Chapitre 10 : Grammaires non contextuelles

1 Introduction

À ce stade de l'année scolaire nous connaissons six grandes familles de langages (aussi appelées **classes de langages**) représentables en machine : les langages finis, les langages locaux, les langages réguliers, les langages de la classe P, ceux de la classe NP, et enfin les langages décidables. Les inclusions entre ces classes de langages sont résumées sur le diagramme de la figure 1, entre autre on sait que tout langage local est régulier mais pas réciproquement, que tout langage régulier est dans P, mais pas réciproquement, que tout langage de la P est décidable mais pas réciproquement.

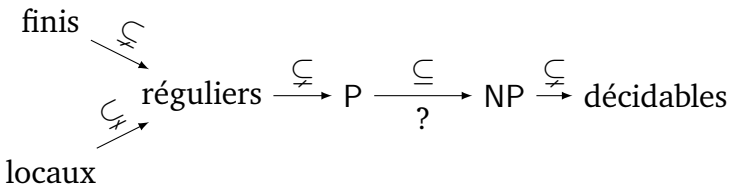


FIGURE 1 – Hiérarchie des classes de langages déjà vues en cours

On l'a dit, ces familles de langages sont représentables en machine, on peut alors écrire des algorithmes manipulant des langages (par exemple pour décider si un mot leur appartient, pour décider s'ils sont vides, pour énumérer les mots de taille bornée leur appartenant, ….)

- Un langage fini peut être représenté par l'ensemble fini de ses mots.
- Un langage local peut être représenté par un automate local (ou par la donnée de ses quatre ensembles caractéristiques Λ, P, S et F qui sont tous finis).
- Un langage régulier peut être représenté par un automate ou par une expression régulière.
- Un langage de P peut être représenté par une fonction indicatrice OCAML de complexité polynomiale.
- Un langage de NP peut être représenté par la donnée de son ensemble de certificats (lui-même dans P) et une fonction OCAML résolvant le problème de vérification associé.
- Un langage décidable peut être représenté par une fonction indicatrice OCAML.

Tandis qu'on gagne en expressivité, on perd en "efficacité" algorithmique. Par exemple pour tester la vacuité d'un langage, l'algorithme mis en place♣ dépend de la classe de langages à traiter :

- pour un langage fini représenté par l'ensemble de ses mots, il suffit d'une comparaison à l'ensemble vide ;
- pour un langage régulier représenté par un automate, on procède à un parcours de graphe ;
- pour un langage décidable représenté par sa fonction indicatrice OCAML tester la vacuité est indécidable.

Exercice de cours 1.1

Afin de justifier de certaines relations de non-inclusion entre les classes de langages susmentionnées, donner :

- un langage local infini ;
- un langage fini non local ;
- un langage régulier infini ;
- un langage régulier non local ;
- un langage de P non régulier.

♣. s'il en existe un

Les langages de programmation. Étant donné un langage de programmation, l'ensemble des chaînes de caractères qui sont des programmes valides de ce langage de programmation forme un langage (au sens ensemble de mots). Par exemple le mot `"let x = 3 in (2 + (3 * x))"` est un mot du langage des programmes OCaml valides. Le mot `"let x = 3 in (2 + (3 * x)"` n'en est pas un. Afin de pouvoir détecter les erreurs de syntaxe comme l'oubli d'une parenthèse fermante ci-avant, il faut être en mesure de répondre algorithmiquement à la question "$w \in$ OCaml ?" où OCaml est l'ensemble des mots composants le langage OCaml.

Il est clair que le langage des expressions OCaml valides est un langage décidable : c'est le langage des codes qu'un compilateur OCaml accepte de compiler. Cependant, le langage des expressions OCaml (Exercice de cours 1.3) n'est pas un langage régulier, aussi dans ce chapitre on s'intéresse à décrire une nouvelle famille de langages, strictement intermédiaire entre les langages réguliers et les langages décidables ♣ qui permet, entre autres, la description des langages de programmation.

Exercice de cours 1.2

Démontrer que le langage des mots bien parenthésés n'est pas un langage régulier.

Exercice de cours 1.3

Déduire de l'exercice de cours précédent que le langage des expressions OCaml valides n'est pas régulier.

On peut remarquer que le langage des mots bien parenthésés, bien que non régulier, admet une définition inductive très simple. En effet en notant B cet ensemble, B est alors le plus petit langage vérifiant :

- $\varepsilon \in \mathsf{B}$;
- si $u \in \mathsf{B}$ et $v \in \mathsf{B}$ alors $u \cdot v \in \mathsf{B}$;
- si $u \in \mathsf{B}$ alors $(u) \in \mathsf{B}$.

Cette remarque nous pousse à définir la classe des langages "admettant une définition inductive très simple". L'étude de cette classe de langages est l'objet de ce chapitre.

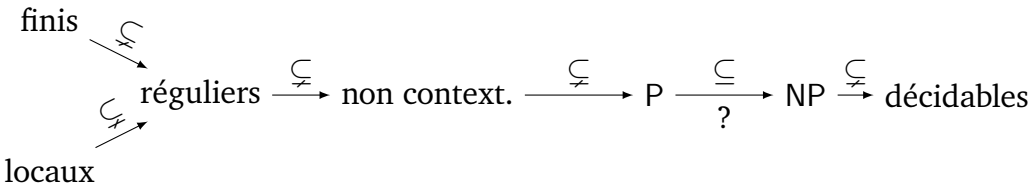


Figure 2 – Hiérarchie des classes de langages au programme

2 Définitions, vocabulaire, propriétés

2.1 Grammaires non contextuelles

Exemple 2.1
Pour l'exemple du langage des mots bien parenthésés, les mots sont définis sur l'alphabet $\Sigma = \{), (\}$, mais afin de décrire ce langage nous avons besoin d'un symbole supplémentaire, disons B. Un tel symbole utilisé pour la description du langage mais pas dans les mots du langage est parfois appelé une variable, c'est pourquoi on note $\mathcal{V} = \{B\}$ dans cet exemple.

♣. On pourra montrer en exercice que cette classe est même strictement incluse dans la classe P.

Définition 2.2

*Soit Σ et $\mathcal{V}$ deux alphabets disjoints. On appelle **règle de production** sur $(\Sigma, \mathcal{V})$ la donnée de deux éléments :*
- *un symbole non terminal de $V \in \mathcal{V}$;*
- *un mot constitué de symboles terminaux ou non terminaux $w_1 w_2 \dots w_n \in (\mathcal{V} \cup \Sigma)^\star$.*

On typographie une telle règle de production $V \to w_1 w_2 \dots w_n$.

Les règles de production sont utilisées pour décrire les différents cas de la définition inductive du langage. La règle $V \to w_1 w_2 \dots w_n$ signifie que la variable V peut être remplacée par $w_1 w_2 \dots w_n$. On remarque qu'une telle règle autorise le remplacement de la variable V indépendamment de ce qui l'entoure♣, c'est pourquoi on parle de grammaire **non contextuelle** dans la suite.

Exemple 2.3

Les trois règles de production permettant de décrire le langage des mots bien parenthésés sont les suivantes.

- $B \to \varepsilon$
- $B \to BB$
- $B \to (B)$

Le membre droit de la première règle est le mot vide. Celui de la seconde est le mot constitué de deux occurrences la variable B. Finalement celui de la dernière règle est constitué de deux symboles terminaux (et) et de la variable B.

Définition 2.4

*Une **grammaire non contextuelle** est un quadruplet $(\Sigma, \mathcal{V}, P, S)$ où :*
- *Σ est un alphabet ;*
- *$\mathcal{V}$ est un alphabet disjoint de Σ ;*
- *P un ensemble fini de règles de production sur Σ et $\mathcal{V}$;*
- *$S \in \mathcal{V}$ appelé le **symbole initial**.*

Vocabulaire 2.5

*Les éléments de Σ sont appelés **symboles terminaux** et ceux de $\mathcal{V}$ symboles **non terminaux** ou **variables**.*

Exemple 2.6

Afin de décrire le langage des mots bien parenthésés, on définit la grammaire non contextuelle $\mathcal{G}_{\mathsf{BP}} = (\mathcal{V}, \Sigma, P, S)$ où :
- $\mathcal{V} \stackrel{\text{déf}}{=} \{B\}$;
- $\Sigma \stackrel{\text{déf}}{=} \{), (\}$;
- $P \stackrel{\text{déf}}{=} \{B \to \varepsilon, B \to BB, B \to (B)\}$;
- $S \stackrel{\text{déf}}{=} B$.

Exercice de cours 2.7

Proposer une définition de type OCaml permettant la représentation des grammaires non contextuelles.

Notation 2.8

Lorsqu'on donne les règles d'une grammaire, on peut mettre en commun plusieurs règles de production ayant le même membre gauche, on regroupe alors les membres droits en les séparant d'un $|$[♡]. Ainsi ici on écrirait $B \to \varepsilon \mid BB \mid (B)$ à la place des trois règles.

♣. On aurait pu imaginer ne pouvoir remplacer V par $w_1 w_2 \dots w_n$ que lorsque V est précédé d'un a par exemple

♡. On supposera donc que le symbole $|$ n'est ni terminal ni non terminal.

Utilisation de la grammaire. Informellement le langage décrit par une grammaire, telle que présentée dans l'exemple précédant, est celui des mots obtenus en partant du mot contenant uniquement le symbole initial, puis en lui appliquant des remplacements autorisés par les règles de production jusqu'à obtenir un mot uniquement constitué de symboles terminaux.

