IN 406 – Théorie des Langages
Cours 7 : Grammaires et Dérivations

Loric Duhaze – loric.duhaze@uvsq.fr
Franck Quessette – franck.quessette@uvsq.fr

Université de Versailles – Saint-Quentin

V3 2019–2020

Rappels

Notions déjà vues :

- lettre, alphabet **fini** ;
- mot, langage **fini** ou **infini** ;
- langage rationnel (opérations ensemblistes) ;
- automate **fini** non-déterminsite (AFN), automate **fini** déterministe ;
- langage reconnaissable par automate **fini** ;
- expression régulière.
- langage reconnaissable par une expression régulière.
- automates à pile

Grammaire

Définition

Une **grammaire** $\mathcal{G} = (\Sigma, V, S, \mathcal{P})$ telle que :

- Σ un alphabet fini de symboles **terminaux** ;
- V un alphabet fini de symboles **non terminaux** (aussi appelés **variables**), $V \cap \Sigma = \emptyset$;
- $S \in V$ appelé **axiome** ;
- $\mathcal{P} \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$ l'ensemble des **rêgles de production**.

Dans toute la suite de ce cours :

- les **terminaux** sont en **minuscules** : a, b, c, ...
- les **non terminaux** sont en **majuscules** : A, B, S, X, Y, ...

Règles de production

Une règle de production d'une grammaire $\mathcal{G} = (\Sigma, V, S, \mathcal{P})$ s'écrit sous la forme : $\alpha \to \beta$ avec $\alpha, \beta \in (V \cup \Sigma)^*$.

- α est appelé la partie gauche de la règle
- β est appelé la partie droite de la règle

La forme de ces règles de production définit le type de la grammaire dans la hiérarchie de Chomski (présentée dans les slides suivants). Plus les restrictions sur ces règles sont importantes et plus la grammaire a un type élevé.

Hiérarchie de Chomski : Type 3

Grammaires dont les règles de production sont de la forme (a terminal et X et Y non terminaux) :

- La partie gauche est un et un seul **non terminal**.
- La forme des règle est soit :
  1. $X \to \varepsilon$, $X \to a$, $X \to aY$: grammaire régulière **droite**
  2. $X \to \varepsilon$, $X \to a$, $X \to Ya$: grammaire régulière **gauche**

On peut aussi autoriser plusieurs terminaux consécutifs au lieu d'un seul, par exemple $X \to abaY$.

Mots avec un nombre pair de a

$\Sigma = \{a\}$, $V = \{S\}$

- Régulière gauche : $\mathcal{P} = \{S \to Sa, S \to \varepsilon\}$
- Régulière droite : $\mathcal{P} = \{S \to aS, S \to \varepsilon\}$

Ce type de grammaire correspond aux langages **réguliers** qui sont reconnus par automates.

Hiérarchie de Chomski : Type 2

Grammaires dont les règles de production sont de la forme :

- La partie gauche est un et un seul **non terminal**.
- La forme des règle est : $X \to \alpha$ avec $X \in V$, $\alpha \in (\Sigma \cup V)^*$

Exemple : Grammaire Algébrique

$\Sigma = \{a, b\}$, $V = \{S\}$ et $\mathcal{P} = \{$
$S \to aSb$,
$S \to \varepsilon\}$

Langage des mots de la forme $a^n b^n$.

- Ce type de grammaire correspond aux langages **algébriques**
- Les mots de ces langages peuvent être reconnus par des **automates à pile**

Hiérarchie de Chomski : Type 1

Grammaires dont les règles de production sont de la forme :

- $\alpha X \beta \to \alpha \gamma \beta$, α et β sont appelés le contexte.
- Avec $X \in V$, $\alpha, \beta, \gamma \in (\Sigma \cup V)^*$

Exemple : Grammaire Contextuelle

$\Sigma = \{a, b, c\}$, $V = \{S, X\}$ et
$\mathcal{P} = \{ \quad S \to abc, \quad S \to aSXc, \quad cX \to Xc, \quad bX \to bb \quad \}$

$S \to aSXc \to aaSXcXc \to aaabcXcXc \to aaabXccXc \to$
$aaabXcXcc \to aaabXXccc \to aaabbXccc \to aaabbbccc$

Langage des mots de la forme $a^n b^n c^n$ pour $n > 0$.

