Langages formels (11X003) - Automne 2024

6. Grammaires formelles

Enseignant: Arnaud Casteigts

Assistants: A.-Q. Berger & M. De Francesco
Monitrices: L. Heiniger & A. Tekkoyun

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) ayant pour symbole initial S et comportant deux règles de production :

$S \to \texttt{a}S\texttt{b}$
$S \to \varepsilon$

Ces règles signifient que lorsqu'on rencontre le symbole S, on peut le remplacer par $\texttt{a}S\texttt{b}$ (règle 1) ou par un mot vide (règle 2). Chaque règle est de la forme $\alpha \to \beta$ où α est le motif à remplacer (appelé **partie gauche** de la règle) et β est le motif qui va remplacer α (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 **généré** (ou **engendré**) un mot du langage. Par exemple, si nous partons de S et 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 plutôt la première règle, cela produit le mot intermédiaire $\texttt{a}S\texttt{b}$. Nous pouvons alors utiliser la seconde règle, qui remplace le S du milieu par ε, donnant le mot `ab`, ou continuer à utiliser la première règle, ce qui donne $\texttt{aa}\,S\texttt{bb}$, et ainsi de suite. Quel est l'ensemble de tous les mots que l'on peut obtenir ? On se convaincra assez facilement qu'il s'agit des mots $\varepsilon, ab, aabb, aaabbb, \ldots$, 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 générale, les grammaires formelles peuvent utiliser des règles de pro-

ductions 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 très expressives, on se restreint donc souvent à des versions plus limitées qui sont plus faciles à étudier et offrent des garanties quant aux modèles de machines pouvant reconnaître les langages qu'elles génèrent. Dans ce cours (et le suivant), nous étudions deux types de grammaires : 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), c'est le cas de 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 Σ de symboles terminaux,
- Un symbole de départ dans V,
- Un ensemble fini de règles de production $\mathcal{P}$ de la forme $\alpha \to \beta$, où $\alpha \in V$ et $\beta \in (V \cup \Sigma)^*$.

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

Lorsque plusieurs règles ont la même partie gauche, on peut les regrouper en utilisant 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.) Et pour être encore plus compact, on écrit parfois directement : $G_1 = (\{S\}, \{\mathtt{a}, \mathtt{b}\}, S, \{S \to \mathtt{a}S\mathtt{b} \mid \varepsilon\})$.

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 :

$$S \Rightarrow \mathtt{a}S\mathtt{b} \Rightarrow \mathtt{aa}S\mathtt{bb} \Rightarrow \mathtt{aaa}S\mathtt{bbb} \Rightarrow \mathtt{aaabbb}.$$

Notez la différence entre les symboles $\to$ (spécification de la règle, dans la définition de la grammaire) et $\Rightarrow$ (son application sur un mot). Nous appelerons **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 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 $L(G)$ engendré par une grammaire G avec symbole de départ S est :

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

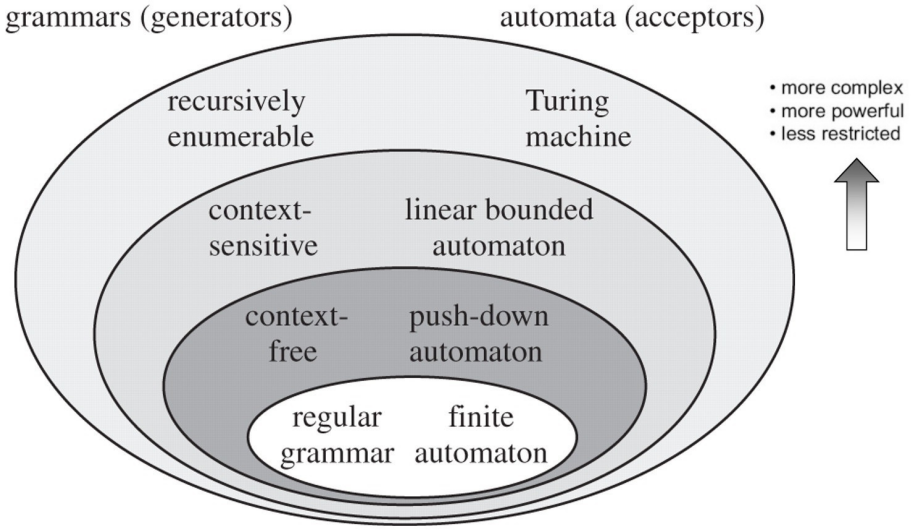
Un langage est **hors-contexte** (LHC) s'il existe une grammaire hors-contexte (GHC) qui l'engendre.