Exemple 2.9
Pour $\mathcal{G}_{\mathsf{BP}} = (\mathcal{V}, \Sigma, P, S)$:
- $B \Rightarrow \varepsilon$ (en utilisant la première des 3 règles de production) donc ε est dans le langage décrit par la grammaire non contextuelle ;
- $B \Rightarrow BB \Rightarrow (B)B \Rightarrow (B)(B) \Rightarrow ()(B) \Rightarrow ()()$ donc `()()` est dans la langage décrit par la grammaire non contextuelle.

On s'attache dans la section suivante à définir la relation notée $\Rightarrow$ ci-dessous, qui permet de passer d'un mot de $(\Sigma \sqcup \mathcal{V})^\star$ à un autre par "remplacement".

2.2 Relations de dérivation

Dans la suite on suppose fixée une grammaire $\mathcal{G} = (\mathcal{V}, \Sigma, P, S)$.

Définition 2.10

Soient deux mots $u \in (\mathcal{V} \cup \Sigma)^\star$ et $v \in (\mathcal{V} \cup \Sigma)^\star$.
*On dit que v **dérive immédiatement** de u, ce que l'on note $u \Rightarrow v$, dès lors qu'il existe $x \in (\mathcal{V} \cup \Sigma)^\star$, $y \in (\mathcal{V} \cup \Sigma)^\star$ et $V \to w$ une règle de production de P tels que $u = x \cdot V \cdot y$ et $v = x \cdot w \cdot y$.*

Exemple 2.11
Pour la grammaire proposée pour les mots bien parenthésés, $(B)B \Rightarrow (B)(B)$. En effet on retrouve la définition avec
- $x = (B)$
- $y = \varepsilon$
- la règle $B \to (B)$

Exercice de cours 2.12

Pour la grammaire exemple, donner l'ensemble des mots w tels que $(B)B \Rightarrow w$.

Exercice de cours 2.13

Pour la grammaire exemple, $\Rightarrow$ est-elle une relation réflexive sur $(\mathcal{V} \cup \Sigma)^\star$?
Est-il possible d'ajouter une règle de production à la grammaire pour rendre $\Rightarrow$ réflexive sur $(\mathcal{V} \cup \Sigma)^\star$?

Lemme 2.14

Soient u, v, w trois mots de $(\mathcal{V} \cup \Sigma)^\star$.
Si $u \Rightarrow v$, alors $u \cdot w \Rightarrow v \cdot w$, et $w \cdot u \Rightarrow w \cdot v$.

Démonstration : Si $u \Rightarrow v$, par définition de la relation $\Rightarrow$ il existe $(x, y) \in ((\mathcal{V} \cup \Sigma)^\star)^2$ et une règle $V \to z$ tels que $u = x \cdot V \cdot y$ et $v = x \cdot z \cdot y$.
En posant $x' = x$ et $y' = y \cdot w$ (resp. $x' = w \cdot x$ et $y' = y$), on a par associativité $u \cdot w = x' \cdot V \cdot y'$ et $v \cdot w = x' \cdot z \cdot y'$ (resp. $w \cdot u = x' \cdot V \cdot y'$ et $w \cdot v = x' \cdot V \cdot y'$), donc $u \cdot w \Rightarrow v \cdot w$ (resp. $w \cdot u \Rightarrow w \cdot v$). □

Définition 2.15

Pour $p \in \mathbb{N}$, on définit la relation $\Rightarrow^p$ comme l'itérée de p fois la relation $\Rightarrow$, formellement :

$$\forall (u,v) \in ((\mathcal{V} \cup \Sigma)^\star)^2, u \Rightarrow^p v \overset{\text{déf}}{\Leftrightarrow} \exists (u_0, u_1, u_2, \ldots, u_p) \in ((\mathcal{V} \cup \Sigma)^\star)^{n+1},$$
$$u = u_0 \text{ et } u_n = v \text{ et } \forall i \in [\![0, n[\![, u_i \Rightarrow u_{i+1}$$

Autrement dit $\Rightarrow^p$ représente la dérivation en p étapes de dérivation immédiates, en particulier si $p = 0$, $\Rightarrow^p$ est la relation d'égalité. On définit la relation $\Rightarrow^\star$ comme la clôture réflexive transitive de la relation $\rightarrow$, formellement :

$$\forall (u,v) \in ((\mathcal{V} \cup \Sigma)^\star)^2, u \overset{\star}{\Rightarrow} v \overset{\text{déf}}{\Leftrightarrow} \exists p \in \mathbb{N}, u \Rightarrow^p v$$

Autrement dit $\Rightarrow^\star$ représente la dérivation en un nombre quelconque (y compris nul) d'étapes de dérivation immédiates.

Remarque 2.16

La relation de dérivation immédiate peut donc être notée $\Rightarrow^1$.

Vocabulaire 2.17

Si $u \Rightarrow^p v$, on dit que v dérive de u en p étapes.
Si $u \Rightarrow^\star v$, on dit que v dérive de u.

Exemple 2.18

Pour $\mathcal{G}_{\text{BP}}$, $B \Rightarrow BB \Rightarrow (B)B \Rightarrow (B)(B) \Rightarrow ()(B) \Rightarrow ()()$, finalement $B \overset{\star}{\Rightarrow} ()()$.

Exercice de cours 2.19

Donner l'ensemble des mots sur $(\Sigma \cup \mathcal{V})^\star$ de taille au plus 4 tels que $B \overset{\star}{\Rightarrow} w$.

Lemme 2.20 (*Lemme de composition*)

Soient u, u', v et v' quatre mots de $(\mathcal{V} \cup \Sigma)^\star$. Soient p et q deux entiers naturels.
Si $u \Rightarrow^p u'$ et $v \Rightarrow^q v'$ alors $u \cdot u' \Rightarrow^{q+p} v \cdot v'$.

Démonstration : Par définition de $\Rightarrow^p$ (resp. de $\Rightarrow^q$), on dispose de $u_1, u_2, \ldots, u_{p-1}$ (resp. de $v_1, v_2, \ldots, v_{q-1}$) des mots de $(\Sigma \cup \mathcal{V})^\star$ tels que $u \Rightarrow u_1 \Rightarrow u_2 \ldots \Rightarrow u_{p-1} \Rightarrow u'$ (resp. $v \Rightarrow v_1 \Rightarrow v_2 \ldots \Rightarrow v_{q-1} \Rightarrow v'$).
D'après le lemme 2.14 $u \cdot v \Rightarrow u_1 \cdot v \Rightarrow u_2 \cdot v \ldots \Rightarrow u_{p-1} \cdot v \Rightarrow u' \cdot v \Rightarrow u' \cdot v_1 \Rightarrow u' \cdot v_2 \ldots \Rightarrow u' \cdot v_{q-1} \Rightarrow u' \cdot v'$. □

Corollaire 2.21

Soient u, u', v et v' quatre mots de $(\mathcal{V} \cup \Sigma)^\star$.
Si $u \overset{\star}{\Rightarrow} u'$ et $v \overset{\star}{\Rightarrow} v'$ alors $u \cdot u' \overset{\star}{\Rightarrow} v \cdot v'$.

Exercice de cours 2.22

Démontrer la propriété précédente.

Lemme 2.23 (*Lemme de décomposition*)

Soient u et v deux mots de $(\mathcal{V}\cup\Sigma)^\star$. Notons $n=|u|$ et $u_1, u_2, \dots, u_n$ les lettres de u.
Si $u \Rightarrow^p v$, alors il existe $(\tilde{v}_1, \tilde{v}_2, \dots, \tilde{v}_n) \in ((\mathcal{V}\cup\Sigma)^\star)^n$ et $(p_1, p_2, \dots, p_n) \in \mathbb{N}^n$ tels que :

$i)$ $v = \tilde{v}_1 \cdot \tilde{v}_2 \cdot \ldots \cdot \tilde{v}_n$; $ii)$ $\forall i \in [\![1,n]\!]$, $u_i \Rightarrow^{p_i} \tilde{v}_i$; $iii)$ $\sum_{i=1}^n p_i = p$.

Autrement dit il est possible de découper v de sorte que chaque facteur dérive d'une lettre de u en un certain nombre d'étapes (et que la somme de ces nombres d'étapes soit p).

Démonstration : Montrons ce résultat par récurrence sur $p \in \mathbb{N}$.

