Langages formels (11X003) - Automne 2026

6. Grammaires formelles

Enseignant: Arnaud Casteigts

Assistants: E. Arm & M. Marseloo
Moniteurs: A. Hugener & O. Oternaud

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* qui produisent des mots en remplaçant des motifs successivement à partir d'un symbole de départ S.

6.1 Exemple simple

Voici un exemple de grammaire comportant deux règles de production :

Grammaire G_1 :

symbole de départ : S
règle 1 : $S \to \mathtt{a}S\mathtt{b}$
règle 2 : $S \to \varepsilon$

Hormis le mot vide ε, on voit deux types de symboles ici : des **symboles terminaux** et des **variables** (aussi appelés symboles non-terminaux). Les symboles terminaux correspondent aux symboles de l'alphabet Σ sur lequel les mots sont générés, par exemple ici $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. Les variables comme S, dénotées en majuscule, font partie d'un autre alphabet V. Elles ont vocation à être remplacées au fur et à mesure, la génération se terminant lorsqu'il n'y a plus de variable. On a alors un mot sur Σ qui a été généré (ou engendré) par la grammaire.

Il se peut qu'il y ait plusieurs choix possible. Par exemple, en partant de S, nous avons le choix d'appliquer la première ou la seconde règle. Si nous utilisons la seconde règle directement, cela produit le mot vide ε et nous avons terminé car il n'y a plus de variable à remplacer. Si nous utilisons la première règle à la place, cela produit le mot intermédiaire $\mathtt{a}S\mathtt{b}$. Nous pouvons alors recommencer en remplaçant le S de ce mot. Si on utilise ici la seconde règle, qui remplace le S du milieu par ε, cela donnera le mot $\mathtt{a}\varepsilon\mathtt{b} = \mathtt{ab}$ et nous aurons terminé. Si on continue à utiliser la première règle, cela donnera le mot $\mathtt{aa}\,S\mathtt{bb}$ et on pourra continuer. Quel est l'ensemble de tous les mots que l'on peut obtenir ici ? On se convaincra assez facilement qu'il s'agit des mots $\varepsilon, ab, aabb, aaabbb, \ldots$, autrement dit le langage $L(G_1) = \{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}\}$. Ce langage n'est pas régulier, les grammaires sont donc potentiellement plus expressives que les expressions régulières.

6.2 Definition

Une grammaire formelle est définie par un 4-tuple :

- Un alphabet V de variables (symboles non terminaux),
- Un alphabet Σ de symboles terminaux,
- Un symbole de départ dans V,
- Un ensemble fini de règles de production $\mathcal{P}$ de la forme $\alpha \to \beta$.

Pour chaque règle, α est appelée la partie gauche et β la partie droite. Il existe différents types de grammaires selon les contraintes supplémentaires qu'on impose à α et β :

Grammaire **générale** : $\alpha \in (V \cup \Sigma)^*$ et $\beta \in (V \cup \Sigma)^*$. Aucune restriction : une règle peut remplacer n'importe quel type de motif (mélant un nombre arbitraire de variables et de symboles terminaux) par n'importe quel autre type de motif.

Grammaire **contextuelle** : Idem, mais seules les variables peuvent être modifiées, les autres symboles de la partie gauche ne servant que de contexte déclencheur.

Grammaire **hors-contexte** : $\alpha \in V$ et $\beta \in (V \cup \Sigma)^*$. Ici les parties gauches sont constituées d'une seule variable et rien d'autre. Les parties droites ne sont pas restreintes.

Grammaire **régulière** : Idem, mais on restreint aussi la partie droite (voir plus loin).

Aujourd'hui, nous nous concentrerons principalement sur les grammaires **hors-contexte** qui décrivent une famille de langages importante. Elles sont par ailleurs très utilisées en théorie des langages de programmation et dans les protocoles réseaux (pour la compilation et le parsing, notamment). On les appelle hors-contexte (ou non-contextuelles) parce qu'elles ne tiennent pas compte de ce qu'il y a autour de la variable à remplacer.

6.3 Concepts de base

Revenons à notre grammaire G_1, qui correspond au 4-tuple

$$G_1 = (\{S\}, \{\mathtt{a}, \mathtt{b}\}, S, \{S \to \mathtt{a}S\mathtt{b}; S \to \varepsilon\}).$$

Lorsque plusieurs règles ont la même partie gauche, on peut simplifier la notation en combinant les parties droites avec une barre verticale, comme $S \to \mathtt{a}S\mathtt{b} \mid \varepsilon$, qui se lit "$S$ donne $\mathtt{a}S\mathtt{b}$ ou ε". (Il s'agit quand même de deux règles différentes.)