- Ce type de grammaire correspond aux langages **contextuels**
- Les mots de ces langages peuvent être reconnus par des **machines de Turing bornées**

Hiérarchie de Chomski : Type 0

Grammaires dont les règles de production sont de la forme :

- $\alpha \rightarrow \beta$
- Avec $\alpha, \beta \in (\Sigma \cup V)^*$

Exemple : Grammaire Générale

Une grammaire générale engendre un langage récursivement énumérable. Un langage récursivement énumérable est un langage dont les mots correspondent à l'ensemble des mots acceptés par une machine de Turing. Il n'y a pas d'exemple simple de Grammaires de ce type.

- Ce type de grammaire correspond aux langages **récursivement enumérables**
- Les mots de ces langages peuvent être reconnus par des **machines de Turing**

Hiérarchie de Chomski

Soit une grammaire $\mathcal{G} = (\Sigma, V, S, \mathcal{P})$:

Type	Langage	Grammaire	Machine
3	rationnel, régulier ou reconnaissable	régulière droite $X \to a, X \to aY, X \to \varepsilon$ $X, Y \in V$, $a \in \Sigma$ (régulière gauche)	automate fini
2	algébrique ou hors-contexte	algébrique ou hors-contexte $X \to \alpha$ $X \in V$, $\alpha \in (\Sigma \cup V)^*$	automate à pile
1	contextuel	contextuelle $\alpha X \beta \to \alpha \gamma \beta$ $X \in V$, $\alpha, \beta, \gamma \in (\Sigma \cup V)^*$	machine de Turing bornée
0	récursivement énumérable	générales $\alpha \to \beta$ $\alpha, \beta \in (\Sigma \cup V)^*$	machine de Turing

Dérivation

Différence entre un automate et une grammaire

- Un automate **reconnaît** les mots d'un langage
- Une grammaire **génère** (les mots d') un langage

Mais, il est possible de savoir si un mot appartient au langage généré par une grammaire.

Dérive : On dit qu'un mot xAy se dérive en $x\gamma y$, si et seulement si $A \rightarrow \gamma$ est une règle de production de G avec $\gamma \in (\Sigma \cup V)$.

Mot dérivé : Soient u et v deux mots sur $\Sigma \cup V$. Le mot v se dérive de u par la grammaire algébrique G = (Σ,V, P, S) s'il existe des mots $w_0, w_1, ..., w_n$ tels que $u = w_0$, $v = w_n$ et pour tout entier i tq $(0 \leq i < n)$, w_{i+1} se dérive directement de w_i.

Dérivation

Pour montrer qu'un mot appartient au langage généré par la grammaire, il montrer qu'il s'obtient à partir des règles de production de la grammaire (en commençant par l'axiome). Néanmoins, ce problème n'est pas toujours simple, il peut y avoir des ambiguïtés dans les dérivations.

Exemple

Soit la grammaire suivante : $\Sigma = \{a, b\}$, $V = \{S\}$ et $\mathcal{P} = \{$

$S \to aSbS$,

$S \to a$,

$S \to b$,

$S \to \varepsilon$ $\}$

Le mot **aabb** est généré par la grammaire (i.e. appartient au langage généré par la grammaire)

$$S \to aSbS \to aabS \to aabb$$

Arbre de dérivation

Un **arbre de dérivation** (pour une grammaire de type 2 ou 3) est un arbre ordonné tel que :

- La racine est le symbole de départ
- Les feuilles sont étiquetées par une lettre terminale ou ε.
- Les sommets internes sont étiquetés par une lettre non terminale.
- Les fils d'un sommet interne étiqueté X sont (dans cet ordre) $x_1, ..., x_n$ avec $x_i \in (\Sigma \cup V)$, si et seulement si il existe une règle de production $X \to x_1 ... x_n$.

