# Automates et Langages

## Partie 2

Emmanuelle Grislin

*INSA 3 FISA Informatique*

Partie 2

Mars 2021

# Organisation

Module *Automates et Langages* :

- Partie 1 : Langages réguliers et automates avec S. Piechowiak
- Partie 2 : Autres types de langages avec focus sur langages algébriques et automates à piles
  - volume horaire : 4h30 CM (FISE et FISA)
  - 4h30 TD en FISE - 6h en FISA
  - 12h TP en FISE - 18hTP en FISA
- évaluation :
  - une note de DS reprenant les 2 parties du module
  - et une note TP

# Objectifs de cette partie du module "Automates et Langages"

- Savoirs :
  - *connaissance du vocabulaire utile à la compréhension de la théorie des langages*
  - *connaissance de la hiérarchie de Chomsky*
  - *connaissance du théorème d'équivalence de Kleene*
  - *connaissance de la représentation par automate à pile*

# Objectifs de cette partie du module "Automates et Langages"

- Savoirs :
  - *connaissance du vocabulaire utile à la compréhension de la théorie des langages*
  - *connaissance de la hiérarchie de Chomsky*
  - *connaissance du théorème d'équivalence de Kleene*
  - *connaissance de la représentation par automate à pile*
- Savoir-faire :
  - *donner des dérivations obtenues à partir d'une grammaire, créer un arbre de dérivation*
  - *établir si un mot m appartient à un langage L de type 3 ou 2*
  - *déterminer le type d'un langage au sens de la hiérarchie de Chomsky*
  - *donner le langage reconnu par une grammaire de type 3 ou 2*
  - *donner une grammaire engendrant un langage de type 3 ou 2*
  - *construire un automate à pile reconnaissant un langage algébrique donné*

# Sommaire

# Objectifs du cours 1/4

- Savoirs :
  - *connaissance du vocabulaire utile à la compréhension de la théorie des langages*
- Savoir-faire :
  - *donner des dérivations obtenues à partir d'une grammaire*

# Plan

# Introduction

- langage source → compilation → langage objet
- niveaux d'analyse du texte source :
  - analyse lexicale :
    reconnaître les éléments en entrée et les catégoriser
  - analyse syntaxique :
    vérifier que le code est conforme aux règles de constitution du langage de programmation, détecter les erreurs de syntaxe
  - analyse sémantique :
    donner du "sens", une interprétation

# Rappels : alphabet, mot

## alphabet

ensemble fini non vide dont les éléments sont appelés des lettres ou symboles

## mot

suite finie de lettres ou symboles d'un alphabet donné

- Un mot $u$ constitué de $n$ symboles est dit de **longueur** $n$ et noté : $|u| = n$
- Mot de longueur 0 est le **mot vide**, noté $\epsilon$
- $\Sigma^*$ : ensemble des mots sur l'alphabet $\Sigma$
- $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$ : ensemble de tous les mots non vides sur $\Sigma$
- $\Sigma^i$ : ensemble des mots de longueur $i$ : $\Sigma^* = \bigcup_{i \in \mathbb{N}} \Sigma^i$

# Rappels : langages

## Langage

Un langage $L$ sur un alphabet $\Sigma$ est un sous-ensemble quelconque de $\Sigma^*$ ($L \subseteq \Sigma^*$), un ensemble de mots définis sur le mÍme alphabet.

## Opérations sur les langages

- Opérations ensemblistes :
  - union : $L_1 \cup L_2 = \{u \in \Sigma^* | u \in L_1 \text{ ou } u \in L_2\}$
  - intersection : $L_1 \cap L_2 = \{u \in \Sigma^* | u \in L_1 \text{ et } u \in L_2\}$
  - complémentaire : $\complement L = \{u \in \Sigma^* | u \notin L\}$
  - différence : $L_1 \setminus L_2 = \{u \in \Sigma^* | u \in L_1 \text{ et } u \notin L_2\}$
- Produit ou concaténation :
  $L_1 L_2 = \{u \in \Sigma^* | u = u_1 u_2 \text{ avec } u_1 \in L_1 \text{ et } u_2 \in L_2\}$
- Puissance : $L^0 = \{\epsilon\}$, $L^1 = L$, $L^{n+1} = LL^n$
- Etoile de Kleene ("fermeture") : union de toutes les puissances de L
  $L^* = L^0 \cup L^1 \cup L^2 \ldots = \bigcup_{n \geq 0} L^n$

# Méthodes de définition d'un langage

- en **extension** : énumération de tous les mots du langage (pas tjrs possible)
- par une **expression régulière** (ou expression "rationnelle") : (pas tjrs possible) vu en partie 1 du cours
- par une **propriété** caractéristique (ex. "tous les mots qui commencent par a")
- en utilisant des **paramètres** (ex. $\{a^n\, b^n | n \geq 0\}$)
- par une **grammaire formelle** : cf. suite du cours

