Théorie des langages (Introduction/Rappels)

Alexis Nasr
Franck Dary
Pacome Perrôtin

Compilation – L3 Informatique
Département Informatique et Interactions
Aix Marseille Université

Langages

Grammaires

- Arbres de dérivation
- Types de grammaires

Reconnaisseurs

- Généralités
- Automates finis
- Automates à pile
- Machines de Turing

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 Σ.
- Une suite de symboles, appartenant à un alphabet Σ, mis bout à bout est appelé un mot (ou une *chaîne*) sur Σ. Le mot de longueur zéro est noté ε.
- 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 Σ est noté Σ^*.
- Un langage sur un alphabet Σ est un ensemble de mots construits sur Σ. Tout langage défini sur Σ est donc une partie de Σ^*.

Exemples de langages

$$\Sigma = \{a\} \qquad L_1 = \{\varepsilon, a, aa, aaa, \ldots\}$$

$$\Sigma = \{a, b\} \qquad L_2 = \{\varepsilon, ab, aabb, aaabbb, aaaabbbb, \ldots\}$$

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

$$\Sigma = \{a, b, c\} \qquad L_4 = \{\varepsilon, abc, aabbcc, aaabbbccc, \ldots\}$$

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_1L_2	$\{xy \mid x \in L_1 \text{ et} y \in L_2\}$
Auto concaténation	$\overbrace{L \ldots L}^{n}$	L^n
Fermeture de Kleene	L^*	$\bigcup_{k \geq 0} L^k$

Comment décrire un langage ?

- Énumération
  $L_2 = \{\varepsilon, ab, aabb, aaabbb, aaaabbbb, \dots\}$
- 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
  $G = \langle \{S\}, \{a, b\}, \{S \to aSb \mid \varepsilon\}, S \rangle$

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.
- Σ est un ensemble de symboles terminaux, appelé l'alphabet terminal, tel que N et Σ soient disjoints.
- P est un ensemble fini de fini de règles de production ou règles de réécriture.
- Une règle est de la forme :

  $$\alpha \rightarrow \beta$$

  α est appelé partie gauche de la règle
  β est appelé partie droite de la règle
- α et β peuvent prendre des formes différentes mais on a toujours
  $\alpha \in (N \cup \Sigma)^* N (N \cup \Sigma)^*$
  $\beta \in (N \cup \Sigma)^*$
- S est un élément de N appelé l'axiome de la grammaire.

Notation

Pour alléger les notations, on note :

$$\alpha \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

les n règles :

$$\alpha \to \beta_1 \;,\; \alpha \to \beta_2 \;, \ldots ,\; \alpha \to \beta_n$$

Quelques exemples de grammaires

- $\langle \{S\}, \{a\}, \{S \to Sa \mid \varepsilon\}, S \rangle$
- $\langle \{S\}, \{a, b\}, \{S \to aSb \mid \varepsilon\}, S \rangle$
- $\langle \{S\}, \{a, b\}, \{S \to aSa \mid bSb \mid aa \mid bb\}, S \rangle$
- $\langle \{S, S_1, S_2\}, \{a, b, c\}, \left\{ \begin{array}{ccccccc} S & \to & aS_1c, & S_1 & \to & b \mid SS_2, \\ cS_2 & \to & S_2c, & bS_2 & \to & bb \end{array} \right\}, S \rangle$

Proto-mots d'une grammaire

Les proto-mots 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 un proto-mot de G
- si $\alpha\beta\gamma$ est un proto-mot de G et $\beta \to \delta \in P$ alors $\alpha\delta\gamma$ est un proto-mot de G.

Un proto-mot 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 un proto-mot $\alpha\delta\gamma$ à partir d'un proto-mot $\alpha\beta\gamma$ 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 $\alpha \overset{k}{\Rightarrow} \beta$ pour indiquer que β se dérive de α en k étapes.
- On définit aussi les deux notations $\overset{+}{\Rightarrow}$ et $\overset{*}{\Rightarrow}$ de la façon suivante :
  - $\alpha \overset{+}{\Rightarrow} \beta \equiv \alpha \overset{k}{\Rightarrow} \beta$ avec $k > 0$
  - $\alpha \overset{*}{\Rightarrow} \beta \equiv \alpha \overset{k}{\Rightarrow} \beta$ avec $k \geq 0$

Attention

Les symboles $\Rightarrow$ et $\rightarrow$ ne représentent pas la même chose.

Conventions

- Les symboles non terminaux appartenant à N sont représentés par des lettres latines majuscules : $A, B, C, S, E, T \dots$
- Les symboles terminaux appartenant à Σ sont représentés par des lettres latines minuscules : $a, b, c, d \dots$
- Les proto-mots appartenant à $(N \cup \Sigma)^*$ sont représentés par des lettres grecques minuscules : $\alpha, \beta, \gamma, \varepsilon \dots$
- L'axiome est représenté par le non-terminal S et constitue la partie gauche de la première règle de production

Langage engendré par une grammaire

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

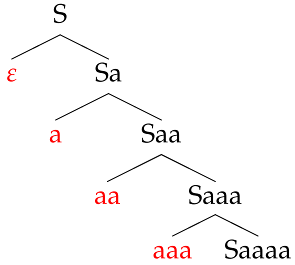
$$L(G) = \{m \in \Sigma^* \mid S \stackrel{+}{\Rightarrow} m\}$$

