

## 6. Grammaires formelles

*Enseignant: Arnaud Casteigts*

*Assistant: Alexandre-Quentin Berger*

Nous entamons une partie du cours à cheval entre la linguistique et l'informatique, à savoir l'étude des grammaires (formelles). Ces dernières permettent de décrire des langages de manière générative, par l'utilisation de *règles de production*. Les grammaires sont plus puissantes que les expressions régulières et les automates finis, elles permettent notamment de représenter des langages non réguliers (et bien plus).

### 6.1 Grammaires formelles (en général)

Intuitivement, une grammaire formelle est un ensemble de règles qui permettent de créer des motifs textuels par remplacement successifs. Voici un exemple de grammaire (appelons-la  $G_1$ ) comportant deux règles de production :

$$S \rightarrow aSb$$

$$S \rightarrow \varepsilon$$

Ces règles signifient que lorsqu'on rencontre le symbole  $S$ , on peut le remplacer par  $aSb$  (règle 1) ou par un mot vide (règle 2). Chaque règle est de la forme  $\alpha \rightarrow \beta$  où  $\alpha$  est le motif à remplacer (appelé **partie gauche** de la règle) et  $\beta$  est le motif qui va remplacer  $\alpha$  (appelé **partie droite**). Il y a deux types de symboles : des **symboles terminaux** (ici  $a$  et  $b$ ), qui correspondent aux symboles habituels des alphabets que nous manipulons, et des **symboles non-terminaux** (ou **variables**), généralement représentés en majuscule, qui ont vocation à être remplacés (ici  $S$  seulement, mais il pourrait y en avoir d'autres).

On part généralement d'un symbole non-terminal seul, puis on applique des règles jusqu'à ce qu'il n'y ait que des symboles terminaux. On a alors **génééré** (ou **engendré**) un mot du langage. Par exemple, si nous partons de  $S$  et utilisons la seconde règle directement, cela produit le mot vide  $\varepsilon$  et nous avons terminé car il n'y a plus de variable à remplacer. Si nous utilisons plutôt la première règle, cela produit le mot intermédiaire  $aSb$ . Nous pouvons alors utiliser la seconde règle, qui remplace le  $S$  du milieu par  $\varepsilon$ , donnant le mot  $ab$ , ou continuer à utiliser la première règle, ce qui donne  $aaSbb$ , et ainsi de suite. Au final, si on explore récursivement tous les choix possibles, on peut produire les mots  $\varepsilon, ab, aabb, aaabbb, \dots$ , autrement dit, le langage  $L(G_1) = \{a^n b^n \mid n \in \mathbb{N}\}$ , qui nous le savons, est un langage non régulier ! (c.f. cours précédent).

Dans leur forme la plus générale, les grammaires formelles peuvent utiliser des règles de productions qui transforment n'importe quel type de motif (partie gauche ayant des symboles

terminaux et/ou non-terminaux) en n'importe quel autre motif (idem pour le membre droit). Ces grammaires sont extrêmement puissantes, on se restreint donc souvent à des versions plus limitées qui sont plus faciles à étudier et déjà très utiles. Dans ce cours (et le suivant), nous étudions deux familles de grammaires, à savoir les grammaires *hors-contexte* et les grammaires *régulières*, la seconde étant un cas particulier de la première.

## 6.2 Grammaires hors-contexte

Les grammaires **hors-contexte** (aussi appelées grammaires non contextuelles ou grammaires algébriques) imposent que la partie gauche de chaque règle consiste en un symbole non-terminal seulement (et rien d'autre), comme dans l'exemple donné plus haut. On les appelle *hors-contexte* parce qu'elles ne tiennent pas compte de ce qu'il y a autour de la variable à remplacer.

Plus formellement, une grammaire hors-contexte est définie par :

- Un alphabet  $V$  de symboles non terminaux (variables),
- Un alphabet  $\Sigma$  de symboles terminaux,
- Un symbole de départ dans  $V$ ,
- Un ensemble fini de règles de production  $\mathcal{P}$  de la forme  $\alpha \rightarrow \beta$ , où  $\alpha \in V$  et  $\beta \in (V \cup \Sigma)^*$ .

Par exemple, la grammaire  $G_1$  donnée plus haut correspond à  $V = \{S\}$ ,  $\Sigma = \{a, b\}$ , le symbole de départ  $S \in V$  et les règles de productions  $\mathcal{P} = \{S \rightarrow aSb, S \rightarrow \varepsilon\}$ .