Arbre de Dérivation

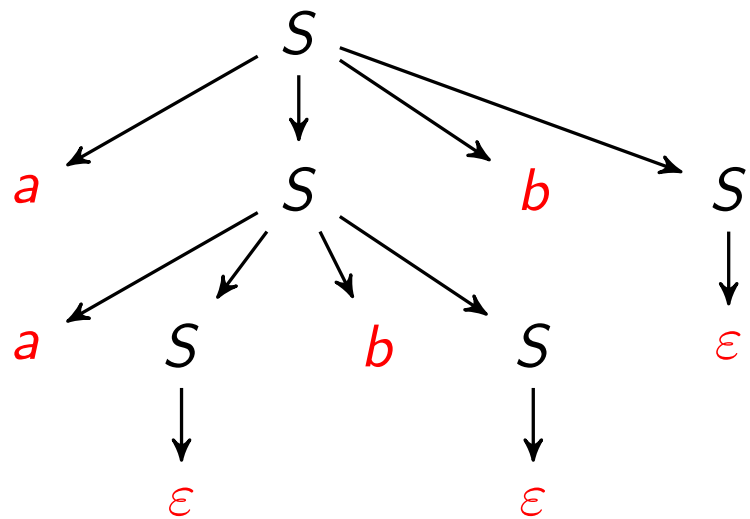


Figure: Arbre de dérivation de aabb

Le mot ainsi généré se lit en lisant les feuilles de gauche à droite.

Ambiguïté

Définition

Une grammaire est **ambiguë** si et seulement s'il existe un mot qui admet deux arbres de dérivation distincts.

- Un langage est ambigu si **toute** grammaire qui l'engendre est ambiguë.
- Bien que le problème de déterminer si une grammaire est ambiguë soit *indécidable*, exhiber deux arbres de dérivations distincts pour un même mot suffit.

Ambiguïté

Comment lever l'ambiguïté ?

- On peut donner une règle pour faire un choix, comme remplacer toujours en priorité la lettre non-terminale la plus à gauche.

if a then s if b then s1 else s2

- Sans parenthésage, la séquence d'instruction ci-dessus est ambiguë. En effet, on ne sait pas à quel *if* se rapporte le *else*. Seule une règle implicite inclue dans le compilateur C nous permet de dire que le *else* se rapporte au dernier *if* rencontré.

Ambiguïté = Deux chemins pour un mot ?

Il peut exister deux chemins pour reconnaître le même mot dans un automate tout comme il peut exister deux arbres de dérivations pour un même mot dans la grammaire