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

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

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

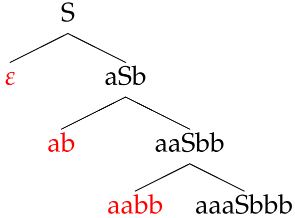
sous-ensemble des proto-mots de G



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

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

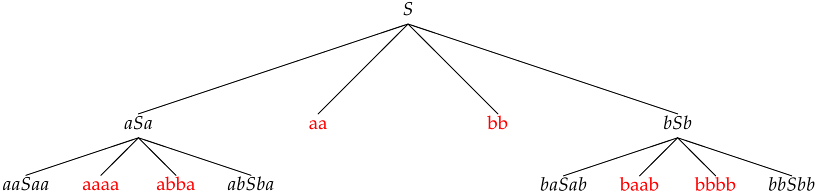
sous-ensemble des proto-mots de G



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

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

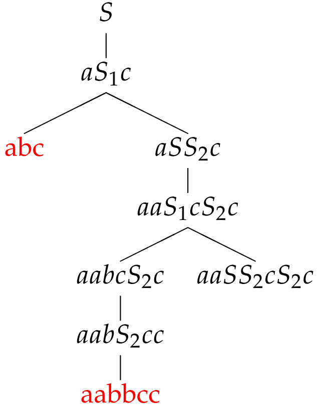
sous-ensemble des proto-mots de G



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

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

sous-ensemble des proto-mots de G



Sens de dérivation

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

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

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

Sens de dérivation

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

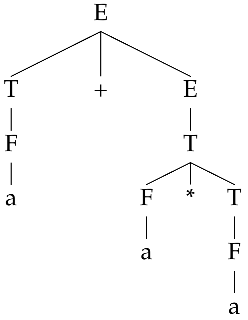
$$
\begin{array}{rcl}
E & \Rightarrow & T + E \\
  & \Rightarrow & F + E \\
  & \Rightarrow & a + E \\
  & \Rightarrow & a + T \\
  & \Rightarrow & a + F * T \\
  & \Rightarrow & a + a * T \\
  & \Rightarrow & a + a * F \\
  & \Rightarrow & a + a * a
\end{array}
$$

Dérivation droite :

...

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

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 $X_1, X_2, ..., X_n$ alors la règle $A \rightarrow X_1, X_2, ..., X_n$ appartient à P.

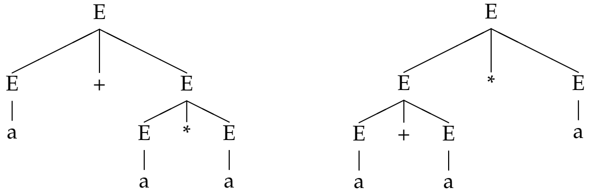
Arbre de dérivation

- 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 :

$$E \to E + E \mid E * E \mid a$$



Symboles et règles de production utiles

Pour manipuler une grammaire, il est souhaitable que tous les symboles et règles de production soient utiles :

- Un symbole terminal ou non-terminal est utile s'il apparaît dans une règle de production utile
- Une règle de production est utile si :
  1. elle peut générer des mots
  2. le symbole non-terminal de la partie gauche peut être généré (sauf l'axiome, qui peut par définition ne pas être généré)
  3. elle n'est pas de la forme $\alpha \rightarrow \alpha$

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 :

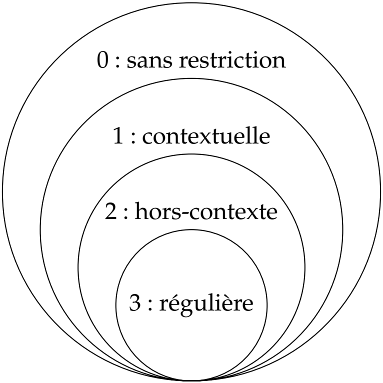
- Une règle est régulière à gauche si et seulement si elle est de la forme $A \to xB$ ou $A \to x$ avec $A, B \in N$ et $x \in \Sigma^*$.
- Une règle est régulière à droite si et seulement si elle est de la forme $A \to Bx$ ou $A \to x$ avec $A, B \in N$ et $x \in \Sigma^*$.
- Une règle $A \to \alpha$ est un règle hors-contexte si et seulement si : $A \in N$ et $\alpha \in (N \cup \Sigma)^*$
- Une règle $\alpha \to \beta$ 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 réecrit B uniquement dans le contexte g_d.
- Une règle $\alpha \to \beta$ 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	récursivement énumérable

Exemples de langages réguliers

$$L_1 = \{m \in \{a,b\}^*\}$$

$$L_2 = \{m \in \{a,b\}^* \mid |m|_a \mod 2 = 0\}$$

$$L_3 = \{m \in \{a,b\}^* \mid m = xaaa \text{ avec } x \in \{a,b\}^*\}$$

$$L_4 = \{m \in \{a,b\}^* \mid |m|_a \mod 2 = 0 \text{ et} |m|_b \mod 2 = 0\}$$

Exemples de langages réguliers

$$
\begin{array}{rcl}
L_1 & = & \{m \in \{a,b\}^*\} \\
G_1 & = & \langle \{S\}, \{a,b\}, \{S \rightarrow aS \mid bS \mid \varepsilon\}, S \rangle
\end{array}
$$

$$L_2 = \{m \in \{a,b\}^* \mid |m|_a \mod 2 = 0\}$$

$$L_3 = \{m \in \{a,b\}^* \mid m = xaaa \text{ avec } x \in \{a,b\}^*\}$$

$$L_4 = \{m \in \{a,b\}^* \mid |m|_a \mod 2 = 0 \text{ et} |m|_b \mod 2 = 0\}$$

Exemples de langages réguliers

$$
\begin{array}{rcl}
L_1 & = & \{m \in \{a,b\}^*\} \\
G_1 & = & \langle \{S\}, \{a,b\}, \{S \to aS \mid bS \mid \varepsilon\}, S\rangle \\
\\
L_2 & = & \{m \in \{a,b\}^* \mid |m|_a \mod 2 = 0\} \\
G_2 & = & \langle \{S,T\}, \{a,b\}, \left\{ \begin{array}{rcl} S & \to & aT \mid bS \mid \varepsilon, \\ T & \to & aS \mid bT \end{array} \right\}, S\rangle \\
L_3 & = & \{m \in \{a,b\}^* \mid m = xaaa \text{ avec } x \in \{a,b\}^*\}
\end{array}
$$

$$
L_4 = \{m \in \{a,b\}^* \mid |m|_a \mod 2 = 0 \text{ et} |m|_b \mod 2 = 0\}
$$