# Plan

# Grammaire formelle

## Grammaire

Une **grammaire** est définie par un quadruplet $G =< \Sigma, V, S, R >$ avec

- $\Sigma$ : ensemble fini de **symboles terminaux**, l'alphabet
- $V$ : ensemble fini de **variables** (symboles non terminaux, $\notin \Sigma$)
- $S$ : symbole de $V$ particulier appelé **racine** (ou "axiome")
- $R$ : ensemble fini de **règles** de production (ou "de réécriture")

# Règles et dérivation

- **Règle** :
  - Sur un alphabet $\Sigma$, une **règle de production** est un couple ordonné de deux mots de $\Sigma^*$ séparés par le symbole $\rightarrow$.
    Ex. : $u \rightarrow v$
  - Avec une règle de production $u \rightarrow v$ et un mot contenant $u$, le mot $\omega_1 u \omega_2$ peut Ítre réécrit en $\omega_1 v \omega_2$ : $\omega_1 u \omega_2 \rightarrow \omega_1 v \omega_2$

# Règles et dérivation

- **Règle** :
  - Sur un alphabet $\Sigma$, une **règle de production** est un couple ordonné de deux mots de $\Sigma^*$ séparés par le symbole $\rightarrow$.

    Ex. : $u \rightarrow v$
  - Avec une règle de production $u \rightarrow v$ et un mot contenant $u$, le mot $\omega_1 u \omega_2$ peut Ítre réécrit en $\omega_1 v \omega_2$ : $\omega_1 u \omega_2 \rightarrow \omega_1 v \omega_2$
- **Dérivation** :
  - Le mot $y$ **dérive** en 1 pas du mot $x$, noté $x \rightarrow y$, ssi il existe une règle $u \rightarrow v$ et 2 mots $\omega_1, \omega_2$ tels que $x = \omega_1 u \omega_2$ et $y = \omega_1 v \omega_2$
  - Généralisation avec plusieurs réécritures successives :
    - **Dérivation** en $n$ pas : $x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n$ : $x_1 \xrightarrow{n} x_n$
    - Le mot $y$ **dérive** du mot $x$ : $x \xrightarrow{*} y$

# Langage engendré par une grammaire

## Grammaire

Une **grammaire** est définie par un quadruplet $G =< \Sigma, V, S, R >$ avec

- $\Sigma$ : ensemble fini de **symboles terminaux**
- $V$ : ensemble fini de **variables** (symboles non terminaux, $\notin \Sigma$)
- $S$ : symbole de $V$ particulier appelé **racine** (ou "axiome")
- $R$ : ensemble fini de **règles** de production (ou "de réécriture")

## Langage

**Langage engendré** par la grammaire G =
ensemble des mots de $\Sigma^*$ qui dérivent de la racine de G

$$L(G) = \{x \in \Sigma^* | S \xrightarrow{*} x\}$$

# Langage engendré par une grammaire

## Exemple de grammaire engendrant un langage régulier

- Soit la grammaire définie par $G = < \Sigma, V, S, R >$ avec
  - l'alphabet $\Sigma = \{a, b\}$
  - les variables $V = \{S, T\}$
  - la racine $S$
  - les règles $R = \{S \to aS, S \to bT, T \to bT, T \to \epsilon\}$

# Langage engendré par une grammaire

## Exemple de grammaire engendrant un langage régulier

- Soit la grammaire définie par $G =< \Sigma, V, S, R >$ avec
  - l'alphabet $\Sigma = \{a, b\}$
  - les variables $V = \{S, T\}$
  - la racine $S$
  - les règles $R = \{S \to aS, S \to bT, T \to bT, T \to \epsilon\}$
- Le langage engendré par la grammaire G est l'ensemble des mots de $\Sigma^*$ qui dérivent de la racine $S$ : $L(G) = \{x \in \Sigma^* | S \xrightarrow{*} x\}$
  - $S \to aS \to aaS \to aaaS \ldots \to a...abT \to a...abbT \ldots \to a...ab...b$
  - ou : $S \to bT \to bbT \ldots \to b...b$
  - ou $S \to bT \to b$

## Exemple de grammaire engendrant un langage régulier

- Soit la grammaire définie par $G =< \Sigma, V, S, R >$ avec
  - l'alphabet $\Sigma = \{a, b\}$
  - les variables $V = \{S, T\}$
  - la racine $S$
  - les règles $R = \{S \to aS, S \to bT, T \to bT, T \to \epsilon\}$
- Le langage engendré par la grammaire G est l'ensemble des mots de $\Sigma^*$ qui dérivent de la racine $S$ : $L(G) = \{x \in \Sigma^* | S \overset{*}{\to} x\}$
  - $S \to aS \to aaS \to aaaS \ldots \to a...abT \to a...abbT \ldots \to a...ab...b$
  - ou : $S \to bT \to bbT \ldots \to b...b$
  - ou $S \to bT \to b$
- Description par une expression régulière : $L(G) = a^* bb^*$