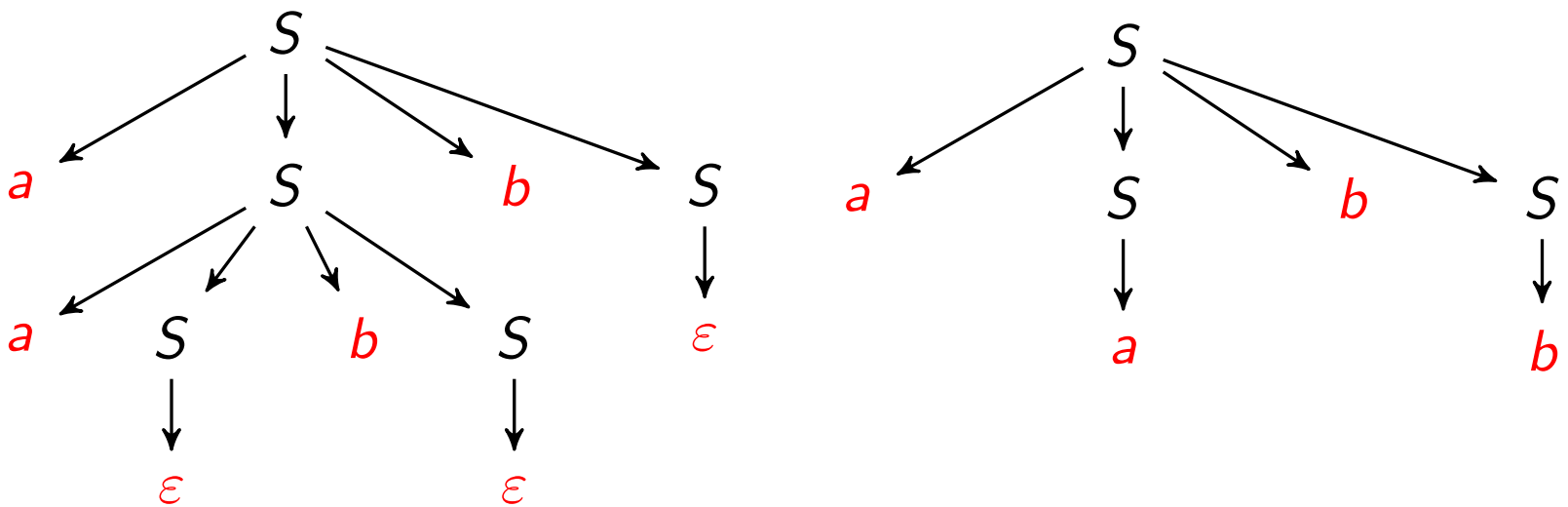
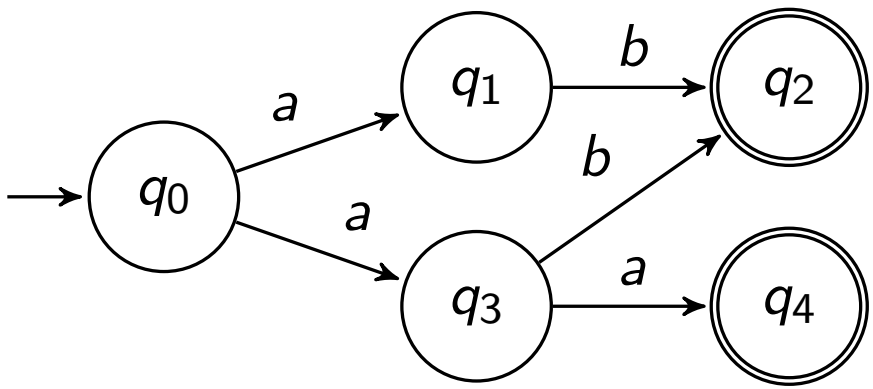


Figure: Deux arbres de dérivations différents pour le mot aabb.

Ambiguïté = Deux transitions identiques ?

Dans un automate, depuis un sommet v, il peut exister deux transitions portant la même lettre, pour reconnaître un mot on doit donc regarder une ou plusieurs transitions en avance pour savoir laquelle emprunter.

Reconnaître le mot **aa** dans cet automate nécessite de regarder des chemins de longueur 2 depuis q_0, En effet, avec des chemins de longueur 1 on risque de ne pas prendre la bonne transition.



Grammaire LL(k)

Il en va de même dans les grammaires, on peut avoir besoin de regarder plusieurs dérivations en avance pour générer le bon mot.

C'est le cas dans la grammaire suivante (LL(2)) : $\Sigma = \{a, b\}$, $V = \{S, A, B\}$ et $\mathcal{P} = \{$

$S \to A$,
$S \to B$,
$A \to a$,
$B \to b$ $\}$

Définition

Une grammaire G est dite **LL(k)** si et seulement si **pour tout** mot w tel que au cours de la dérivation, pour *choisir la bonne règle de production*, il faut regarder en avance **au plus** les k règles successives que l'on va appliquer ensuite pour générer le mot w.

Formes Normales

Pour chaque langage : Il existe une infinité de grammaires.
Décider si deux grammaires génèrent le même langage **pour les grammaire hors-contexte** : mise sous forme normale (canonique).

Ci-dessous a est un terminal quelconque, S est le non terminal de départ, X, Y, Y_i, Z sont des non terminaux quelconques.

Forme normale de Greibach Toutes les règles sont de la forme :
$Y_0 \to aY_1Y_2...Y_n$

Forme normale de Chomsky Toutes les règles sont de la forme :
$X \to YZ$
$X \to a$
$S \to \varepsilon$