Exemples de langages réguliers

$$
\begin{array}{rcl}
L_1 & = & \{m \in \{a,b\}^*\} \\
G_1 & = & \langle\{S\},\{a,b\},\{S \to aS \mid bS \mid \varepsilon\},S\rangle \\
\\
L_2 & = & \{m \in \{a,b\}^* \mid |m|_a \mod 2 = 0\} \\
G_2 & = & \langle\{S,T\},\{a,b\},\left\{\begin{array}{lcl} S & \to & aT \mid bS \mid \varepsilon, \\ T & \to & aS \mid bT \end{array}\right\},S\rangle \\
L_3 & = & \{m \in \{a,b\}^* \mid m = xaaa \text{ avec } x \in \{a,b\}^*\} \\
G_3 & = & \langle\{S,T,U\},\{a,b\},\left\{\begin{array}{lcl} S & \to & aS \mid bS \mid aT, \\ T & \to & aU, \\ U & \to & a \end{array}\right\},S\rangle \\
L_4 & = & \{m \in \{a,b\}^* \mid |m|_a \mod 2 = 0 \text{ et} |m|_b \mod 2 = 0\}
\end{array}
$$

Exemples de langages réguliers

$$
\begin{array}{rcl}
L_1 & = & \{m \in \{a,b\}^*\} \\
G_1 & = & \langle \{S\}, \{a,b\}, \{S \to aS \mid bS \mid \varepsilon\}, S\rangle \\
\\
L_2 & = & \{m \in \{a,b\}^* \mid |m|_a \mod 2 = 0\} \\
G_2 & = & \langle \{S,T\}, \{a,b\}, \left\{ \begin{array}{lcl} S & \to & aT \mid bS \mid \varepsilon, \\ T & \to & aS \mid bT \end{array} \right\}, S\rangle \\
L_3 & = & \{m \in \{a,b\}^* \mid m = xaaa \text{ avec } x \in \{a,b\}^*\} \\
G_3 & = & \langle \{S,T,U\}, \{a,b\}, \left\{ \begin{array}{lcl} S & \to & aS \mid bS \mid aT, \\ T & \to & aU, \\ U & \to & a \end{array} \right\}, S\rangle \\
L_4 & = & \{m \in \{a,b\}^* \mid |m|_a \mod 2 = 0 \text{ et} |m|_b \mod 2 = 0\} \\
G_4 & = & \langle \{S,T,U,V\}, \{a,b\}, \\
 & & \left\{ \begin{array}{lclclcl} S & \to & aT \mid bU \mid \varepsilon, & T & \to & aS \mid bV, \\ V & \to & aU \mid bT, & U & \to & aV \mid bS \end{array} \right\}, S\rangle
\end{array}
$$

Exemples de langages hors-contexte

$L_1 = \{a^n b^n \mid n \geq 0\}$

$L_2 = \{mm^{-1} \mid m \in \{a, b\}^*\}$ (langage miroir - palindromes paires)

Exemples de langages hors-contexte

$L_1 = \{a^n b^n \mid n \geq 0\}$
$G_1 = \langle \{S\}, \{a, b\}, \{S \rightarrow aSb \mid \varepsilon\}, S \rangle$

$L_2 = \{mm^{-1} \mid m \in \{a, b\}^*\}$ (langage miroir - palindromes paires)

Exemples de langages hors-contexte

$L_1 = \{a^n b^n \mid n \geq 0\}$
$G_1 = \langle \{S\}, \{a, b\}, \{S \to aSb \mid \varepsilon\}, S \rangle$

$L_2 = \{mm^{-1} \mid m \in \{a, b\}^*\}$ (langage miroir - palindromes paires)
$G_2 = \langle \{S\}, \{a, b\}, \{S \to aSa \mid bSb \mid \varepsilon\}, S \rangle$

Exemples de langages contextuels

$$L_1 = \{a^n b^n c^n \mid n \geq 0\}$$

$$L_2 = \{m\sharp m \mid \text{avec } m \in \{a,b\}^*\}$$

Exemples de langages contextuels

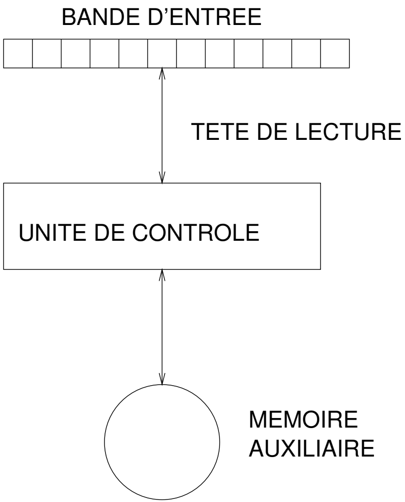
$$
\begin{array}{rcl}
L_1 & = & \{a^n b^n c^n \mid n \geq 0\} \\
G_1 & = & \langle \{S, B, \bar{B}, C\}, \{a, b, c\}, \\
 & & \left\{ \begin{array}{rclrcl}
S & \rightarrow & aSBC \mid \varepsilon, & CB & \rightarrow & \bar{B}B, \\
\bar{B}B & \rightarrow & \bar{B}C, & \bar{B}C & \rightarrow & BC, \\
aB & \rightarrow & ab, & bB & \rightarrow & bb, \\
bC & \rightarrow & bc, & cC & \rightarrow & cc
\end{array} \right\}, S \rangle \\
L_2 & = & \{m \sharp m \mid \text{avec } m \in \{a, b\}^*\} \\
 & & \ldots
\end{array}
$$

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



Éléments 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

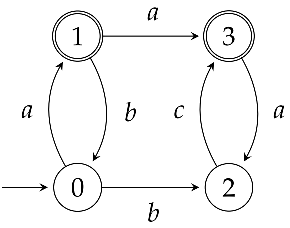
- L'unité de contrôle est dite déterministe si à toute configuration correspond au plus un mouvement. S'il peut exister plus d'un mouvement, elle est dite non déterministe.
- Le déterminisme est une propriété importante :
  - Un reconnaisseur déterministe reconnaît un mot de longueur n en $O(n)$