Plus précisément montrons par récurrence sur p la propriété suivante.

$$H_p : \forall (u,v) \in ((\mathcal{V}\cup\Sigma)^\star)^2, u \Rightarrow^p v \Rightarrow \exists (\tilde{v}_i) \in ((\mathcal{V}\cup\Sigma)^\star)^{|u|}, \exists (p_i) \in \mathbb{N}^{|u|}, i), ii) \text{ et } iii)$$

- Pour $p=0$ et $(u,v) \in ((\mathcal{V}\cup\Sigma)^\star)^2$, si $u \Rightarrow^0 v$ alors $u=v$. Ainsi, en choisissant pour tout $i \in [\![1,n]\!]$, $\tilde{v}_i = u_i$, on obtient le résultat souhaité.
- Soit $p \in \mathbb{N}$. Supposons la propriété vraie pour au rang p. Soient de plus deux mots u et v. Notons $n=|u|$. Supposons que $u \Rightarrow^{p+1} v$.
  Puisque $\Rightarrow^{p+1}$ est la composée de $\Rightarrow$ et $\Rightarrow^p$, il existe un mot w tel que $u \Rightarrow w$ et $w \Rightarrow^p v$.
  D'une part puisque $u \Rightarrow w$, il existe deux mots $x \in (\mathcal{V}\cup\Sigma)^\star$ et $y \in (\mathcal{V}\cup\Sigma)^\star$ ainsi qu'une règle de production $V \to z_1 z_2 \dots z_r \in P$ tels que $u = x\cdot V \cdot y$ et $w = x \cdot z_1 z_2 \dots z_r \cdot y$. On remarque alors que $|w| = |x| + r + |y| = n - 1 + r$. Notons alors $n' = n - 1 + r$.
  D'autre part puisque $w \Rightarrow^p v$, l'hypothèse de récurrence assure qu'il existe des mots $\hat{v}_1, \hat{v}_2, \dots, \hat{v}_{n'}$ et des entiers $s_1, s_2, \dots, s_{n'}$ tels que :

  $i)$ $v = \hat{v}_1 \cdot \hat{v}_2 \cdot \ldots \cdot \hat{v}_{n'}$; $ii)$ $\forall i \in [\![1,n']\!], w_i \Rightarrow^{s_i} \hat{v}_i$; $iii)$ $\sum_{i=1}^{n'} s_i = p$.

  Notons $q = |x|$. On pose alors :
  - pour $i \in [\![1,q]\!]$, $\tilde{v}_i \overset{\text{déf}}{=} \hat{v}_i$ et $p_i \overset{\text{déf}}{=} s_i$;
  - $\tilde{v}_{q+1} \overset{\text{déf}}{=} \hat{v}_{q+1} \cdot \hat{v}_{q+2} \cdot \ldots \cdot \hat{v}_{q+r}$ et $p_{q+1} \overset{\text{déf}}{=} 1 + s_{q+1} + \dots + s_{q+r}$;
  - pour $i \in [\![q+2, n]\!]$, $\tilde{v}_i \overset{\text{déf}}{=} \hat{v}_{i+r-1}$ et $p_i \overset{\text{déf}}{=} s_{i+r-1}$.

  De tels $(\tilde{v}_i)_{i\in[\![1,n]\!]}$ et $(p_i)_{i\in[\![1,n]\!]}$ conviennent car ils assurent $i)$ $ii)$ et $iii)$. Ainsi la propriété est vraie au rang $p+1$.

On conclut par principe de récurrence. □

Exercice de cours 2.24

Justifier que $(\tilde{v}_i)_{i\in[\![1,n]\!]}$ et $(p_i)_{i\in[\![1,n]\!]}$ définis dans la preuve précédente conviennent, *i.e.* qu'ils vérifient bien les trois conditions attendues $i)$ $ii)$ et $iii)$.

Exercice de cours 2.25

Justifier que $BB \Rightarrow^8$ `()(()())` pour la grammaire $\mathcal{G}_{\text{BP}}$.
Donner alors la décomposition de cette dérivation correspondant au lemme de décomposition.

Définition 2.26

Soit $\mathcal{G} = (\mathcal{V}, \Sigma, P, S)$ une grammaire non contextuelle.
*Le **langage engendré par la grammaire** $\mathcal{G}$, noté $\mathcal{L}(\mathcal{G})$, est l'ensemble des mots de $\Sigma^\star$ qui dérivent du symbole initial.*

$$\mathcal{L}(\mathcal{G}) \overset{\text{déf}}{=} \{u \in \Sigma^\star \mid S \overset{\star}{\Rightarrow} u\}$$

Exercice de cours 2.27

Donner l'ensemble des mots de longueur au plus 4 dans le langage de la grammaire $\mathcal{G}_{\text{BP}}$.

Exercice de cours 2.28

Donner le langage engendré par la grammaire $(\mathcal{V}, \Sigma, P, S)$ où
- $\mathcal{V} \overset{\text{déf}}{=} \{S, A, B, C, D\}$;
- $\Sigma \overset{\text{déf}}{=} \{a\}$;
- $P \overset{\text{déf}}{=} \{S \to A|B|C|D, A \to BB, B \to CC, C \to DD, D \to a\}$.

Définition 2.29

*On dit qu'un langage est **non contextuel** dès lors qu'il est le langage d'une grammaire non contextuelle.*

Exercice de cours 2.30

Montrer qu'un langage fini est non contextuel.

Définition 2.31

*Deux grammaires sont dites **faiblement équivalentes** dès lors qu'elles engendrent le même langage.*

2.3 Preuves par induction

La propriété suivante 2.32 énonce un principe d'induction. Ce principe permet de démontrer que les mots dérivants d'un symbole non terminal ♣ vérifient tous une certaine propriété.
Pour la grammaire $\mathcal{G}_{\text{BP}}$ on pourrait par exemple démontrer que l'ensemble des mots dérivants du symbole B contiennent autant de) que de (.

Proposition 2.32 (*Principe d'induction sur les grammaires*)

Pour chaque symbole $V \in \mathcal{V}$, on note♡ *$V_\downarrow$ l'ensemble des mots de $\Sigma^\star$ dérivant de V.*

$$V_\downarrow \overset{\text{déf}}{=} \{u \in \Sigma^\star \mid V \overset{\star}{\Rightarrow} u\}$$

Soit $(\mathcal{P}_V)_{V\in\mathcal{V}}$ une famille de propriétés sur $\Sigma^\star$.

*On dit que $u \in \Sigma^\star$ est une **extension de** $w_1 w_2 \dots w_m \in (\mathcal{V} \cup \Sigma)^m$ **valide pour** $(\mathcal{P}_V)_{V\in\mathcal{V}}$ dès lors que u peut s'écrire $\hat{w}_1 \cdot \hat{w}_2 \cdot \dots \cdot \hat{w}_m$ où chaque mot $\hat{w}_i$ dérive du symbole w_i, et si de plus celui-ci est non terminal, le mot $\hat{w}_i$ vérifie la propriété $\mathcal{P}_{w_i}$.*

*On dit que les propriétés $(\mathcal{P}_V)_{V\in\mathcal{V}}$ sont **propagées par une règle** $(X \to w)$ dès lors que toute extension de w valide pour $(\mathcal{P}_V)_{V\in\mathcal{V}}$ vérifie la propriété $\mathcal{P}_X$.*

Le principe d'induction est alors le suivant : si les propriétés $(\mathcal{P}_V)_{V\in\mathcal{V}}$ sont propagées par toutes les règles de la grammaire, alors $\forall V \in \mathcal{V}, \forall u \in V_\downarrow, u$ vérifie la propriété $\mathcal{P}_V$.

Démonstration : On suppose que les propriétés sont propagées par toutes les règles. On veut montrer que $\forall V \in \mathcal{V}, \forall u \in V_\downarrow, \mathcal{P}_V(u)$. Pour $p \in \mathbb{N}$, on définit $V_\downarrow^p = \{u \in \Sigma^\star \mid V \Rightarrow^p u\}$. On montre alors par récurrence forte sur $p \in \mathbb{N}$ la propriété suivante.

$$H_p : \forall V \in \mathcal{V}, \forall w \in V_\downarrow^p, w \text{ vérifie } \mathcal{P}_V$$

- La propriété H_0 est trivialement vraie. En effet, pour tout $V \in \mathcal{V}$, $V_\downarrow^0 = \emptyset$. Par l'absurde, s'il existait $w \in V_\downarrow^0$, on aurait d'une part $w \in \Sigma^\star$ et d'autre part $V \Rightarrow^0 w$ donc $w = V \in \mathcal{V}$: Absurde.

♣. et donc en particulier tous les mots du langage engendré par une grammaire
♡. Cette notation est locale. Il en est de même pour le vocabulaire d'extension valide et de propriété propagée.

- Soit $p \in \mathbb{N}^*$. Supposons $\forall q < p, H_q$. Montrons alors que H_p est vraie.
  Soit $V \in \mathcal{V}$, soit $w \in V_\downarrow^p = \{w \in \Sigma^\star \mid V \Rightarrow^p w\}$.
  Puisque $p > 0$, il existe $u \in (\mathcal{V} \cup \Sigma)^\star$ tel que $V \Rightarrow u$ et $u \Rightarrow^{p-1} w$. Notons $n = |u|$.
  D'après le lemme de décomposition appliqué à la dérivation $u \Rightarrow^{p-1} w$, il existe des mots $\tilde{w}_1, \ldots, \tilde{w}_n$ et des entiers $p_1, \ldots, p_n$ tels que :

  $$i)\ w = \tilde{w}_1 \cdot \tilde{w}_2 \cdot \ldots \cdot \tilde{w}_n \qquad ii)\ \forall i \in [\![1, n]\!], u_i \Rightarrow^{p_i} \tilde{w}_i \qquad iii)\ \sum_{i=1}^{n} p_i = p$$

  De plus la dérivation immédiate $V \Rightarrow u$ assure que $V \to u_1 u_2 \ldots u_n$ est une règle de P, ainsi elle propage $\mathcal{P}_V$ par hypothèse.
  Afin d'utiliser cette propagation, montrons que w est une extension valide de $u_1 u_2 \ldots u_n$. Soit $i \in [\![1, n]\!]$,
  - Si $u_i \in \Sigma$, alors $\tilde{w}_i = u_i$.
  - Sinon, $u_i \in \mathcal{V}$ et d'après $ii)$ $\tilde{w}_i \in u_{i\downarrow}^{p_i}$. Comme $p_i \leqslant p - 1 < p$, on peut appliquer H_{p_i} à la variable u_i et au mot $\tilde{u}_i$, ainsi $\tilde{u}_i$ vérifie $\mathcal{P}_{w_i}$.

