilon\}, S \rangle$$

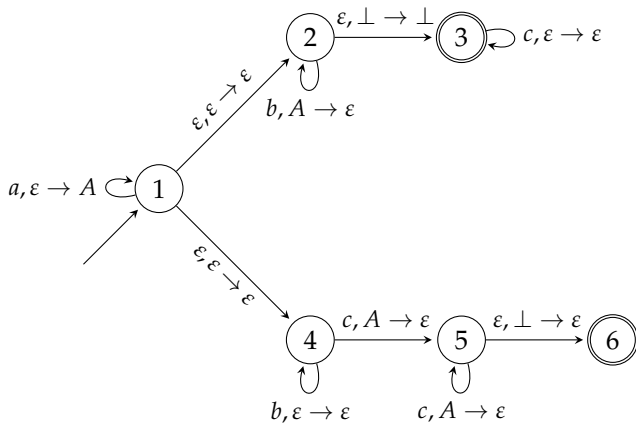
$$L = \{m \in \{a,b\}^*, |m|_a = |m|_b\}$$



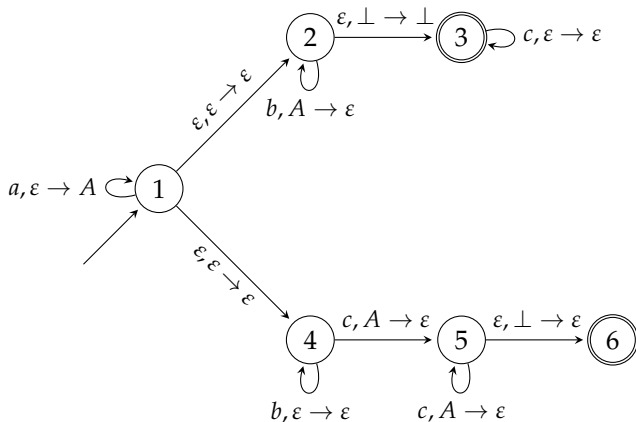
$$G = \langle \{S\}, \{a, b\}, \{S \rightarrow aSb \mid bSa \mid SS \mid \varepsilon\}, S \rangle$$

$$L = \{a^i b^j c^k \mid i > 0, i = j \text{ ou } i = k\}$$

$$L = \{a^i b^j c^k \mid i > 0, i = j \text{ ou } i = k\}$$



$$L = \{a^i b^j c^k \mid i > 0, i = j \text{ ou } i = k\}$$

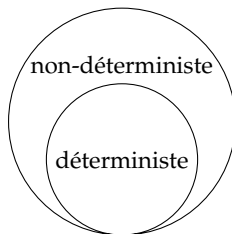


$L = \{a^i b^j c^k \mid i = j \text{ ou } i = k\}$  ne peut être reconnu par un automate à pile déterministe !

# Langages hors-contexte déterministes

Automate à pile déterministe :

- $|\delta(q, a, Z)| \leq 1$  pour tout  $q \in Q, a \in \Sigma \cup \{\varepsilon\}, Z \in \Gamma$
- si  $\delta(q, \varepsilon, Z)$  est défini, alors  $\delta(q, a, Z)$  ne peut être défini pour aucun  $a \in \Sigma$ .



# Limites des automates à pile

- Certains langages ne peuvent être reconnus par les automates à pile (ne peuvent être engendrés par une grammaire hors-contexte).
- Exemple : le langage  $m\#m$  avec  $m \in \{0,1\}^*$  :
  - 1 L'automate lit le premier  $m$  et le stocke dans la pile.
  - 2 Il lit le premier symbole du second  $m$ .
  - 3 Comment vérifier qu'il est identique au symbole se trouvant au fond de la pile ?



# Machines de Turing

- Proches des automates finis mais avec une mémoire infinie et à accès direct.
- Modèle plus proche d'un ordinateur.
- Une machine de Turing (MT) peut faire tout ce qu'un ordinateur peut faire.
- Thèse de Church-Turing : tout traitement réalisable par un algorithme peut être accompli par une machine de Turing.

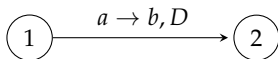
# Généralités

- La mémoire de la MT est matérialisée par une bande de lecture/écriture.
- Elle possède une tête de lecture/écriture pouvant se déplacer vers la gauche et vers la droite.
- Au départ, la bande contient le mot à reconnaître et possède des ☐ dans toutes les autres cases.

# Caractéristiques

- Une MT peut lire **et écrire** sur la bande de lecture/écriture.
- La tête de lecture/écriture peut se déplacer vers la droite **et vers la gauche**.
- La bande de lecture écriture est infinie.
- Lorsque la MT atteint l'état d'acceptation ou l'état de rejet, elle s'arrête et accepte ou rejette le mot.
- Si la MT n'atteint pas l'état d'acceptation ou de rejet, elle peut continuer indéfiniment.