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

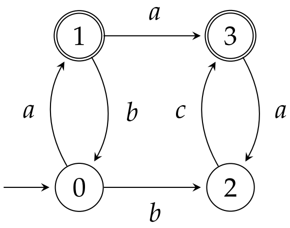
Un automate fini est un 5-uplet $\langle Q, \Sigma, \delta, q_0, F \rangle$

- Q est l'ensemble des états,
- Σ est l'alphabet de l'entrée
- δ est la fonction de transition :

$$\delta \; : \; Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \wp(Q)$$

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

Exemple



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

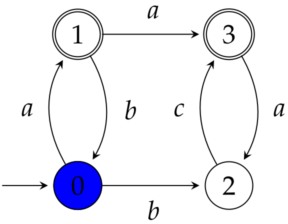
$$\begin{array}{rcl}
Q & = & \{0,1,2,3\} \\
\Sigma & = & \{a,b,c\} \\
\delta(0,a) & = & \{1\} \\
\delta(0,b) & = & \{2\} \\
\delta(1,a) & = & \{3\} \\
\delta(1,b) & = & \{0\} \\
\delta(2,c) & = & \{3\} \\
\delta(3,a) & = & \{2\} \\
q_0 & = & 0 \\
F & = & \{1,3\}
\end{array}$$

Configurations et mouvement

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

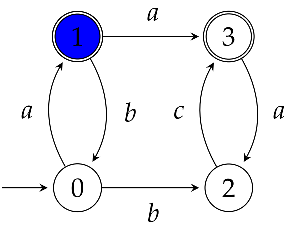
- Configuration : $(q, m) \in Q \times \Sigma^*$ où :
  - q représente l'état courant de l'unité de contrôle
  - m est la partie du mot à reconnaître non encore lue. Le premier symbole de m (le plus à gauche) est celui qui se trouve sous la tête de lecture. Si $m = \varepsilon$ alors tout le mot a été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

Reconnaissance



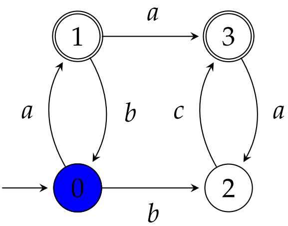
$(0, ababbc)$

Reconnaissance



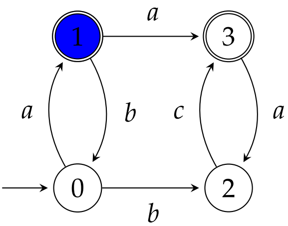
$$
\begin{array}{ll}
 & (0, ababbc) \\
\vdash & (1, babbc)
\end{array}
$$

Reconnaissance



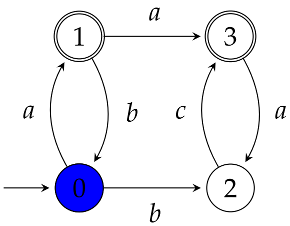
$$
\begin{array}{ll}
 & (0, ababbc) \\
\vdash & (1, babbc) \\
\vdash & (0, abbc)
\end{array}
$$

Reconnaissance



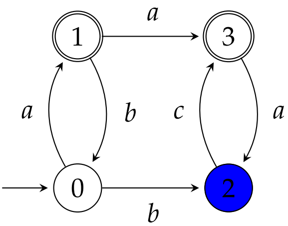
$$
\begin{array}{ll}
 & (0, ababbc) \\
\vdash & (1, babbc) \\
\vdash & (0, abbc) \\
\vdash & (1, bbc)
\end{array}
$$

Reconnaissance



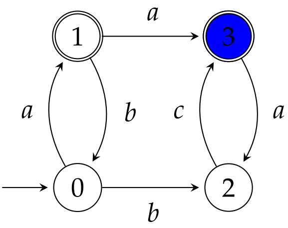
$$
\begin{array}{ll}
 & (0, ababbc) \\
\vdash & (1, babbc) \\
\vdash & (0, abbc) \\
\vdash & (1, bbc) \\
\vdash & (0, bc)
\end{array}
$$

Reconnaissance



$$
\begin{array}{ll}
 & (0, ababbc) \\
\vdash & (1, babbc) \\
\vdash & (0, abbc) \\
\vdash & (1, bbc) \\
\vdash & (0, bc) \\
\vdash & (2, c)
\end{array}
$$

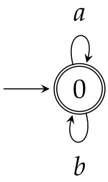
Reconnaissance



$$
\begin{array}{ll}
 & (0, ababbc) \\
\vdash & (1, babbc) \\
\vdash & (0, abbc) \\
\vdash & (1, bbc) \\
\vdash & (0, bc) \\
\vdash & (2, c) \\
\vdash & (3, \varepsilon)
\end{array}
$$

$$L_1 = \{m \in \{a,b\}^*\}$$

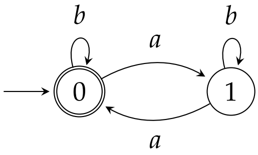
$L_1 = \{m \in \{a,b\}^*\}$



$$G_1 = \langle \{S\}, \{a,b\}, \{S \to aS \mid bS \mid \varepsilon\}, S \rangle$$

$$L_2 = \{m \in \{a,b\}^* \mid |m|_a \mod 2 = 0\}$$

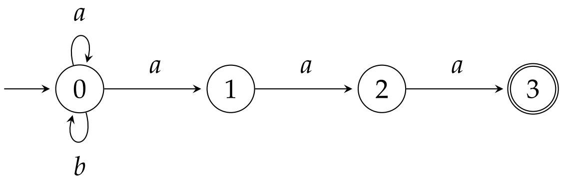
$L_2 = \{m \in \{a,b\}^* \mid |m|_a \mod 2 = 0\}$



$$G_2 = \langle \{S, T\}, \{a, b\}, \{S \to aT \mid bS \mid \varepsilon, T \to aS \mid bT\}, S \rangle$$

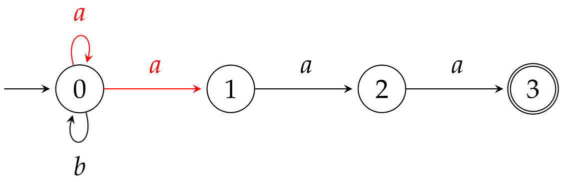
$L_3 = \{xaaa \mid x \in \{a,b\}^*\}$

$L_3 = \{xaaa \mid x \in \{a,b\}^*\}$



$$G_3 = \langle \{S, T, U\}, \{a, b\}, \{S \to aS \mid bS \mid aT, T \to aU, U \to a\}, S \rangle$$