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 β 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 $\alpha \to \beta$, la partie droite β doit être de la forme Xa, ou a, ou ε, avec $X \in V$ et $a \in \Sigma$. Pour les **grammaires régulières à droite**, c'est l'inverse : β est de la forme aX, ou a, 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*.

6.4 Hiérarchie de Chomsky (aperçu)

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 capable de connaître les langages correspondants.



6.5 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 régulière à droite $G_2 = (\{S\}, \{\texttt{0}, \texttt{1}\}, S, \{S \to \texttt{0}S \mid \texttt{1}S \mid \texttt{0}\})$

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

On aurait aussi pu générer ce langage en utilisant une grammaire régulière à gauche, mais cela aurait nécessité l'utilisation de deux variables plutôt qu'une et quatre règles plutôt que trois. En l'occurrence, $G_2' = (\{S, T\}, \{\texttt{0}, \texttt{1}\}, S, \{S \to T\texttt{0}, T \to T\texttt{0} \mid T\texttt{1} \mid \varepsilon\})$.

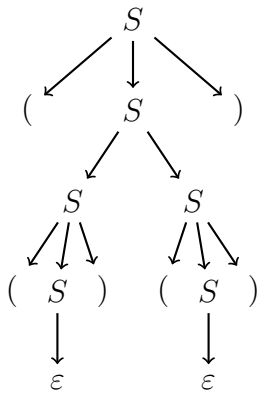
Le fait qu'on puisse générer ce langage avec une grammaire régulière implique que ce langage est régulier, et d'ailleurs, comme nous le savons déjà, il correspond à l'expression régulière $(0 \cup 1)^*0$.

Mots bien parenthésés

On peut représenter l'ensemble des mots bien parenthésés par une grammaire $G_3 = (V = \{S\}, \Sigma = \{(,)\}, S, \mathcal{P} = \{S \to (S) \mid SS \mid \varepsilon\}\}$. Cette grammaire produit tous les mots faits 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**. 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

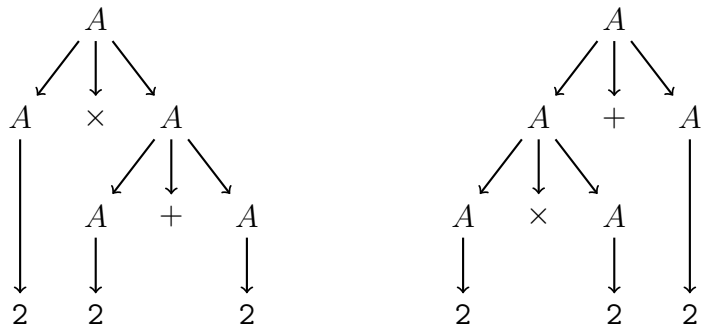
Voici un exemple de règles qui produisent des expressions arithmétiques avec les opérations $+$ et $\times$ à partir du nombre $\mathsf{2}$, l'alphabet terminal étant $\Sigma = \{\times, +, \mathsf{2}, (,)\}$:

$A \to A + A \mid A \times A \mid (A) \mid \mathsf{2}$

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

$A \Rightarrow A \times A \Rightarrow \mathsf{2} \times A \Rightarrow \mathsf{2} \times (A) \Rightarrow \mathsf{2} \times (A + A) \Rightarrow \mathsf{2} \times (\mathsf{2} + A) \Rightarrow \mathsf{2} \times (\mathsf{2} + \mathsf{2}).$

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 $\mathsf{2} \times \mathsf{2} + \mathsf{2}$ (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 ambigüe**. 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 mirroir" peut signifier "j'ai vu [sa main avec un mirroir]" ou "[j'ai vu sa main] avec un mirroir".

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

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

Langage humain

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

Phrase $\to$ GroupeNominal GroupeVerbal
GroupeVerbal $\to$ Verbe Complement
Verbe $\to$ `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.

6.6 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.

6.6.1 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 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, a, q_j), on crée une règle $q_i \to aq_j$,
- 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.

6.6.2 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 reconnait 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 $A \to aB$, 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 $A \to a$, on ajoute une transition depuis l'état correspondant à A vers q_f avec le symbole a,
- Pour chaque règle de la forme $A \to \varepsilon$, 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.