Lorsque plusieurs règles ont la même partie gauche, on peut les regrouper en utilisant une barre verticale, comme  $S \rightarrow aSb \mid \varepsilon$ , qui se lit “ $S$  donne  $aSb$  ou  $\varepsilon$ ”. (Il s'agit quand même de deux règles différentes.) Et pour être encore plus compact, on écrit parfois directement :  $G_1 = (\{S\}, \{a, b\}, S, \{S \rightarrow aSb \mid \varepsilon\})$ .

L'application d'une ou plusieurs règles d'une grammaire  $G$  est appelé une **dérivation** de  $G$ . Voici une dérivation possible de la grammaire  $G_1$  :

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbbb.$$

Notez la différence entre les symboles  $\rightarrow$  (spécification de la règle, dans la définition de la grammaire) et  $\Rightarrow$  (son application sur un mot). Nous appellerons **mot intermédiaire** les mots qui contiennent au moins une variable, et **mot terminal** ceux qui n'ont que des symboles terminaux. S'il existe une dérivation (arbitrairement longue, et potentiellement de longueur zéro) d'un mot  $w_1$  vers un mot  $w_2$  (terminal ou non), on note  $w_1 \xRightarrow{*} w_2$  et on dit qu'on peut atteindre (ou obtenir)  $w_2$  depuis  $w_1$ . L'ensemble des mots terminaux que l'on peut atteindre depuis le symbole de départ définit le **langage engendré** par la grammaire. Plus formellement, le langage  $L(G)$  engendré par une grammaire  $G$  avec symbole de départ  $S$  est :

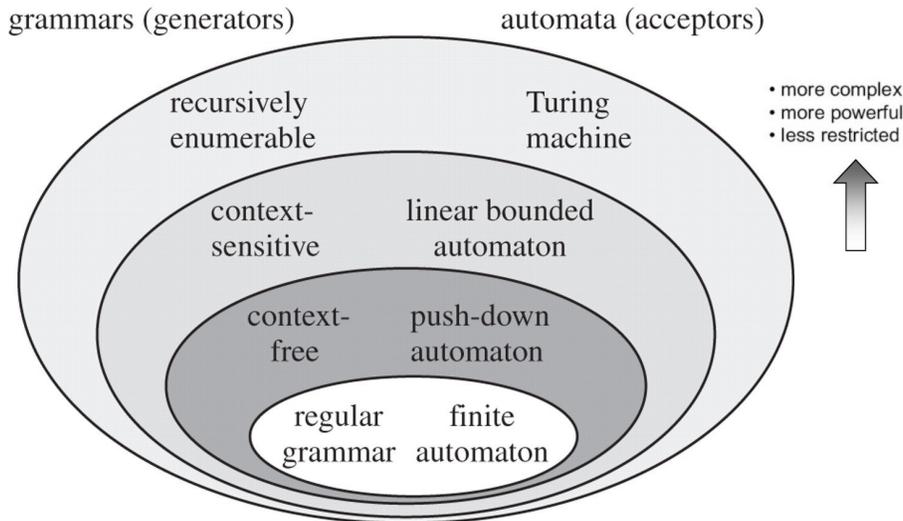
$$L(G) = \{w \in \Sigma^* \mid S \xRightarrow{*} w\}$$

Un langage est **hors-contexte** (LHC) s'il existe une grammaire hors-contexte (GHC) qui l'engendre.

### 6.3 Grammaire régulière

Les grammaires **régulières** sont un cas particulier de grammaires hors-contexte, qui imposent une restriction supplémentaire sur la forme des règles. La partie gauche n'est pas affectée (on a toujours  $\alpha \in V$ ), mais la partie droite  $\beta$  est plus restreinte. En fait, on distingue deux types de grammaires régulières : les grammaires régulières à gauche et les grammaires régulières à droite (qui ont la même puissance). Les **grammaires régulières à gauche** imposent que pour toute règle  $\alpha \rightarrow \beta$ , la partie droite  $\beta$  doit être de la forme  $Xa$ , ou  $a$ , ou  $\varepsilon$ , avec  $X \in V$  et  $a \in \Sigma$ . Pour les **grammaires régulières à droite**, c'est l'inverse :  $\beta$  est de la forme  $aX$ , ou  $a$ , ou  $\varepsilon$ . Autrement dit, chaque règle ne peut générer qu'une variable à la fois et au plus un symbole terminal, qui sera toujours *du même côté*.

**Spoiler :** Nous montrerons la semaine prochaine que les grammaires régulières correspondent exactement aux langages régulier (d'où leur nom). Les grammaires hors-contextes représentent une famille de langage plus grande et correspondent à un modèle de machine plus puissant : les automates à pile. En fait, la hiérarchie de Chomsky met en relation quatre types de grammaires différentes avec quatre types de machines.