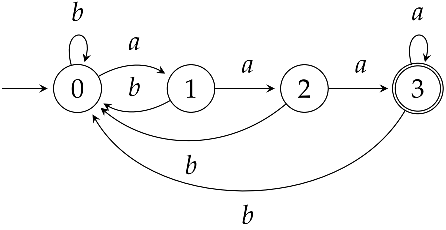
$L_3 = \{xaaa \mid x \in \{a,b\}^*\}$



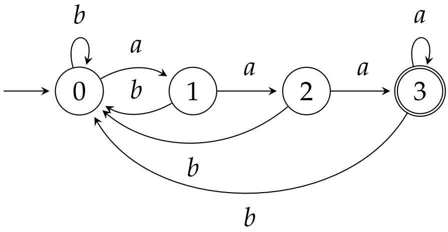
$$G_3 = \langle \{S, T, U\}, \{a, b\}, \{S \to aS \mid bS \mid aT, T \to aU, U \to a\}, S\rangle$$

Non-déterminisme !

$L_3 = \{xaaa \mid x \in \{a, b\}^*\}$



$L_3 = \{xaaa \mid x \in \{a,b\}^*\}$



$$
\begin{aligned}
& (0, abaabaaa) \\
\vdash \quad & (1, baabaaa) \\
\vdash \quad & (0, aabaaa) \\
\vdash \quad & (1, abaaa) \\
\vdash \quad & (2, baaa) \\
\vdash \quad & (0, aaa) \\
\vdash \quad & (1, aa) \\
\vdash \quad & (2, a) \\
\vdash \quad & (3, \varepsilon)
\end{aligned}
$$

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

$$\textcircled{1} \xrightarrow{a,\, B \to c} \textcircled{2}$$

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

$$1 \xrightarrow{a, B \to c} 2$$

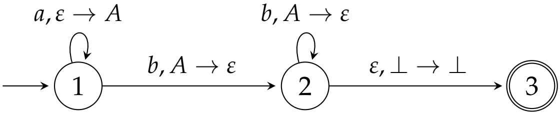
Si l'automate est en 1, et que la tête de lecture est sur a, l'automate :

- 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
- Σ est l'alphabet d'entrée
- Γ est l'alphabet de symboles de pile
  (en particulier, $\perp \in \Gamma$)
- δ est la fonction de transition :

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \to \wp(Q \times \Gamma^*)$$

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

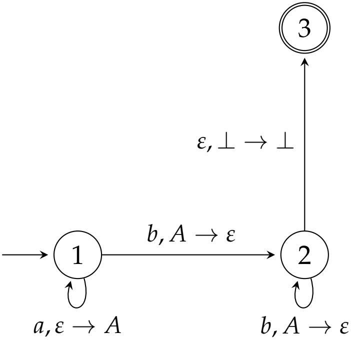
Configurations et mouvement

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

- 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.
  - α 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 : $(q_0, m, \perp)$ où m est le mot à reconnaître
- Configuration d'acceptation : $(q, \varepsilon, \perp)$ avec $q \in F$
- Mouvement : $(q, aw, Z\alpha) \vdash (q', w, \gamma\alpha)$ si $(q', \gamma) \in \delta(q, a, Z)$.

Équivalence

Représentation graphique



Définition formelle

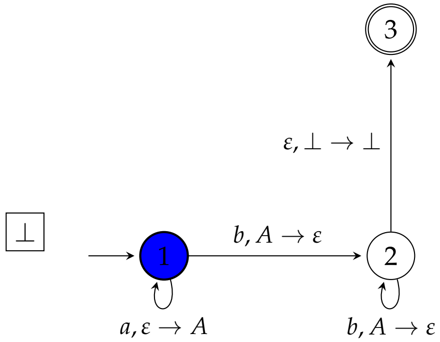
$$\langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$$

$$\begin{array}{rcl} Q & = & \{0, 1, 2, 3\} \\ \Sigma & = & \{a, b\} \\ \Gamma & = & \{A, \perp\} \end{array}$$

$$\begin{array}{rcl} \delta(1, a, \varepsilon) & = & \{(1, A)\} \\ \delta(1, b, A) & = & \{(2, \varepsilon)\} \\ \delta(2, b, A) & = & \{(2, \varepsilon)\} \\ \delta(2, \varepsilon, \perp) & = & \{(3, \perp)\} \end{array}$$

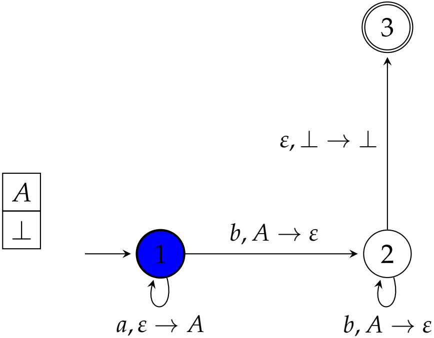
$$\begin{array}{rcl} q_0 & = & 1 \\ F & = & \{3\} \end{array}$$

Reconnaissance



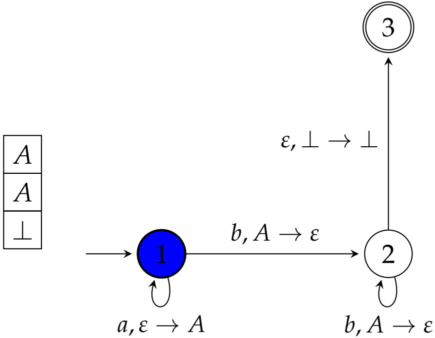
$(1, aaabbb, \perp)$

Reconnaissance



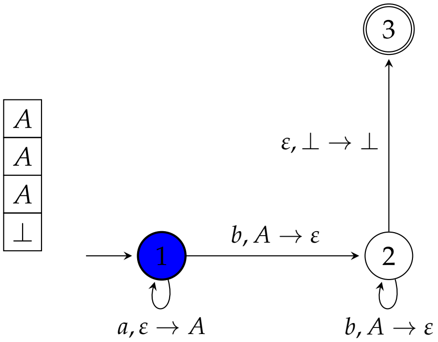
$$
\begin{array}{ll}
 & (1, aaabbb, \perp) \\
\vdash & (1, aabbb, A\perp)
\end{array}
$$