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

$$S \overset{r_1}{\Longrightarrow} \mathtt{a}S\mathtt{b} \overset{r_1}{\Longrightarrow} \mathtt{aa}S\mathtt{bb} \overset{r_1}{\Longrightarrow} \mathtt{aaa}S\mathtt{bbb} \overset{r_2}{\Longrightarrow} \mathtt{aaabbb}.$$

Notez la différence entre les symboles $\to$ (définition de la règle) et $\Rightarrow$ (son application). C'est une convention. Par ailleurs, l'information de la règle utilisée sur le symbole $\Rightarrow$ est facultative (nous l'avons ajoutée par souci pédagogique).

Un mot est **intermédiaire** s'il contient au moins une variable, il est **terminal** sinon. S'il existe une dérivation d'un mot w_1 vers un mot w_2 (terminal ou non), on note $w_1 \overset{*}{\Rightarrow} w_2$ et on dit qu'on peut dériver w_2 depuis w_1. L'ensemble des mots terminaux que l'on peut dériver depuis le symbole de départ définit le **langage engendré** par la grammaire. Autrement dit, le langage engendré par une grammaire G avec symbole de départ S est définit comme :

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

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

6.4 Exemples et autres notions

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

6.4.1 Mots binaires se terminant par `0` (nombres pairs)

On peut représenter le langage des mots binaires se terminant par `0` par une grammaire hors-contexte $G_2 = (V = \{S\}, \Sigma = \{\mathtt{0}, \mathtt{1}\}, S, \mathcal{P} = \{S \to \mathtt{0}S \mid \mathtt{1}S \mid \mathtt{0}\})$

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

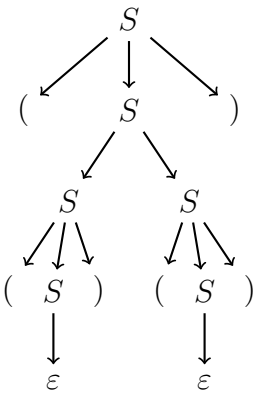
Remarquez que $L(G_2)$ est le langage régulier correspondant à l'expression $(\mathtt{0} \cup \mathtt{1})^*\mathtt{0}$. La grammaire G_2 est en effet régulière (cas particulier de grammaire hors-contexte).

6.4.2 Mots bien parenthésés

On peut représenter l'ensemble des mots bien parenthésés par une grammaire $G_3 = (V = \{S\}, \Sigma = \{(,)\}, S, \mathcal{P} = \{S \to (S) \mid SS \mid \varepsilon\}\}$. Cette grammaire produit tous les mots constitués de parenthèses correctement entrelacées, p.ex. $(), (()), ()(), (()()), (())(), \ldots$ 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**. Mais nous aurions aussi pu effectuer une **dérivation droite** :

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

Ou encore prendre les variables dans un ordre arbitraire... Cependant, bien que la dérivation du mot $(()())$ n'est pas unique, l'arbre de dérivation, lui, est unique ici. Est-ce toujours le cas ?

Notez que $L(G_3)$ n'est pas un langage régulier. Vous le démontrerez en exercices en utilisant le lemme de l'étoile. En revanche, c'est bien évidemment un langage hors-contexte, puisque la grammaire qui le génère est hors-contexte.

6.4.3 Expressions arithmétiques

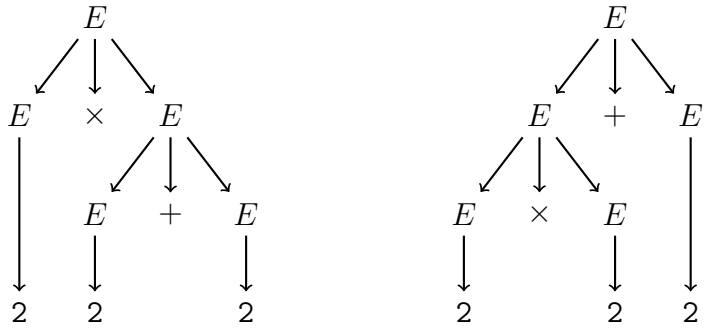
Voici un exemple de grammaire G_4 qui engendre des expressions arithmétiques avec les opérations $+$ et $\times$ utilisant le symbole $\mathsf{2}$. On utilise ici $V = \{E\}$, $\Sigma = \{\times, +, \mathsf{2}, (,)\}$, le symbole de départ E et les règles :

$$E \to E + E \mid E \times E \mid (E) \mid \mathsf{2} \qquad \text{(que l'on appelera } r_1, r_2, r_3, r_4\text{)}$$