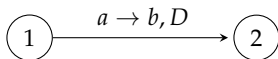
# Représentation graphique



La machine est en 1, la tête de lecture est sur  $a$ , elle :

- écrit un  $b$  sur la bande (à la place du  $a$ ),
- décale la tête de lecture d'une case vers la droite ( $D$ )
- va en 2.

# Représentation graphique



La machine est en 1, la tête de lecture est sur  $a$ , elle :

- écrit un  $b$  sur la bande (à la place du  $a$ ),
- décale la tête de lecture d'une case vers la droite ( $D$ )
- va en 2.

Cas particuliers :

- $a \rightarrow G$  : la machine n'écrit rien.
- $a$  : la machine n'écrit rien et ne bouge plus (elle arrive généralement dans un état final).

# Définition

Une MT est un octuplet  $\langle Q, \Sigma, \Gamma, \square, \delta, q_0, q_A, q_R \rangle$  où :

- $Q$  est l'ensemble des états,
- $\Sigma$  est l'alphabet de l'entrée (qui ne contient pas le symbole spécial  $\square$ ),
- $\Gamma$  est l'alphabet de la bande ( $\square \in \Gamma$  et  $\Sigma \subseteq \Gamma$ ),
- $\delta$  est la fonction de transition :

$$\delta : Q \times (\Gamma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q \times \Gamma \times \{D, G\})$$

- $q_0 \in Q$  est l'état initial,
- $q_A \in Q$  est l'état d'acceptation,
- $q_R \in Q$  est l'état de rejet, avec  $q_R \neq q_A$ .

# Configurations et mouvement

$$M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A \rangle$$

- **Configuration** :  $(u, q, av) \in \Gamma^* \times Q \times \Gamma^*$  où :
  - $q$  représente l'état courant de l'unité de contrôle
  - $u$  est la partie de la bande se trouvant à gauche de la tête.
  - $v$  est la partie de la bande se trouvant à droite de la tête.
  - $a$  est le symbole se trouvant sous la tête.
- **Configuration initiale** :  $(\varepsilon, q_0, m)$  où  $m$  est le mot à reconnaître
- **Configuration d'acceptation** :  $(u, q_A, v)$
- **Configuration de rejet** :  $(u, q_R, v)$
- **Mouvement** :  $(ua, q_i, bv) \vdash (u, q_j, acv)$  si  $(q_j, c, G) \in \delta(q_i, b)$ .

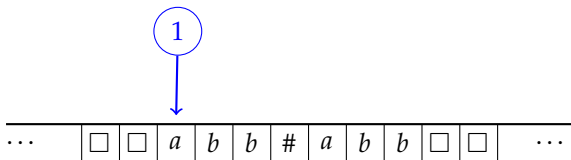
# Exemple

Principe d'une machine reconnaissant  $L = \{m\#m \mid m \in \{0,1\}^*\}$ .

- 1 Fait des allers-retours entre les deux occurrences de  $m$  pour vérifier qu'elles contiennent bien le même symbole. Si ce n'est pas le cas, ou qu'un  $\#$  supplémentaire est détecté, va dans l'état de rejet. Les symboles sont éliminés au fur et à mesure qu'ils sont vérifiés (par le symbole  $x$  à gauche, et  $z$  à droite).
- 2 Lorsque tous les symboles au gauche de  $\#$  ont été éliminés, vérifie qu'il ne reste plus de symboles à droite de  $\#$ . Si c'est le cas, va dans l'état d'acceptation, sinon va dans l'état de rejet.



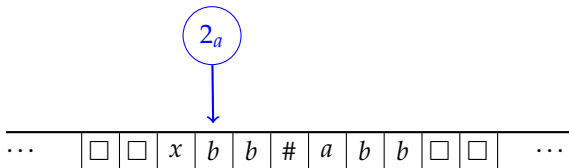
## Exemple — début d'exécution



$$\delta(1, a) = (x, 2_a, D)$$

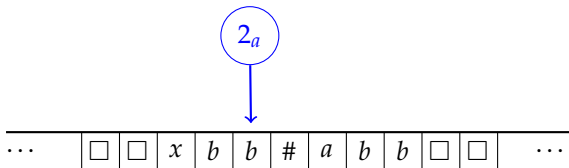
enregistre *a*

## Exemple — début d'exécution



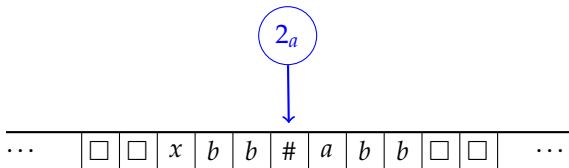
$$\begin{aligned}\delta(1, a) &= (x, 2_a, D) && \text{enregistre } a \\ \delta(2_a, \alpha) &= (\alpha, 2_a, D) \text{ pour } \alpha \neq \# && \text{cherche } \#\end{aligned}$$