Reconnaissance



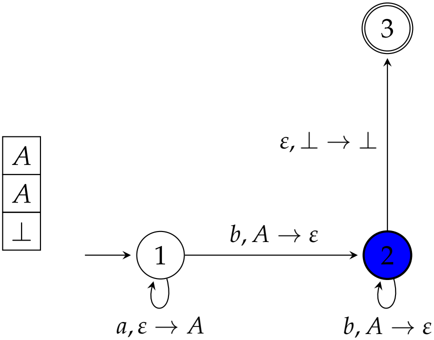
$$\begin{array}{ll} & (1, aaabbb, \perp) \\ \vdash & (1, aabbb, A\perp) \\ \vdash & (1, abbb, AA\perp) \end{array}$$

Reconnaissance



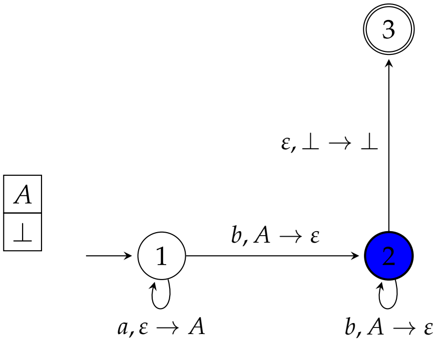
$$
\begin{array}{ll}
 & (1, aaabbb, \perp) \\
\vdash & (1, aabbb, A\perp) \\
\vdash & (1, abbb, AA\perp) \\
\vdash & (1, bbb, AAA\perp)
\end{array}
$$

Reconnaissance



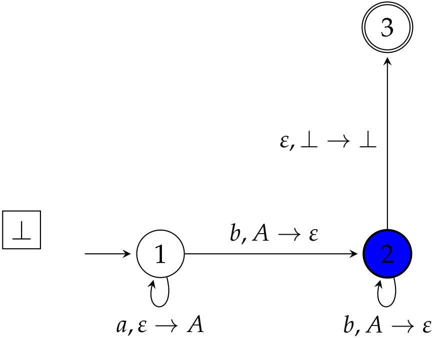
$$
\begin{array}{ll}
 & (1, aaabbb, \perp) \\
\vdash & (1, aabbb, A\perp) \\
\vdash & (1, abbb, AA\perp) \\
\vdash & (1, bbb, AAA\perp) \\
\vdash & (2, bb, AA\perp)
\end{array}
$$

Reconnaissance



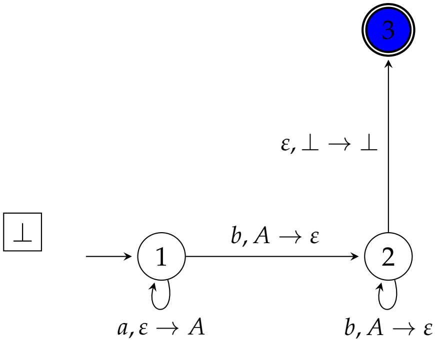
$$
\begin{array}{ll}
 & (1, aaabbb, \perp) \\
\vdash & (1, aabbb, A\perp) \\
\vdash & (1, abbb, AA\perp) \\
\vdash & (1, bbb, AAA\perp) \\
\vdash & (2, bb, AA\perp) \\
\vdash & (2, b, A\perp)
\end{array}
$$

Reconnaissance



$$
\begin{aligned}
& (1, aaabbb, \perp) \\
\vdash\ & (1, aabbb, A\perp) \\
\vdash\ & (1, abbb, AA\perp) \\
\vdash\ & (1, bbb, AAA\perp) \\
\vdash\ & (2, bb, AA\perp) \\
\vdash\ & (2, b, A\perp) \\
\vdash\ & (2, \varepsilon, \perp)
\end{aligned}
$$

Reconnaissance

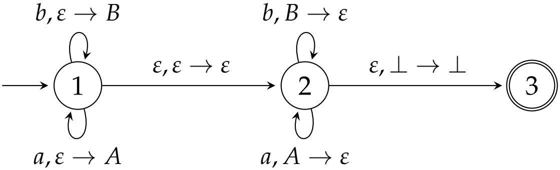


$$
\begin{array}{ll}
 & (1, aaabbb, \perp) \\
\vdash & (1, aabbb, A\perp) \\
\vdash & (1, abbb, AA\perp) \\
\vdash & (1, bbb, AAA\perp) \\
\vdash & (2, bb, AA\perp) \\
\vdash & (2, b, A\perp) \\
\vdash & (2, \varepsilon, \perp) \\
\vdash & (3, \varepsilon, \perp)
\end{array}
$$

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

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

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

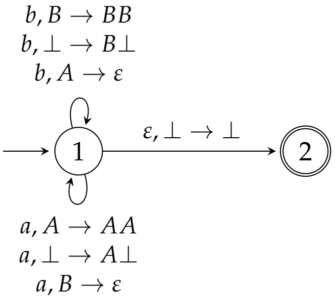


$$G = \langle \{S\}, \{a,b\}, \{S \to aSa \mid bSb \mid \varepsilon\}, S \rangle$$