  Ainsi w est une extension valide de $u_1 u_2 \ldots u_n$, et puisque la règle $V \to u_1 u_2 \ldots u_n$ propage les propriétés, w vérifie $\mathcal{P}_V$.
  Ceci étant vrai pour tout $V \in \mathcal{V}$ et tout $w \in V_\downarrow^p$, on a bien démontré que H_p est vraie.

□

Exemple 2.33
Considérons la grammaire $\mathcal{G} = (\Sigma, \mathcal{V}, P, S)$ où :

- $\mathcal{V} \overset{\text{déf}}{=} \{P, I, X\}$;
- $\Sigma \overset{\text{déf}}{=} \{a\}$;
- $P \overset{\text{déf}}{=} \{I \to a \mid aIa,\ P \to aPa \mid \varepsilon,\ X \to IPI\}$;
- $S \overset{\text{déf}}{=} X$.

On souhaite montrer que le langage engendré par une telle grammaire ne contient que des mots de longueur paire. On considère à cet effet les propriétés suivants, associés à chacune des trois variables de $\mathcal{G}$.

- $\mathcal{P}_P(w)$: $|w|$ est paire.
- $\mathcal{P}_I(w)$: $|w|$ est impaire.
- $\mathcal{P}_X(w)$: $|w|$ est paire.

Montrons que ces propriétés sont propagées par chaque règle de production.

$P \to \varepsilon$ Une extension de ε ne peut être que ε, qui est bien de longueur paire, donc vérifie bien $\mathcal{P}_P$.

$P \to aPa$ Une extension valide de aPa s'écrit $a \cdot w \cdot a$ où w est de longueur paire (car il vérifie $\mathcal{P}_P$), un tel mot est bien de longueur paire, donc vérifie bien $\mathcal{P}_P$.

$I \to a$ Une extension de a ne peut être que a , qui est bien de longueur impaire, donc vérifie bien $\mathcal{P}_I$.

$I \to aIa$ Une extension valide de aIa s'écrit $a \cdot w \cdot a$ où w est de longueur impaire (car il vérifie $\mathcal{P}_P$), cette extension est de longueur $|w| + 2$ donc de longueur impaire, ainsi elle vérifie bien $\mathcal{P}_I$.

$X \to IPI$ Une extension valide de IPI s'écrit $w_1 \cdot w_2 \cdot w_3$ où w_1, w_2 et w_3 sont resp. de longueurs impaire, paire, impaire, cette extension est de longueur $|w_1| + |w_2| + |w_3|$ qui est paire, ainsi elle vérifie $\mathcal{P}_X$.

Par principe d'induction sur les grammaires, on en déduit que $X_\downarrow$ ne contient que des mots de longueur paire, $I_\downarrow$ ne contient que des mots de longueur impaire, $P_\downarrow$ ne contient que des mots de longueur paire. En particulier $\mathcal{L}(\mathcal{G}) = X_\downarrow$ ne contient que des mots de longueur paire.

Exemple 2.34
Le langage engendré par la grammaire $\mathcal{G}_{\mathsf{BP}}$ est le langage des mots bien parenthésés que l'on note BP. On peut démontrer ce résultat par double inclusion.

$\mathcal{L}(\mathcal{G}) \subseteq \mathsf{BP}$. Soit la propriété $\mathcal{P}_B(w) : w \in \mathsf{BP}$. Montrons que $B_\downarrow$ vérifie $\mathcal{P}_B$, par induction :

- $B \to \varepsilon$. $\varepsilon \in \mathsf{BP}$.
- $B \to BB$. Soit $u \in \mathsf{BP}$ et $v \in \mathsf{BP}$ alors $u \cdot v \in \mathsf{BP}$.
- $B \to (B)$. Soit $u \in \mathsf{BP}$ alors $(u) \in \mathsf{BP}$.

Finalement, par principe d'induction, que $B_\downarrow$ vérifie $\mathcal{P}_B$ et donc $\mathcal{L}(\mathcal{G}) \subseteq \mathsf{BP}$.

$\mathsf{BP} \subseteq \mathcal{L}(\mathcal{G})$. Par induction sur la définition de BP.

- $B \to \varepsilon$ donc $B \overset{\star}{\Rightarrow} \varepsilon$ donc $\varepsilon \in \mathcal{L}(\mathcal{G})$
- Soit $u \in$ BP et $v \in$ BP tels que $u \in \mathcal{L}(\mathcal{G})$ et $v \in \mathcal{L}(\mathcal{G})$. $B \overset{\star}{\Rightarrow} u$ et $B \overset{\star}{\Rightarrow} v$, donc $BB \overset{\star}{\Rightarrow} u{\cdot}v$, or $B \to BB$ donc $B \overset{\star}{\Rightarrow} u{\cdot}v$ donc $u{\cdot}v \in \mathcal{L}(\mathcal{G})$
- Soit $u \in$ BP tel que $u \in \mathcal{L}(\mathcal{G})$. $B \overset{\star}{\Rightarrow} u$ donc $(B) \overset{\star}{\Rightarrow} (u)$, or $B \to (B)$ donc $B \overset{\star}{\Rightarrow} (u)$ donc $(u) \in \mathcal{L}(\mathcal{G})$

Exercice de cours 2.35

Montrer, en utilisant le principe d'induction que les mots du langage de la grammaire $(\Sigma, \mathcal{V}, P, S)$ ci-dessous ne contiennent pas plus de 2 occurrences de la lettre a.

- $\mathcal{V} \overset{\text{déf}}{=} \{A, B\}$;
- $\Sigma \overset{\text{déf}}{=} \{a, b\}$;
- $P \overset{\text{déf}}{=} \{A \to a \mid Ab, B \to A \mid AA \mid Bb\}$;
- $S \overset{\text{déf}}{=} B$.

2.4 Définitions équivalentes

Dans cette section on s'intéresse à d'autres notions de dérivation, que l'on montrera comme étant toutes équivalentes.

2.4.1 Dérivations gauche et droite

Définition 2.36

Soient deux mots $u \in (\mathcal{V} \cup \Sigma)^\star$ et $v \in (\mathcal{V} \cup \Sigma)^\star$.
On dit que v ***dérive immédiatement à gauche (resp. à droite)*** *de u, ce que l'on note $u \Rightarrow_g v$ (resp. $u \Rightarrow_d v$), dès lors qu'il existe deux mots $x \in \Sigma^\star$ et $y \in (\mathcal{V} \cup \Sigma)^\star$ (resp. $x \in (\mathcal{V} \cup \Sigma)^\star$ et $y \in \Sigma^\star$) et une règle de production $V \to w \in P$ tels que $u = x{\cdot}V{\cdot}y$ et $v = x{\cdot}w{\cdot}y$. Autrement dit $u \Rightarrow_g v$ (resp. $u \Rightarrow_d v$) lorsque l'on a appliqué une règle de production sur le premier (resp. dernier) symbole non terminal de u pour obtenir v.*
On définit alors $\overset{\star}{\Rightarrow}_g$ (resp. $\overset{\star}{\Rightarrow}_d$) comme étant la clôture réflexive transitive de $\Rightarrow_g$ (resp. $\Rightarrow_d$).

Exercice de cours 2.37

En utilisant la grammaire $\mathcal{G}_{\text{BP}}$, justifier que $B \overset{\star}{\Rightarrow}_g (())()$. A-t-on pour cette même grammaire $B \overset{\star}{\Rightarrow}_d (())()$?

Exercice de cours 2.38

Soient u, v, w trois mots de $(\mathcal{V} \cup \Sigma)^\star$.
Montrer que si $u \Rightarrow_g v$, alors $u{\cdot}w \Rightarrow_g v{\cdot}w$.
Montrer que si de plus $w \in \Sigma^\star$ on a aussi $w{\cdot}u \Rightarrow_g w{\cdot}v$.

Exercice de cours 2.39

Proposer une grammaire pour laquelle il n'y a pas unicité de la dérivation gauche. Plus précisément on attend une grammaire $\mathcal{G} = (\Sigma, \mathcal{V}, P, S)$, une variable V et un mot $w \in \Sigma^\star$ tels qu'il existe plusieurs suites de dérivations gauches immédiates transformant V en w.

2.4.2 Arbre de dérivation

Définition 2.40

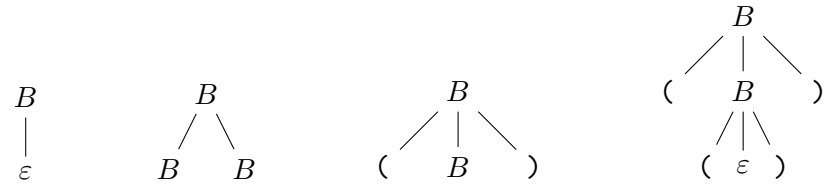
*On appelle **arbre d'analyse** (ou **arbre de dérivation**) un arbre non vide dont les nœuds sont étiquetés par $\mathcal{V} \cup \Sigma \cup \{\varepsilon\}$ et tel que :*

- *la racine est étiquetée par S ;*
- *tout nœud interne (i.e. ayant des fils) est étiqueté par un symbole non terminal $V \in \mathcal{V}$ et*
  - *ses fils sont étiquetés de gauche à droite par s_1, s_2, ..., s_n ♣ et $V \to s_1 s_2 \dots s_n \in P$,*
  - *ou bien il a un seul fils étiqueté par ε et $V \to \varepsilon \in P$.*

*On dit d'un arbre de dérivation qu'il est **clos** lorsque ses feuilles sont toutes étiquetées par des symboles terminaux ou par ε.*

Exemple 2.41

Les arbres suivants sont des arbres de dérivation de la grammaire $\mathcal{G}_{\mathsf{BP}}$.