## 6.4 Exemples et autres notions

Voici quelques exemples de grammaires hors contexte (dont certaines sont régulières). Nous en profitons pour illustrer quelques notions supplémentaires, comme les *arbres de dérivation*, les *dérivations gauches* (ou *droite*), et les *grammaires ambiguës*.

### Mots se terminant par 0

On peut représenter les mots binaires se terminant par zéro par une grammaire  $G_2 = \{\{S\}, \{0, 1\}, S, \{S \rightarrow 0S \mid 1S \mid 0\}\}$

Exemple de dérivation :  $S \Rightarrow 1S \Rightarrow 10S \Rightarrow 101S \Rightarrow 1010$

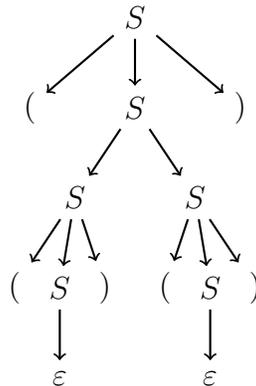
Il s'agit d'une grammaire régulière à droite, et en effet, nous savons que ce langage est régulier, d'ailleurs, il correspond à l'expression régulière  $(0 \cup 1)^*0$ .

### Mots bien parenthésés

On peut représenter l'ensemble des mots bien parenthésés par une grammaire  $G_3 = (V = \{S\}, \Sigma = \{(, )\}, S, \mathcal{P} = \{S \rightarrow (S) \mid SS \mid \varepsilon\})$ . Cette grammaire produit tous les mots faits de parenthèses correctement entrelacées, p.ex.  $()$ ,  $(())$ ,  $()()$ ,  $((())$ ,  $((()))$ ,  $((())()$ ,  $\dots$ . Elle ne produit pas, en revanche, les mots tels que  $($ , ou  $)$ , ou encore  $((()$ .

Exemple de dérivation :  $S \Rightarrow (S) \Rightarrow (SS) \Rightarrow ((S)S) \Rightarrow (()S) \Rightarrow (()S) \Rightarrow (()())$

On peut représenter la dérivation d'un mot par un **arbre de dérivation**, comme suit :



En fait, cet arbre représente *plusieurs* dérivations possibles, mais qui sont toutes équivalentes. Par exemple, dans la dérivation précédente, nous avons systématiquement utilisé la variable la plus à gauche pour dériver. Une telle dérivation est appelée **dérivation gauche**. Elle correspond à un parcours en profondeur de la gauche vers la droite dans l'**arbre de dérivation**. Mais nous aurions aussi pu effectuer une **dérivation droite**, correspondant à un parcours en profondeur de la droite vers la gauche comme suit :

$S \Rightarrow (S) \Rightarrow (SS) \Rightarrow (S(S)) \Rightarrow (S()) \Rightarrow ((S)()) \Rightarrow (()())$

Ou encore prendre les variables dans un ordre arbitraire.

### Expressions arithmétiques

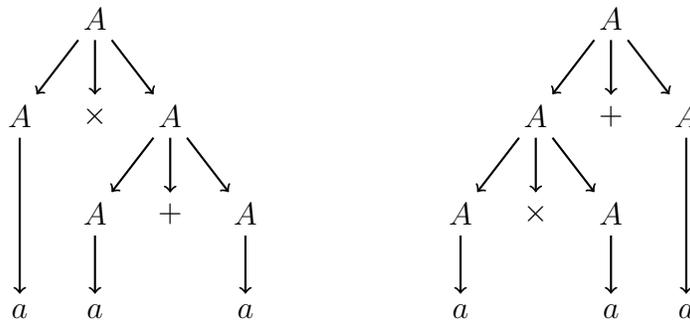
Voici un exemple de règles qui produisent des expressions arithmétiques avec les opérations  $+$  et  $\times$  pour une variable  $a$ , l'alphabet terminal étant  $\Sigma = \{\times, +, a, (, )\}$  :

$A \rightarrow A + A \mid A \times A \mid (A) \mid a$

Par exemple, voici une dérivation gauche qui produit le mot  $a \times (a + a)$  :

$A \Rightarrow A \times A \Rightarrow a \times A \Rightarrow a \times (A) \Rightarrow a \times (A + A) \Rightarrow a \times (a + A) \Rightarrow a \times (a + a)$ .