# Langage et grammaire

- Chaque grammaire engendre un langage et un seul
- Mais un langage peut Ítre engendré par une infinité de grammaires

## Grammaires équivalentes

Deux grammaires $G_1$ et $G_2$ sont équivalentes ssi $L(G_1) = L(G_2)$.
On note $G_1 \sim G_2$

# Langage et grammaire

- Chaque grammaire engendre un langage et un seul
- Mais un langage peut Ítre engendré par une infinité de grammaires

## Grammaires équivalentes

Deux grammaires $G_1$ et $G_2$ sont équivalentes ssi $L(G_1) = L(G_2)$.
On note $G_1 \sim G_2$

## Ex.

- $G_2 =< \{a, b\}, \{S, T, U, V\}, S,$
  $\{S \rightarrow aT|U, T \rightarrow Ta|U, U \rightarrow bU|V, V \rightarrow b\} >$

# Langage et grammaire

- Chaque grammaire engendre un langage et un seul
- Mais un langage peut Ítre engendré par une infinité de grammaires

## Grammaires équivalentes

Deux grammaires $G_1$ et $G_2$ sont équivalentes ssi $L(G_1) = L(G_2)$.
On note $G_1 \sim G_2$

## Ex.

- $G_2 =< \{a, b\}, \{S, T, U, V\}, S,$
  $\{S \to aT|U, T \to Ta|U, U \to bU|V, V \to b\} >$
- $L(G_2) = a^*b^*b = a^*bb^*$

# Langage et grammaire

- Chaque grammaire engendre un langage et un seul
- Mais un langage peut Ítre engendré par une infinité de grammaires

## Grammaires équivalentes

Deux grammaires $G_1$ et $G_2$ sont équivalentes ssi $L(G_1) = L(G_2)$.
On note $G_1 \sim G_2$

## Ex.

- $G_2 =< \{a, b\}, \{S, T, U, V\}, S,$
  $\{S \to aT|U, T \to Ta|U, U \to bU|V, V \to b\} >$
- $L(G_2) = a^*b^*b = a^*bb^*$
- $G_2 \sim G_1 =< \{a, b\}, \{S, T\}, S, \{S \to aS|bT, T \to bT|\epsilon\} >$

# Dérivation

Application d'une règle de grammaire sur un mot nécessite choix :

- de la variable (symbole non terminal) à réduire
- de la règle à appliquer

## Dérivation gauche *(resp. droite)*

Dérivation s'appliquant uniquement à la variable la plus à gauche *(resp. à droite)* du mot :

- dérivation gauche : $uA\gamma \rightarrow_G ua\gamma$
- dérivation droite : $\gamma Au \rightarrow_D \gamma au$

oˇ $u \in \Sigma^*, A \rightarrow a \in R$ at $\gamma \in (\Sigma \cup V)^*$

# Arbre de dérivation

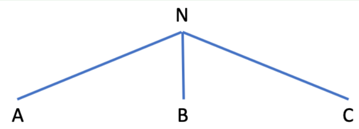
L'analyse syntaxique permet de créer un arbre de dérivation :

## Arbre de dérivation

Un arbre de dérivation d'une grammaire $G =< \Sigma, V, S, R >$ est un arbre ordonné tel que :

- racine étiquetée par la racine $S$ de $G$
- feuilles étiquetées sur $\Sigma \cup \{\epsilon\}$ (symboles terminaux)
- nœuds internes étiquetés sur $V$ (symboles non terminaux)
- $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ : liste ordonnée des étiquettes des nœuds fils du nœud N ssi $N \rightarrow \alpha_1\, \alpha_2\, \ldots\, \alpha_n$ est une règle de $R$

Ex. : Si $N \rightarrow ABC \in R$, on représente graphiquement la dérivation par :



# Arbre de dérivation

## Arbre de dérivation

- mot obtenu par dérivation = concaténation des étiquettes des feuilles, lues de gauche à droite
- des suites de dérivation différentes peuvent Ítre associées à un mÍme arbre
- à toute arbre de dérivation correspond une unique dérivation à gauche (resp. à droite) = parcours *en profondeur d'abord* de l'arbre

# Cours 1 : objectifs atteints ?

- Savoirs :
  - *connaissance du vocabulaire utile à la compréhension de la théorie des langages :*
    - *langage ? grammaire ? différence entre les 2 ?*
    - *grammaires équivalentes ?*
    - *règle de production ? dérivation ? dérivation gauche  droite ?*
- Savoir-faire :
  - *donner des dérivations obtenues à partir d'une grammaire ? créer un arbre de dérivation ?*

# Plan

# Plan

# Plan