Le premier et le dernier sont des arbres de dérivation clos.

Exercice de cours 2.42

Pour la grammaire $\mathcal{G}_{\mathsf{BP}}$, donner l'ensemble des arbres de dérivation ayant exactement 4 nœuds.

Définition 2.43

*La **production** (ou le produit) d'un arbre dérivation T, notée prod(T), est définie par la fonction inductive suivante.*

$$\mathit{prod}(T) = \begin{cases} w & \text{si } T = \mathit{Leaf}(w) \\ \mathit{prod}(T_1) \cdot \mathit{prod}(T_2) \dots \cdot \mathit{prod}(T_n) & \text{si } T = \mathit{Node}(\_, T_1, T_2, \dots, T_n) \end{cases}$$

*On dit qu'un mot w **admet un arbre de dérivation** s'il est le produit d'un arbre de dérivation.*

Exemple 2.44

Les arbres de l'exemple précédent ont respectivement pour production ε, BB, (B) et $(())$.

Exercice de cours 2.45

Pour la grammaire $\mathcal{G}_{\mathsf{BP}}$, donner deux arbres de dérivation différents ayant la même production $(())()(())$.

Exercice de cours 2.46

Donner une condition nécessaire et suffisante sur la production d'un arbre de dérivation pour que celui-ci soit clos.

2.4.3 Équivalence

♣. On sous-entend ainsi que les s_i sont des symboles de $\mathcal{V} \cup \Sigma$.

Proposition 2.47

Pour tout mot $w \in \Sigma^\star$, les 4 propriétés suivantes sont équivalentes :

1. $w \in \mathcal{L}(\mathcal{G})$;
2. $w \in \mathcal{L}_g(\mathcal{G})$ *où* $\mathcal{L}_g(\mathcal{G}) = \{w \in \Sigma^\star \mid S \overset{\star}{\Rightarrow}_g w\}$;
3. $w \in \mathcal{L}_d(\mathcal{G})$ *où* $\mathcal{L}_d(\mathcal{G}) = \{w \in \Sigma^\star \mid S \overset{\star}{\Rightarrow}_d w\}$;
4. w *admet un arbre de dérivation dans la grammaire* $\mathcal{G}$.

Démonstration : On démontre $4 \Rightarrow 3$, $4 \Rightarrow 2$, $2 \Rightarrow 1$, $3 \Rightarrow 1$, $1 \Rightarrow 4$, ce qui suffira pour conclure. On généralise la notion d'arbre de dérivation : on l'autorise à être enraciné en n'importe quel symbole $V \in \mathcal{V}$ et pas seulement en S. On appelle de tels arbres des **arbres de dérivation généralisés**. On remarque qu'un sous-arbre d'un arbre de dérivation généralisé est, ou bien réduit à une feuille, ou bien encore un arbre de dérivation généralisé. De plus si T est un tel arbre, on note $\mathsf{racine}(T)$ le symbole étiquetant sa racine.

$4 \Rightarrow 3$ On montre par induction sur les arbres non vides étiquetés par $\mathcal{V} \cup \Sigma \cup \{\varepsilon\}$ la propriété suivante.

$$\mathcal{P}(T) : \text{Si } T \text{ est un arbre de dérivation généralisé clos, alors } \mathsf{racine}(T) \overset{\star}{\Rightarrow}_g \mathsf{prod}(T)$$

- Soit T un arbre de la forme $\mathsf{Leaf}(\_)$. Par définition un tel arbre n'est pas un arbre de dérivation généralisé, ainsi la propriété $\mathcal{P}(T)$ est trivialement vérifiée.
- Soit T un arbre non réduit à une feuille, dont les sous-arbres vérifient tous $\mathcal{P}$. Montrons $\mathcal{P}(T)$.
  On suppose que T est un arbre de dérivation généralisé clos (*i.e.* $\mathsf{prod}(T) \in \Sigma^\star$).
  Par définition des arbres de dérivation, il y a deux cas possibles.
  - Soit T est de la forme $\mathsf{Node}(V, \varepsilon)$ et $V \to \varepsilon \in P$.
    Dans ce cas $\mathsf{prod}(T) = \varepsilon$, donc on a bien $\mathsf{racine}(T) = V \overset{\star}{\Rightarrow}_g \varepsilon = \mathsf{prod}(T)$ donc $\mathcal{P}(T)$ est vraie.
  - Soit T est de la forme $\mathsf{Node}(V, T_1, T_2, \dots, T_n)$ et $V \to \mathsf{racine}(T_1) \cdot \mathsf{racine}(T_2) \dots \mathsf{racine}(T_n)$.
    Pour tout $i \in [\![1, n]\!]$, $\mathsf{racine}(T_i) \overset{\star}{\Rightarrow}_g \mathsf{prod}(T_i)$. En effet T_i est, ou bien réduit à une feuille étiquetée par un symbole $s_i \in \mathcal{V} \cup \Sigma$, auquel cas $\mathsf{racine}(T_i) = \mathsf{prod}(T_i) = s_i$, ou bien un arbre de dérivation clos (car $\mathsf{prod}(T_i) \in \Sigma^\star$ en tant que facteur de $\mathsf{prod}(T) \in \Sigma^\star$) vérifiant $\mathcal{P}(T_i)$.
    Notons alors $(v_i^j)_{j \in [\![0, p_i]\!]}$ la suite correspondant à cette dérivation gauche, ainsi on a en particulier $v_i^0 = \mathsf{racine}(T_i)$ et $v_i^{p_i} = \mathsf{prod}(T_i) \in \Sigma^\star$. On considère alors la suite de dérivation gauches immédiates suivante.

$$\begin{aligned}
\mathsf{racine}(T) = V &\Rightarrow_g \mathsf{racine}(T_1) \dots \mathsf{racine}(T_n) = v_1^0 \cdot v_2^0 \dots \cdot v_n^0 \\
&\Rightarrow_g v_1^1 \cdot v_2^0 \dots \cdot v_n^0 \Rightarrow_g v_1^2 \cdot v_2^0 \dots \cdot v_n^0 \Rightarrow_g \dots \Rightarrow_g v_1^{p_1} \cdot v_2^0 \dots \cdot v_n^0 \\
&\Rightarrow_g v_1^{p_1} \cdot v_2^1 \dots \cdot v_n^0 \Rightarrow_g v_1^{p_1} \cdot v_2^2 \dots \cdot v_n^0 \Rightarrow_g \dots \Rightarrow_g v_1^{p_1} \cdot v_2^{p_2} \dots \cdot v_n^0 \\
&\vdots \\
&\Rightarrow_g v_1^{p_1} \dots v_{n-1}^{p_{n-1}} \cdot v_n^1 \Rightarrow_g v_1^{p_1} \dots v_{n-1}^{p_{n-1}} \cdot v_n^2 \Rightarrow_g \dots \Rightarrow_g v_1^{p_1} \dots v_{n-1}^{p_{n-1}} \cdot v_n^{p_n} \\
&= \mathsf{prod}(T_1)\mathsf{prod}(T_2) \dots \mathsf{prod}(T_n) = \mathsf{prod}(T)
\end{aligned}$$

  Finalement $\mathsf{racine}(T) \overset{\star}{\Rightarrow}_g \mathsf{prod}(T)$, ainsi $\mathcal{P}(T)$ est vrai.

On conclut donc par induction.

$4 \Rightarrow 2$ De même mais on commence par réécrire v_n

$2 \Rightarrow 1$ $\overset{\star}{\Rightarrow}_g \subseteq \overset{\star}{\Rightarrow}$, le résultat est donc trivial

$3 \Rightarrow 1$ $\overset{\star}{\Rightarrow}_d \subseteq \overset{\star}{\Rightarrow}$, le résultat est donc trivial

$1 \Rightarrow 4$ Montrons par récurrence forte sur $n \in \mathbb{N}$ la propriété $\mathcal{P}_n$ définie ci-dessous.

$$\mathcal{P}_n : \forall V \in \mathcal{V}, \forall w \in \Sigma^\star, \text{si } V \Rightarrow^n w \text{ alors } w \text{ admet un arbre de dérivation enraciné en } V$$

- La propriété P_0 est trivialement vraie car aucun mot $w \in \Sigma^\star$ ne dérive d'une variable V en 0 étape.
- Soit $n \in \mathbb{N}^*$. Supposons que $\forall k < n, \mathcal{P}_k$. Montrons alors $\mathcal{P}_n$.
  Soit $V \in \mathcal{V}$ et $w \in \Sigma^\star$ tels que $V \Rightarrow^n w$.
  Puisque $n > 0$, il existe $u \in (\mathcal{V} \cup \Sigma)^\star$ tel que $V \Rightarrow u$ et $u \Rightarrow^{p-1} w$. Notons $n = |u|$.
  D'une part, d'après le lemme de décomposition appliqué à la dérivation $u \Rightarrow^{p-1} w$, il existe des mots $\tilde{w}_1$, …, $\tilde{w}_n$ et des entiers p_1, …, p_n tels que :

$i)\ w = \tilde{w}_1 \cdot \tilde{w}_2 \cdot \ldots \cdot \tilde{w}_n$ $\quad ii)\ \forall i \in [\![1,n]\!], u_i \Rightarrow^{p_i} \tilde{w}_i$ $\quad iii)\ \sum_{i=1}^{n} p_i = n$

On peut alors considérer, pour chaque $i \in [\![1,r]\!]$, l'arbre T_i défini de la manière suivante.