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

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

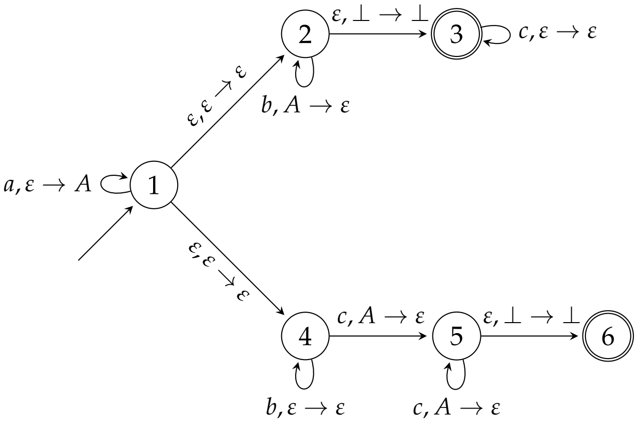
$L = \{m \in \{a,b\}^*, |m|_a = |m|_b\}$



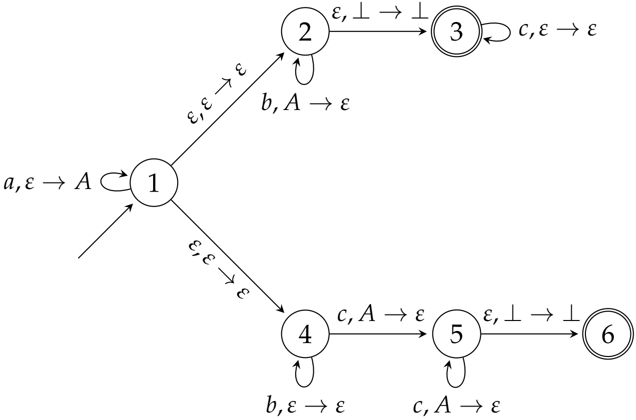
$$G = \langle \{S\}, \{a,b\}, \{S \to aSb \mid bSa \mid SS \mid \varepsilon\}, S\rangle$$

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

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



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

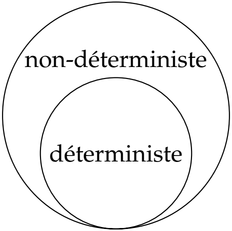


$L = \{a^i b^j c^k \mid i = j \text{ ou } i = k\}$ ne peut être reconnu par un automate à pile déterministe !

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 $m\sharp m$ 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 $\square$ 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

$$1 \xrightarrow{a \to b,\, D} 2$$

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

$$1 \xrightarrow{\;a \to b, D\;} 2$$

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 :

- $a \to G$: 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 $\langle Q, \Sigma, \Gamma, \Box, \delta, q_0, q_A, q_R \rangle$ où :

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

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

- $q_0 \in Q$ 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 : (ε, 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 :

$$(ua, q_i, bv) \vdash (u, q_j, acv) \text{si} (q_j, c, G) \in \delta(q_i, b)$$
$$(ua, q_i, bv) \vdash (uac, q_j, v) \text{si} (q_j, c, D) \in \delta(q_i, b)$$

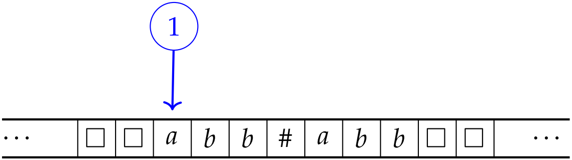
.

Exemple

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

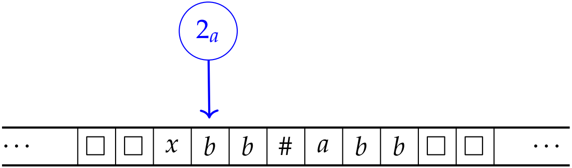
1. Fait des allers-retours entre les deux occurrences de m pour vérifier qu'elles contiennent bien le même symbole. Si ce n'est pas le cas, ou qu'un # supplémentaire est détecté, va dans l'état de rejet. Les symboles sont éliminés au fur et à mesure qu'ils sont vérifiés (par le symbole x à gauche, et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

Exemple — début d'exécution



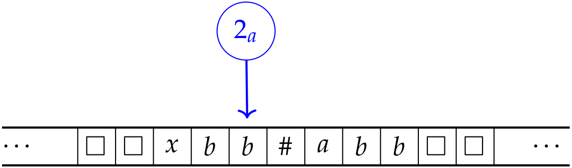
$$\delta(1, a) \quad = \quad (x, 2_a, D)$$ enregistre a

Exemple — début d'exécution



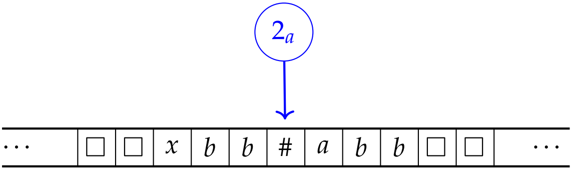
$$
\begin{aligned}
\delta(1, a) &= (x, 2_a, D) && \text{enregistre } a \\
\delta(2_a, \alpha) &= (\alpha, 2_a, D) \text{ pour } \alpha \neq \# && \text{cherche } \#
\end{aligned}
$$

Exemple — début d'exécution



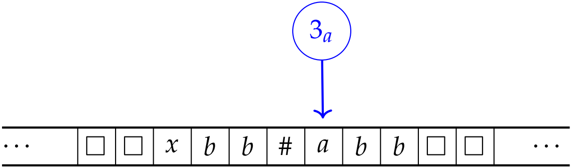
$$\begin{aligned} \delta(1,a) &= (x, 2_a, D) && \text{enregistre } a \\ \delta(2_a,\alpha) &= (\alpha, 2_a, D) \text{ pour } \alpha \neq \# && \text{cherche } \# \end{aligned}$$

Exemple — début d'exécution



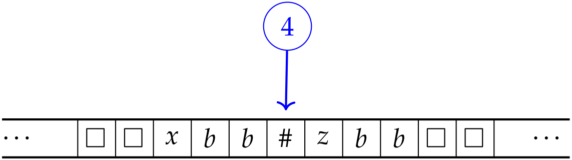
$$
\begin{aligned}
\delta(1, a) &= (x, 2_a, D) && \text{enregistre } a \\
\delta(2_a, \alpha) &= (\alpha, 2_a, D) \text{ pour } \alpha \neq \# && \text{cherche } \# \\
\delta(2_a, \#) &= (\#, 3_a, D) && 3_a \text{ cherche une lettre non-}z
\end{aligned}
$$