## Exemple — début d'exécution



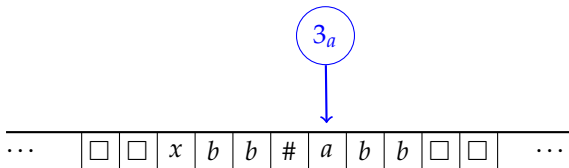
$$\begin{aligned}\delta(1, a) &= (x, 2_a, D) && \text{enregistre } a \\ \delta(2_a, \alpha) &= (\alpha, 2_a, D) \text{ pour } \alpha \neq \# && \text{cherche } \#\end{aligned}$$

## Exemple — début d'exécution



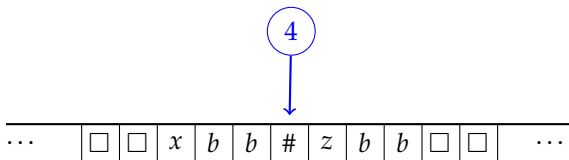
$$\begin{aligned}\delta(1, a) &= (x, 2_a, D) && \text{enregistre } a \\ \delta(2_a, \alpha) &= (\alpha, 2_a, D) \text{ pour } \alpha \neq \# && \text{cherche } \# \\ \delta(2_a, \#) &= (\#, 3_a, D) && 3_a \text{ cherche une lettre non-} z\end{aligned}$$

# Exemple — début d'exécution



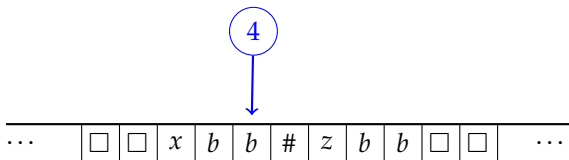
$\delta(1, a)$	$=$	$(x, 2_a, D)$	enregistre $a$
$\delta(2_a, \alpha)$	$=$	$(\alpha, 2_a, D)$	pour $\alpha \neq \#$ cherche #
$\delta(2_a, \#)$	$=$	$(\#, 3_a, D)$	$3_a$ cherche une lettre non-z
$\delta(3_a, a)$	$=$	$(x, 4, G)$	vérifie la lettre

## Exemple — début d'exécution



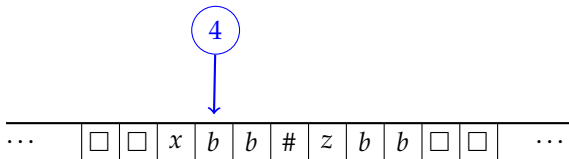
$\delta(1, a)$	$=$	$(x, 2_a, D)$	enregistre $a$
$\delta(2_a, \alpha)$	$=$	$(\alpha, 2_a, D)$ pour $\alpha \neq \#$	cherche $\#$
$\delta(2_a, \#)$	$=$	$(\#, 3_a, D)$	$3_a$ cherche une lettre non- $z$
$\delta(3_a, a)$	$=$	$(x, 4, G)$	vérifie la lettre
$\delta(4, \alpha)$	$=$	$(\alpha, 4, G)$ pour $\alpha \neq x$	revient vers $x$

## Exemple — début d'exécution



$\delta(1, a)$	$=$	$(x, 2_a, D)$	enregistre $a$
$\delta(2_a, \alpha)$	$=$	$(\alpha, 2_a, D)$	cherche $\#$
$\delta(2_a, \#)$	$=$	$(\#, 3_a, D)$	$3_a$ cherche une lettre non- $z$
$\delta(3_a, a)$	$=$	$(x, 4, G)$	vérifie la lettre
$\delta(4, \alpha)$	$=$	$(\alpha, 4, G)$	revient vers $x$

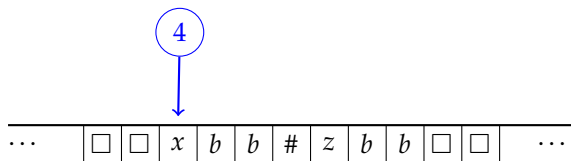
## Exemple — début d'exécution



$\delta(1, a)$	$=$	$(x, 2_a, D)$	enregistre $a$
$\delta(2_a, \alpha)$	$=$	$(\alpha, 2_a, D)$ pour $\alpha \neq \#$	cherche $\#$
$\delta(2_a, \#)$	$=$	$(\#, 3_a, D)$	$3_a$ cherche une lettre non- $z$
$\delta(3_a, a)$	$=$	$(x, 4, G)$	vérifie la lettre
$\delta(4, \alpha)$	$=$	$(\alpha, 4, G)$ pour $\alpha \neq x$	revient vers $x$