- Si $v_i \in \Sigma$ alors la dérivation $u_i \Rightarrow^{p_i} \tilde{w}_i$ est en fait une égalité : $p_i = 0$ et $\tilde{w}_i = u_i$. On définit alors T_i comme étant la feuille $\mathsf{Leaf}(v_i)$, ainsi T_i est enraciné en u_i et de production $\tilde{w}_i$.
- Si $v_i \in \mathcal{V}$, l'hypothèse de récurrence $\mathscr{P}_{p_i}$ (qui est vraie car $p_i \leqslant n$ d'après iii)) appliquée à la dérivation $u_i \Rightarrow^{p_i} \tilde{w}_i$ assure qu'il existe un arbre de dérivation généralisé enraciné en u_i de production $\tilde{w}_i$. On définit alors T_i comme étant cet arbre.

D'autre part la dérivation immédiate $V \Rightarrow u$ assure que $V \to u_1 u_2 \ldots u_n$ est une règle de P. On peut donc construire un arbre de dérivation généralisé enraciné en V en plaçant sous le nœud racine r arbres de dérivations respectivement enracinés en u_i.

Ainsi on construit finalement l'arbre de dérivation généralisé $T = \mathsf{Node}(V, T_1, T_2, \ldots, T_r)$. Par définition de la production, $\mathsf{prod}(T) = \mathsf{prod}(T_1) \cdot \mathsf{prod}(T_2) \cdot \ldots \cdot \mathsf{prod}(T_r) = \tilde{w}_1 \cdot \tilde{w}_2 \cdot \ldots \cdot \tilde{w}_r = w$, ainsi T convient.

Ceci étant vrai pour tout $V \in \mathcal{V}$ et $w \in \Sigma^\star$ tels que $V \Rightarrow^n w$, on en déduit que $\mathscr{P}_n$ est vraie.

On conclut par récurrence.

□

Remarque 2.48

> On remarque que dans la preuve précédente, on associe à une dérivation $S \overset{\star}{\Rightarrow} w$ un arbre de dérivation en suivant une suite de dérivations immédiates. On parle alors de l'arbre de dérivation associé à une suite de dérivations. Un mot peut admettre plusieurs arbres de dérivations, mais suite de dérivations ne correspond qu'à un seul arbre de dérivation.

Exercice de cours 2.49

> Préciser la propriété précédente 2.47 en indiquant s'il existe des liens entre les longueurs des dérivations gauches, droites et quelconques.

Exemple 2.50

On considère la grammaire $\mathcal{G}_{\mathsf{arith}} = (\{A\}, \{\mathtt{+}, \mathtt{-}, \mathtt{1}, \mathtt{2}\}, P, A)$ où $P = \{A \to \mathtt{1} \mid \mathtt{2} \mid A\mathtt{+}A \mid A\mathtt{-}A\}$.

Dans $\mathcal{G}_{\mathsf{arith}}$ le mot 1-2+1 admet les dérivations suivantes :

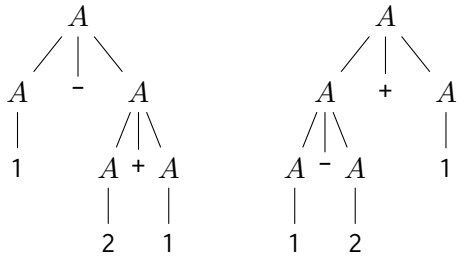
$a)\ A \Rightarrow A\mathtt{-}A \Rightarrow A\mathtt{-}A\mathtt{+}A \Rightarrow \mathtt{1}\mathtt{-}A\mathtt{+}A \Rightarrow \mathtt{1}\mathtt{-}\mathtt{2}\mathtt{+}A \Rightarrow \mathtt{1}\mathtt{-}\mathtt{2}\mathtt{+}\mathtt{1}$

$b)\ A \Rightarrow A\mathtt{-}A \Rightarrow A\mathtt{-}A\mathtt{+}A \Rightarrow A\mathtt{-}\mathtt{2}\mathtt{+}A \Rightarrow \mathtt{1}\mathtt{-}\mathtt{2}\mathtt{+}A \Rightarrow \mathtt{1}\mathtt{-}\mathtt{2}\mathtt{+}\mathtt{1}$

$c)\ A \Rightarrow A\mathtt{+}A \Rightarrow A\mathtt{-}A\mathtt{+}A \Rightarrow \mathtt{1}\mathtt{-}A\mathtt{+}A \Rightarrow \mathtt{1}\mathtt{-}\mathtt{2}\mathtt{+}A \Rightarrow \mathtt{1}\mathtt{-}\mathtt{2}\mathtt{+}\mathtt{1}$

$d)\ A \Rightarrow A\mathtt{+}A \Rightarrow A\mathtt{-}A\mathtt{+}A \Rightarrow A\mathtt{-}\mathtt{2}\mathtt{+}A \Rightarrow \mathtt{1}\mathtt{-}\mathtt{2}\mathtt{+}A \Rightarrow \mathtt{1}\mathtt{-}\mathtt{2}\mathtt{+}\mathtt{1}$

Les dérivations $a)$ et $b)$ ci-dessus correspondent à l'arbre de gauche ci-contre, tandis que $c)$ et $d)$ correspondent à l'arbre de droite.



L'exemple ci-dessus démontre que, pour une grammaire donnée, un mot du langage engendré par cette grammaire ne correspond pas nécessairement à un unique arbre de dérivation. On rappelle qu'un des objectifs de ce chapitre était de permettre la description d'une nouvelle famille de langage qui permet la description, entre autres, des langages de programmation. La grammaire de l'exemple précédent est un début de description de l'ensemble des expressions arithmétiques du langage OCaml, or cette description ne permet pas une lecture unique des mots qu'elle décrit, autrement dit elle n'assure pas l'unicité de l'arbre de syntaxe d'une expressions valide. En effet l'expression 1-2+1 est comprise comme $(1-(2+1))$ ou comme $((1-2)+1)$ selon l'arbre utilisé, or les valeurs de ces deux termes ne sont pas les mêmes.

2.5 Ambiguïté

Définition 2.51

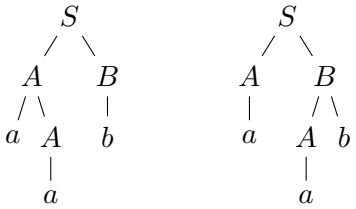
On dit d'une grammaire qu'elle est ***ambiguë*** *dès lors qu'il existe des mots de son langage qui admettent strictement plus d'un arbre de dérivation.*

Exemple 2.52

On considère la grammaire $\mathcal{G} = (\Sigma, \mathcal{V}, P, S)$ où :

- $\Sigma = \{a, b\}$;
- $\mathcal{V} \overset{\text{déf}}{=} \{S, A, B\}$;
- $P \overset{\text{déf}}{=} \{S \to AB,\ A \to aA \mid a,\ B \to aB \mid b\}$.

Le mot aab admet les deux arbres de dérivation ci-contre pour $\mathcal{G}$, ainsi $\mathcal{G}$ est une grammaire ambiguë.



Exemple 2.53

On souhaite définir une grammaire non contextuelle dont le langage est l'ensemble des programmes OCaml. Intéressons-nous plus particulièrement aux expressions construites avec la syntaxe `if ... then ... else ...`, où le `else` peut être omis. On propose la grammaire $(\Sigma, \mathcal{V}, P, S)$ où :

- $\Sigma \overset{\text{déf}}{=} \{$`if`, `then`, `else`, c$\}$;
- $\mathcal{V} \overset{\text{déf}}{=} \{A\}$;
- $P \overset{\text{déf}}{=} \{A \to$ c $\mid$ `if` A `then` A `else` $A \mid$ `if` A `then` $A\}$;
- $S \overset{\text{déf}}{=} A$.

Cette grammaire est ambiguë. En effet l'expression `if c then if c then c else c` admet deux arbres de dérivation, correspondant aux parenthésages suivants : `if c then (if c then c else c)` et `if c then (if c then c) else c`.

Exercice de cours 2.54

Représenter les arbres de dérivation correspondant aux deux parenthésages indiqués dans l'exemple précédent.

Exercice de cours 2.55

Justifier que la grammaire $\mathcal{G}_{\text{BP}}$ est ambiguë.

La proposition ci-dessous donne un exemple de preuve de non ambiguïté d'une grammaire non contextuelle.

Proposition 2.56

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

- $\Sigma \overset{\text{déf}}{=} \{(, c,)\}$;
- $\mathcal{V} \overset{\text{déf}}{=} \{S, T\}$;
- $P \overset{\text{déf}}{=} \{S \to TS|c,\ T \to (S)\}$.

Cette grammaire est non ambiguë.

Démonstration : Pour cela établissons les définitions et propriétés suivantes.