Par exemple, voici une dérivation qui produit le mot $\mathsf{2} \times (\mathsf{2} + \mathsf{2})$:

$$E \overset{r_2}{\Longrightarrow} E \times E \overset{r_4}{\Longrightarrow} \mathsf{2} \times E \overset{r_3}{\Longrightarrow} \mathsf{2} \times (E) \overset{r_1}{\Longrightarrow} \mathsf{2} \times (E + E) \overset{r_4}{\Longrightarrow} \mathsf{2} \times (\mathsf{2} + E) \overset{r_4}{\Longrightarrow} \mathsf{2} \times (\mathsf{2} + \mathsf{2}).$$

Notez qu'il s'agit d'une dérivation gauche (nous avons à chaque fois remplacé la variable la plus à gauche), 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 $\mathsf{2} \times \mathsf{2} + \mathsf{2}$. On peut l'obtenir via les arbres :



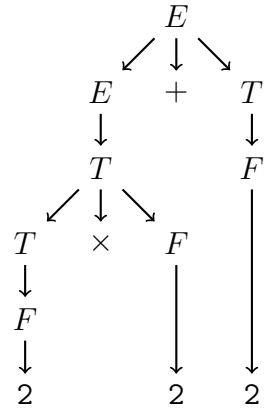
Si un mot peut avoir plusieurs arbres de dérivations, on dit que ce mot est **ambigu** et par extension, la grammaire elle-même est **ambiguë**. En général, on essaie d'éviter cela, par exemple, dans les langages de programmation, il est préférable qu'il n'y ait qu'une façon possible d'analyser syntaxiquement un programme.

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

Ici, on pourrait changer un peu la grammaire de sorte à forcer des parenthèses pour l'addition lorsqu'elle vient du facteur d'une multiplication :

$E \to E + T \mid T$
$T \to T \times F \mid F$
$F \to (E) \mid \texttt{2}$

L'expression $2 \times 2 + 2$ n'a maintenant qu'un seul arbre :



6.4.4 Langage humain ?

On pourrait utiliser des variables $V = \{$Phrase, GroupeNominal, GroupeVerbal, Complement, Nom, Verbe$\}$ et des terminaux $\Sigma = \{\texttt{a...z}, \texttt{␣}\}$ avec des règles du type :

Phrase → GroupeNominal GroupeVerbal
GroupeVerbal → Verbe Complement
Verbe → `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 Grammaires régulières (deuxième semaine)

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 β est plus restreinte. En fait, on distingue deux types de grammaires régulières : les grammaires régulières à gauche et les grammaires régulières à droite (les deux ayant la même expressivité). Les **grammaires régulières à gauche** imposent que pour toute règle $\alpha \to \beta$, la partie droite β doit être de la forme $X\mathtt{s}$, ou bien $\mathtt{s}$, ou bien ε, avec $X \in V$ et $\mathtt{s} \in \Sigma$. Pour les **grammaires régulières à droite**, c'est l'inverse : β est de la forme $\mathtt{s}X$, ou $\mathtt{s}$, ou ε. Autrement dit, chaque règle ne peut générer qu'au plus une variable et au plus un symbole terminal, qui sera toujours *du même côté*.

Par exemple, la grammaire G_2 (déjà vue plus haut) qui génère les mots binaires se terminant par `0` est une grammaire régulière. Ses trois règles sont $S \to \mathtt{0}S \mid \mathtt{1}S \mid \mathtt{0}$, ce qui respecte bien les contrainte d'une grammaire régulière (en l'occurrence, à droite). On aurait aussi pu générer ce langage avec une grammaire régulière à gauche, en utilisant deux variables et quatre règles comme suit : $G_2' = (\{S, T\}, \{\mathtt{0}, \mathtt{1}\}, S, \{S \to T\mathtt{0}, T \to T\mathtt{0} \mid T\mathtt{1} \mid \varepsilon\})$.

6.5.1 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 donc en AFD, par déterminisation).

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 génère 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, s, q_j), on crée une règle $q_i \to sq_j$,
- Pour chaque état final $q \in F$, on créé une règle $q \to \varepsilon$.

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

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 reconnait exactement $L(G)$, en le définissant comme suit :

- Les états de l'automate correspondent aux variables V + un état final q_f,
- L'état initial q_0 correspond à la variable de départ S,
- L'alphabet de l'automate correspond aux symboles terminaux de la grammaire,
- Pour chaque règle de la forme $A \to sB$, on ajoute une transition depuis l'état correspondant à A vers l'état correspondant à B avec le symbole s,
- Pour chaque règle de la forme $A \to s$, on ajoute une transition depuis l'état correspondant à A vers q_f avec le symbole s,
- Pour chaque règle de la forme $A \to \varepsilon$, on ajoute une ε-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.

6.6 Hiérarchie de Chomsky (aperçu)

La hiérarchie de Chomsky met en relation quatre types de grammaires différentes avec quatre types de machines capable de connaître les langages correspondants. Traduction : context-free = hors-contexte ; push-down automaton = automate à pile.