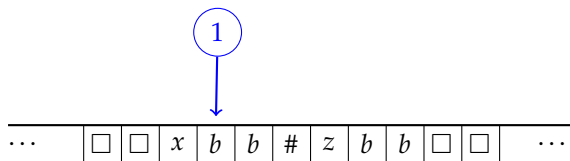


# Exemple — début d'exécution



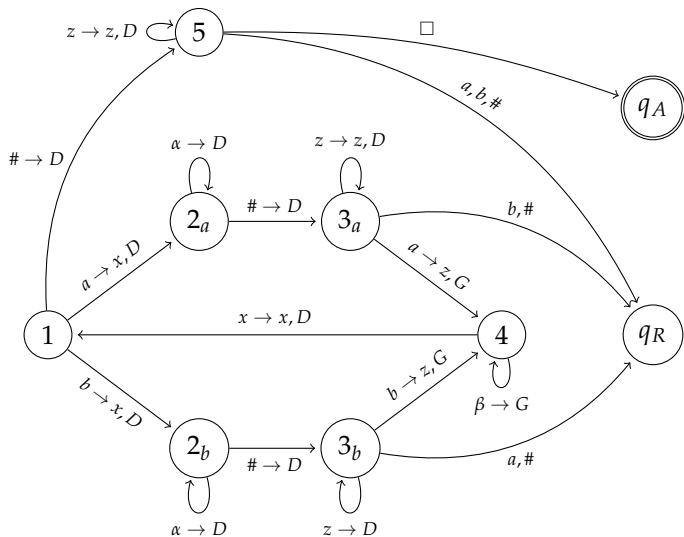
$\delta(1, a)$	$=$	$(x, 2_a, D)$	enregistre $a$
$\delta(2_a, \alpha)$	$=$	$(\alpha, 2_a, D)$ pour $\alpha \neq \#$	cherche $\#$
$\delta(2_a, \#)$	$=$	$(\#, 3_a, D)$	$3_a$ cherche une lettre non- $z$
$\delta(3_a, a)$	$=$	$(x, 4, G)$	vérifie la lettre
$\delta(4, \alpha)$	$=$	$(\alpha, 4, G)$ pour $\alpha \neq x$	revient vers $x$
$\delta(4, x)$	$=$	$(x, 1, D)$	relance le processus

# Exemple — début d'exécution



$\delta(1, a)$	$=$	$(x, 2_a, D)$	enregistre $a$
$\delta(2_a, \alpha)$	$=$	$(\alpha, 2_a, D)$ pour $\alpha \neq \#$	cherche $\#$
$\delta(2_a, \#)$	$=$	$(\#, 3_a, D)$	$3_a$ cherche une lettre non- $z$
$\delta(3_a, a)$	$=$	$(x, 4, G)$	vérifie la lettre
$\delta(4, \alpha)$	$=$	$(\alpha, 4, G)$ pour $\alpha \neq x$	revient vers $x$
$\delta(4, x)$	$=$	$(x, 1, D)$	relance le processus
$\delta(1, b)$	$=$	$(x, 2_b, D) \dots$	

## Exemple — machine



$$\alpha = \Gamma \setminus \{\#\}, \quad \beta = \Gamma \setminus \{x\}.$$

# Déterminisme

- Pour toute MT  $A$  non déterministe, il existe une MT  $A'$  telle que  $L(A) = L(A')$ .
- Le non déterminisme n'augmente pas la puissance du modèle des MT.

# Langages récursivement énumérables

- Un langage est récursivement énumérable si et seulement si il existe une MT qui le reconnaît.
- Un langage est récursivement énumérable si et seulement si il existe une MT déterministe qui le reconnaît.

# Rapports avec la compilation

- Analyse lexicale et automates finis
- Analyse syntaxique et automates à pile
- Production de code et machines de Turing

# Sources

- Michael Sipser,  
*Introduction to the Theory of Computation*.  
PWS Publishing Company, 1997.
- John Hopcroft, Rajeev Motwani, Jeffrey Ullman,  
*Introduction to Automata Theory, Languages and Computation*.  
2ème édition, Pearson Education International, 2001.
- John Aho, Jeffrey Ullman, *The Theory of Parsing, Translation and Compiling, Vol I : Parsing*.  
Prentice-Hall, 1972