Soit $w \in \Sigma^\star$. Notons $n = |w|$. Pour $i \in [\![0, n]\!]$, notons $\overline{w}^i$ le nombre de parenthèses ouvertes et non fermées dans le préfixe de longueur i de w.

$$\overline{w}^i \overset{\text{déf}}{=} |w_{|[\![1,i]\!]}|_(- |w_{|[\![1,i]\!]}|_)$$

Lemme 2.57

Pour tout mot $w \in \Sigma^\star$, si $T \overset{\star}{\Rightarrow} w$ alors $\forall i \in [\![1, |w|[\![, \overline{w}^i > 0$ et $\overline{w}^{|w|} = 0$,
si $S \overset{\star}{\Rightarrow} w$ alors $\forall i \in [\![1, |w|]\!], \overline{w}^i \geqslant 0$ et $\overline{w}^{|w|} = 0$.

Démonstration : On le montre par induction sur la grammaire, avec comme prédicats associés aux deux variables les deux propriétés ci-dessus (notées $\mathcal{P}_T$ et $\mathcal{P}_S$ respectivement).

- Cas 1 : règle $S \to c$.
  Soit w un mot de la forme du membre droit de cette règle, à savoir : $w = c$. Puisque w ne contient aucune parenthèse, $\overline{w}^1 = 0$, donc $\mathscr{P}_S(w)$ est vraie.

- Cas 2 : règle $S \to TS$.
  Soit w un mot de la forme uv où u vérifie $\mathscr{P}_T$ et v vérifie $\mathscr{P}_S$. On remarque que $|w| = |u| + |v|$
  Ainsi d'une part $\forall i \in [\![1, |u|[\![, \overline{u}^i > 0$ et $\overline{u}^{|u|} = 0$ et d'autre part $\forall i \in [\![1, |v|]\!], \overline{v}^i \geqslant 0$ et $\overline{v}^{|v|} = 0$.
  Soit $i \in [\![1, |w|]\!]$, raisonnons par disjonction de cas.
  - Si $i \in [\![1, |u|]\!]$, $\overline{w}^i = \overline{u}^i \geqslant 0$.
  - Sinon si $i \in [\![|u| + 1, |u| + |v| - 1]\!]$, $\overline{w}^i = \overline{u}^{|u|} + \overline{v}^{i-|u|} = \overline{v}^{i-|u|} \geqslant 0$.
  - Sinon $\overline{w}^{|w|} = \overline{u}^{|u|} + \overline{v}^{|v|} = 0$.
  Ces trois points montrent que $\mathscr{P}_S(w)$ est vraie.

- Cas 3 : règle $T \to (S)$.
  Soit w un mot de la forme $w = (v)$, où u vérifie $\mathscr{P}_S$, autrement dit $\forall i \in [\![1, |u|]\!], \overline{u}^i \geqslant 0$ et $\overline{u}^{|u|} = 0$.
  Soit alors $i \in [\![1, |w|]\!]$, raisonnons par disjonction de cas.
  - Si $i = 1$, $\overline{w}^1 = 1 > 0$.
  - Sinon si $i \in [\![2, |u|]\!]$, $\overline{w}^i = 1 + \overline{u}^{i-1} \geqslant 1 > 0$.
  - Sinon $\overline{w}^{|w|} = 1 + \overline{u}^{|u|} - 1 = 0$.
  Ces trois points montrent que $\mathscr{P}_T(w)$ est vraie.

□

Lemme 2.58

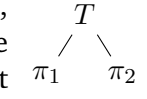
Pour tout mot $w \in \Sigma^\star$, *Si* $T \overset{\star}{\Rightarrow} w$ *alors* w *admet un unique arbre de dérivation de racine* T
Si $S \overset{\star}{\Rightarrow} w$ *alors* w *admet un unique arbre de dérivation de racine* S

Démonstration : On le montre par induction sur la grammaire, avec comme prédicat associé à la variable $V \in \{T, S\}$,

$$\mathscr{Q}_V(w) : w \text{ admet un unique arbre de dérivation généralisé de racine } V$$

- Cas 1 : règle $S \to c$.
  Soit donc w un mot de la forme du membre droit de cette règle, à savoir : $w = c$. Une dérivation de w depuis S ne peut être de la forme $S \Rightarrow TS$ sans quoi le mot w commencerait par (ce qui est absurde. Ainsi une dérivation de w depuis S est de la forme $S \Rightarrow c$, elle est donc unique. w admet donc un unique arbre de dérivation depuis S.

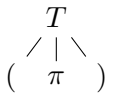
- Cas 2 : règle $S \to TS$.
  Soit w un mot de la forme uv, où u vérifie $\mathscr{Q}_T$ et v vérifie $\mathscr{Q}_S$. D'après le lemme 2.57, $u_1 = ($, donc $w_1 = ($. Ainsi une suite de dérivations de S à w ne peut être de la forme $S \Rightarrow c$, elle commence donc par $S \Rightarrow TS$. Un arbre de dérivation de racine S et de production w est donc de la forme suivante où π_1 (resp. π_2) est un arbre de racine T (resp. S).

  Notons $u' = \mathsf{prod}(\pi_1)$ et $v' = \mathsf{prod}(\pi_2)$ (ainsi $w = u' \cdot v'$) et montrons alors que $u = u'$ et $v = v'$. Puisque $uv = w = u'v'$, ou bien $u = u'$ (1), ou bien u est un préfixe strict de u' (2), ou bien u' est un préfixe strict de u (3). Montrons que le cas (2) est en fait absurde. Pour des raisons de symétrie le cas (3) le sera aussi, nous serons donc dans le cas d'égalité (1).
  Si u est un préfixe strict de u', en appliquant le lemme 2.57 à u et u' : $0 = \overline{u}^{|u|} = \overline{u'}^{|u|} > 0$ ce qui est absurde. Autrement dit : u ne peut être un préfixe strict de u' pour des raisons de niveau de parenthèses. Finalement $u = u'$ et $v = v'$.
  Ainsi π_1 est un arbre de dérivation de racine T et de production u, or d'après $\mathscr{Q}_T(u)$ un tel arbre est unique. De même d'après $\mathscr{Q}_S(v)$, π_2 est unique en tant qu'arbre de dérivation de racine S et de production v. Par unicité de ces deux arbres, on conclut à l'unicité de l'arbre de dérivation de racine S et de production w.

- Cas 3 : règle $T \to (S)$.

Soit w un mot de la forme (u), où u vérifie $\mathscr{P}_S$. Une dérivation de w depuis T ne peut être que de la forme $T \Rightarrow (S) \Rightarrow \ldots \Rightarrow (u)$. Ainsi un arbre de dérivation de racine T et de production w est nécessairement de la forme ci-contre où π est un arbre de dérivation de racine S et de production u. D'après $\mathscr{Q}_S(u)$ un tel arbre π est unique, d'où l'unicité de l'arbre de dérivation de racine T et de production w.



Nous avons traité toutes les règles de la grammaire. Nous pouvons en particulier en déduire, par principe d'induction sur les grammaires, que tous les mots engendrés par la grammaire admettent un unique arbre de dérivation (de racine S). □

□

3 Hiérarchie des langages au programme

3.1 Stabilité des langages non contextuels

Proposition 3.1

L'ensemble des langages non contextuels est stable par union.

Démonstration : Soit deux langages L_1 et L_2 tels qu'il existe $\mathcal{G}_1 = (\mathcal{V}_1, \Sigma, P_1, S_1)$ et $\mathcal{G}_2 = (\mathcal{V}_2, \Sigma, P_2, S_2)$ telles que $\mathcal{L}(\mathcal{G}_1) = L_1$ et $\mathcal{L}(\mathcal{G}_2) = L_2$. Quitte à renommer les symboles non terminaux, on peut supposer $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$. Soit alors la grammaire $\mathcal{G} = (\{S\} \cup \mathcal{V}_1 \cup \mathcal{V}_2, \Sigma, P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}, S)$ où $S \notin \mathcal{V}_1 \cup \mathcal{V}_2$. Montrons alors que $\mathcal{L}(\mathcal{G}) = L_1 \cup L_2$. Par récurrence que $S_1 \overset{\star}{\Rightarrow}_{\mathcal{G}_1} w \Leftrightarrow S_1 \overset{\star}{\Rightarrow}_{\mathcal{G}} w$, de même pour S_2 dans $\mathcal{G}_2$, pour tout $w \in \Sigma^\star$. Ainsi, pour tout $w \in \Sigma^\star$:

$$\begin{aligned} w \in \mathcal{L}(\mathcal{G}) &\Leftrightarrow S \overset{\star}{\Rightarrow}_{\mathcal{G}} w \\ &\Leftrightarrow (S \Rightarrow_{\mathcal{G}} S_1 \overset{\star}{\Rightarrow}_{\mathcal{G}} w) \text{ ou } (S \Rightarrow_{\mathcal{G}} S_2 \overset{\star}{\Rightarrow}_{\mathcal{G}} w) \\ &\Leftrightarrow (S_1 \overset{\star}{\Rightarrow}_{\mathcal{G}_1} w) \text{ ou } (S_2 \overset{\star}{\Rightarrow}_{\mathcal{G}_2} w) \\ &\Leftrightarrow (w \in \mathcal{L}(\mathcal{G}_1)) \text{ ou } (w \in \mathcal{L}(\mathcal{G}_2)). \end{aligned}$$

Ce qui conclut la preuve. □

Proposition 3.2

L'ensemble des langages non contextuels est stable par concaténation.