Comme précédemment, nous aurions pu aussi utiliser une dérivation droite, ces dérivations revenant in fine au même arbre de dérivation. Cependant, il est cette fois possible qu'un même mot admette *deux arbres* de dérivations différents ! Prenons l'exemple de  $a \times a + a$  (sans les parenthèses). On peut l'obtenir via les arbres :



Si cela peut se produire pour au moins un mot, alors on dit que ce mot est ambigu et que la grammaire elle-même est une **grammaire ambiguë**. En général, on essaie d'éviter cela, par exemple, dans les langages de programmation, il est préférable qu'il n'y ait qu'une façon possible d'analyser syntaxiquement (= de "parser") une instruction.

Le langage humain est naturellement ambigu, p.ex : "j'ai vu sa main avec un miroir" peut signifier "j'ai vu [sa main avec un miroir]" ou "j'ai vu [sa main] avec un miroir".

Dans certains cas, une grammaire peut être modifiée pour être rendue non-ambiguë, mais ce n'est pas toujours possible. Ici, c'est possible, la grammaire suivante engendre le même langage, sans ambiguïté (au prix d'utiliser trois variables au lieu d'une) :

$E \rightarrow E + T \mid T$   
 $T \rightarrow T \times F \mid F$   
 $F \rightarrow (E) \mid a$

## Langage humain

On pourrait utiliser des variables du genre  $V = \{\text{PHRASE, GROUPE}\text{NOMINAL, GROUPE}\text{VERBAL, COMPLEMENT, NOM, VERBE}\}$  et des terminaux  $\Sigma = \{\text{a...z, } \_ \}$  avec des règles du type :

PHRASE  $\rightarrow$  GROUPE $\text{NOMINAL}$  GROUPE $\text{VERBAL}$

GROUPE $\text{VERBAL}$   $\rightarrow$  VERBE COMPLEMENT

VERBE  $\rightarrow$  manger, dormir, ...

...

Ce type de grammaires est utilisée en linguistique pour modéliser les langues naturelles. Elles ont longtemps servi de base au traitement automatique des langues (pour la traduction, la rédaction, etc.). Elles ont récemment été supplantées par les méthodes statistiques (réseaux de neurones, notamment), mais peuvent continuer à être utilisées en conjonction avec ces dernières.

## 6.5 Équivalence entre grammaires régulières et langages réguliers

Les grammaires régulières correspondent exactement aux langages réguliers. Cela peut se démontrer (comme souvent) par des transformations. En l'occurrence, n'importe quel AFD peut être converti en grammaire régulière et n'importe quelle grammaire régulière peut être convertie en AFN. Et puisque les AFDs et les AFNs correspondent tous deux aux langages réguliers, on a bien l'équivalence souhaitée.

### 6.5.1 Transformation d'un AFD vers une grammaire régulière

Soit un AFD  $A = (Q, \Sigma, \delta, q_0, F)$ . On peut construire une grammaire régulière à droite  $G = (V, \Sigma, S, \mathcal{P})$  qui reconnaît exactement  $L(A)$ , en la définissant comme suit :

- Les variables correspondent aux états de l'automate ( $V = Q$ ),
- Les symboles terminaux correspondent à l'alphabet de l'automate,
- Le symbole de départ  $S$  correspond à l'état initial  $q_0$ ,
- Pour chaque transition  $(q_i, a, q_j)$ , on crée une règle  $q_i \rightarrow aq_j$ ,
- Pour chaque état final  $q \in F$ , on crée une règle  $q \rightarrow \varepsilon$ .

Il existe une construction similaire pour transformer l'automate  $A$  en une grammaire régulière à gauche.

### 6.5.2 Transformation d'une grammaire régulière vers un AFN

Soit une grammaire régulière à droite  $G = (V, \Sigma, S, \mathcal{P})$ , on peut construire un AFN  $A = (Q, \Sigma, \delta, q_0, F)$  qui reconnaît exactement  $L(G)$ , en le définissant comme suit :

- Les états de l'automate correspondent aux variables  $V$  + un état final  $q_f$ ,
- L'alphabet de l'automate correspond aux symboles terminaux de la grammaire,
- Pour chaque règle de la forme  $A \rightarrow aB$ , on ajoute une transition depuis l'état correspondant à  $A$  vers l'état correspondant à  $B$  avec le symbole  $a$ ,
- Pour chaque règle de la forme  $A \rightarrow a$ , on ajoute une transition depuis l'état correspondant à  $A$  vers  $q_f$  avec le symbole  $a$ ,
- Pour chaque règle de la forme  $A \rightarrow \varepsilon$ , on ajoute une  $\varepsilon$ -transition depuis l'état correspondant à  $A$  vers  $q_f$ .

Il existe une construction similaire pour transformer une grammaire régulière à gauche vers un AFN.