Exemple — début d'exécution



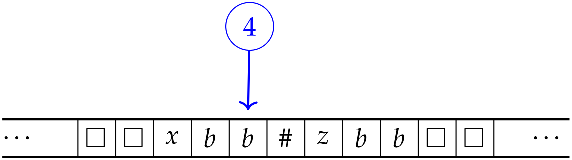
$$
\begin{array}{rcll}
\delta(1,a) & = & (x, 2_a, D) & \text{enregistre } a \\
\delta(2_a,\alpha) & = & (\alpha, 2_a, D) \text{ pour } \alpha \neq \# & \text{cherche } \# \\
\delta(2_a,\#) & = & (\#, 3_a, D) & 3_a \text{ cherche une lettre non-}z \\
\delta(3_a,a) & = & (z, 4, G) & \text{vérifie la lettre}
\end{array}
$$

Exemple — début d'exécution



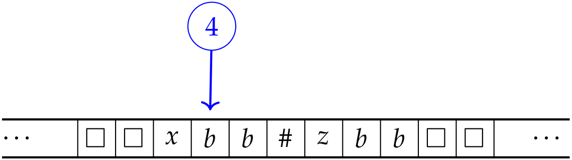
$$
\begin{aligned}
\delta(1,a) &= (x,2_a,D) && \text{enregistre } a\\
\delta(2_a,\alpha) &= (\alpha,2_a,D) \text{ pour } \alpha \neq \# && \text{cherche } \#\\
\delta(2_a,\#) &= (\#,3_a,D) && 3_a \text{ cherche une lettre non-}z\\
\delta(3_a,a) &= (z,4,G) && \text{vérifie la lettre}\\
\delta(4,\alpha)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

Exemple — début d'exécution



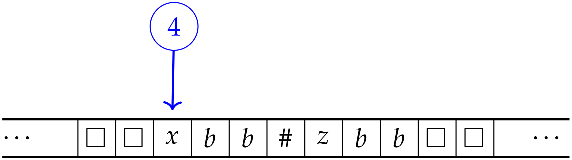
$$
\begin{aligned}
\delta(1,a) &= (x,2_a,D) && \text{enregistre } a\\
\delta(2_a,\alpha) &= (\alpha,2_a,D) \text{ pour } \alpha \neq \# && \text{cherche } \#\\
\delta(2_a,\#) &= (\#,3_a,D) && 3_a \text{ cherche une lettre non-}z\\
\delta(3_a,a) &= (z,4,G) && \text{vérifie la lettre}\\
\delta(4,\alpha)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

Exemple — début d'exécution



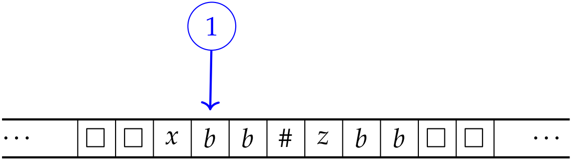
$$
\begin{array}{rcll}
\delta(1,a) & = & (x, 2_a, D) & \text{enregistre } a \\
\delta(2_a,\alpha) & = & (\alpha, 2_a, D) \text{ pour } \alpha \neq \# & \text{cherche } \# \\
\delta(2_a,\#) & = & (\#, 3_a, D) & 3_a \text{ cherche une lettre non-}z \\
\delta(3_a,a) & = & (z, 4, G) & \text{vérifie la lettre} \\
\delta(4,\alpha) & = & (\alpha, 4, G) \text{ pour } \alpha \neq x & \text{revient vers } x
\end{array}
$$

Exemple — début d'exécution



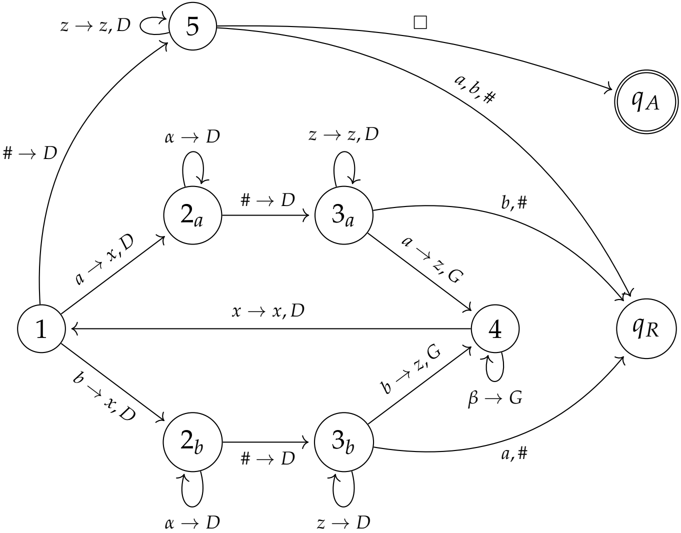
$$
\begin{array}{rcll}
\delta(1,a) & = & (x,2_a,D) & \text{enregistre } a \\
\delta(2_a,\alpha) & = & (\alpha,2_a,D) \text{ pour } \alpha \neq \# & \text{cherche } \# \\
\delta(2_a,\#) & = & (\#,3_a,D) & 3_a \text{ cherche une lettre non-}z \\
\delta(3_a,a) & = & (z,4,G) & \text{vérifie la lettre} \\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

Exemple — début d'exécution



$$
\begin{array}{rcll}
\delta(1,a) & = & (x,2_a,D) & \text{enregistre } a \\
\delta(2_a,\alpha) & = & (\alpha,2_a,D) \text{ pour } \alpha \neq \# & \text{cherche \#} \\
\delta(2_a,\#) & = & (\#,3_a,D) & 3_a \text{ cherche une lettre non-}z \\
\delta(3_a,a) & = & (z,4,G) & \text{vérifie la lettre} \\
\delta(4,\alpha) & = & (\alpha,4,G) \text{ pour } \alpha \neq x & \text{revient vers } x \\
\delta(4,x) & = & (x,1,D) & \text{relance le processus} \\
\delta(1,b) & = & (x,2_b,D)\ldots &
\end{array}
$$

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