Démonstration : Soit deux langages L_1 et L_2 tels qu'il existe $\mathcal{G}_1 = (\mathcal{V}_1, \Sigma, P_1, S_1)$ et $\mathcal{G}_2 = (\mathcal{V}_2, \Sigma, P_2, S_2)$ telles que $\mathcal{L}(\mathcal{G}_1) = L_1$ et $\mathcal{L}(\mathcal{G}_2) = L_2$. Quitte à renommer les symboles non terminaux, on peut supposer $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$. Soit alors la grammaire $\mathcal{G} = (\{S\} \cup \mathcal{V}_1 \cup \mathcal{V}_2, \Sigma, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S)$ où $S \notin \mathcal{V}_1 \cup \mathcal{V}_2$. Montrons alors que $\mathcal{L}(\mathcal{G}) = L_1 \cdot L_2$. Par récurrence que $S_1 \overset{\star}{\Rightarrow}_{\mathcal{G}_1} w \Leftrightarrow S_1 \overset{\star}{\Rightarrow}_{\mathcal{G}} w$, de même pour S_2 dans $\mathcal{G}_2$, pour tout $w \in \Sigma^\star$. Ainsi, pour tout $w \in \Sigma^\star$:

$$\begin{aligned} w \in \mathcal{L}(\mathcal{G}) &\Leftrightarrow S \overset{\star}{\Rightarrow}_{\mathcal{G}} w \\ &\Leftrightarrow (S \Rightarrow_{\mathcal{G}} S_1 S_2 \overset{\star}{\Rightarrow}_{\mathcal{G}} w) \\ &\Leftrightarrow (\exists w_1, w_2 \in \Sigma^\star, S_1 \overset{\star}{\Rightarrow}_{\mathcal{G}} w_1 \text{ et } S_2 \overset{\star}{\Rightarrow}_{\mathcal{G}} w_2 \text{ et } w = w_1 w_2) && \text{lemme de décomposition} \\ &\Leftrightarrow (\exists w_1, w_2 \in \Sigma^\star, S_1 \overset{\star}{\Rightarrow}_{\mathcal{G}_1} w_1 \text{ et } S_2 \overset{\star}{\Rightarrow}_{\mathcal{G}_2} w_2 \text{ et } w = w_1 w_2) \\ &\Leftrightarrow (\exists w_1, w_2 \in \Sigma^\star, w_1 \in \mathcal{L}(\mathcal{G}_1) \text{ et } w_2 \in \mathcal{L}(\mathcal{G}_2) \text{ et } w = w_1 w_2) \\ &\Leftrightarrow w \in L_1 \cdot L_2. \end{aligned}$$

Ce qui conclut la preuve. □

Proposition 3.3

L'ensemble des langages non contextuels est stable par étoile de Kleene.

Démonstration : Soit un langage L_1 tel qu'il existe $\mathcal{G}_1 = (\mathcal{V}_1, \Sigma, P_1, S_1)$ telle que $\mathcal{L}(\mathcal{G}_1) = L_1$. Soit alors un nouveau symbole non terminal $S \notin \mathcal{V}_1$. Considérons la grammaire $\mathcal{G} = (\mathcal{V}_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to SS_1, S \to \varepsilon\}, S)$. Par récurrence que $S_1 \overset{\star}{\Rightarrow}_{\mathcal{G}_1} w \Leftrightarrow S_1 \overset{\star}{\Rightarrow}_{\mathcal{G}} w$. Ainsi pour tout $w \in \Sigma^\star$:

$$
\begin{aligned}
w \in \mathcal{L}(\mathcal{G}) &\Leftrightarrow S \overset{\star}{\Rightarrow}_{g,\mathcal{G}} w && \text{dérivation gauche}\\
&\Leftrightarrow S \Rightarrow_{g,\mathcal{G}} SS_1 \Rightarrow_{g,\mathcal{G}} \dots \Rightarrow_{g,\mathcal{G}} SS_1S_1 \dots S_1 \Rightarrow_{g,\mathcal{G}} \underbrace{S_1S_1 \dots S_1}_{n} \overset{\star}{\Rightarrow}_{g,\mathcal{G}} w \\
&\Leftrightarrow \exists w_1, w_2, \dots, w_n, \forall i \in [\![1,n]\!], S_1 \overset{\star}{\Rightarrow}_{\mathcal{G}} w_i \text{ et } w = w_1w_2 \dots w_n && \text{lemme décomp.}\\
&\Leftrightarrow \exists w_1, w_2, \dots, w_n, \forall i \in [\![1,n]\!], S_1 \overset{\star}{\Rightarrow}_{\mathcal{G}_1} w_i \text{ et } w = w_1w_2 \dots w_n \\
&\Leftrightarrow \exists w_1, w_2, \dots, w_n, \forall i \in [\![1,n]\!], w_i \in \mathcal{L}(\mathcal{G}_1) \text{ et } w = w_1w_2 \dots w_n \\
&\Leftrightarrow w \in L_1^\star.
\end{aligned}
$$

Ce qui conclut la preuve. □

Remarque 3.4

L'ensemble des langages non contextuels n'est pas clos par intersection et par complémentaire. La démonstration de ce résultat sort du contexte du programme.

3.2 Lien avec les langages réguliers

Proposition 3.5

L'ensemble des langages réguliers est strictement inclus dans l'ensemble des langages non contextuels.

Démonstration : Le fait que l'inclusion est stricte nous est donné par le langage des mots bien parenthésés, qui n'est pas régulier, mais que nous avons montré être le langage de la grammaire exemple de ce chapitre. De plus les langages non contextuels sont stables par les opérations régulières, ainsi pour montrer que les langages réguliers sont non contextuels, il suffit de montrer que les langages réguliers "de base" sont des langages d'une grammaire non contextuelle.

- $\emptyset$ est non contextuel. Il suffit de considérer la grammaire $(\{S\}, \Sigma, \emptyset, S)$
- $\{\varepsilon\}$ est non contextuel. Il suffit de considérer la grammaire $(\{S\}, \Sigma, \{S \to \varepsilon\}, S)$
- Pour tout $a \in \Sigma$, $\{a\}$ est non contextuel. Il suffit de considérer la grammaire $(\{S\}, \Sigma, \{S \to a\}, S)$

□

3.3 Lien avec les langages décidables

Proposition 3.6

L'ensemble des langages non contextuels est strictement inclus dans l'ensemble des langages décidables.

Démonstration : La démonstration du côté strict de l'inclusion sort du contexte du programme. Le programme demande de plus de montrer l'inclusion des langages non contextuels dans les langages décidables

uniquement sur des exemples. Ainsi on doit se donner des exemples de grammaires non contextuelles et fournir un algorithme permettant de tester si un mot est reconnu par ce langage. □

Exemple 3.7

On considère la grammaire ♣ $S \to TS|\texttt{C}$ et $T \to \texttt{O}S\texttt{F}$ où les symboles terminaux sont `C`, `O` et `F` ♡.
On cherche alors un algorithme permettant de tester l'appartenance d'un mot au langage engendré par cette grammaire (et dans le cas d'un mot appartenant au langage, de fournir un arbre de dérivation pour ce mot). L'idée de l'algorithme est la suivante. On définit une fonction par symbole de variable (ici `parse_s` et `parse_t`). La fonction pour la variable V prend en argument un mot w et cherche un préfixe v de w tel que $S \overset{\star}{\Rightarrow} v$. Elle retourne alors un arbre de dérivation dde racine S et de production v et le mot u tel que $v \cdot u = w$. Cela conduit au code suivant.

```
type lettre = C | O | F
type mot    = lettre list
type var    = S | T

(*
  On fabrique un parseur pour la grammaire G
   S -> TS | C
   T -> O S F
*)

(* adc pour arbre de dérivation clos *)
type adc =
  | Node of var * adc list
  | Leaf of lettre option       (* None pour epsilon *)

exception NonValide

(** Calcule un arbre a de racine S et un mot m' tq prod(a).m'= m s'il en
    existe, et déclenche l'erreur [NonValide] sinon *)
let rec parse_s (m: mot) : adc * mot =
  match m with
  | []      -> raise NonValide
  | C :: m' -> Node(S, [Leaf(Some(C))] ), m'
  | _ ->
    let aT, m'  = parse_t m in
    let aS, m'' = parse_s m' in
    Node(S, [aT; aS]), m''

(** Calcule un arbre a de racine T et un mot m' tq prod(a).m'= m si cela
    existe et déclenche l'exceptino [NonValide] sinon. *)
and parse_t (m: mot) : adc * mot =
  match m with
  | []      -> raise NonValide
  | O :: m' ->
    begin
      let aS, m'' = parse_s m' in
      match m'' with
      | F :: m''' -> Node(S, [Leaf(Some(O)); aS; Leaf(Some(F))]), m'''
      | _ -> raise NonValide
    end
  | _ -> raise NonValide
```

♣. En remplaçant les parenthèses ouvrantes par `O` et les fermantes par `F` cette grammaire est celle dont la propriété 2.56 affirme la non ambiguïté.

♡. ce choix de symbole est justifié par le fait que l'objectif de cet exemple est de fournir une implémentation en OCaml d'un parseur pour cette grammaire

```
42
43 (** Teste si le mot [m] est généré par la grammaire. *)
44 let est_genere (m: mot) : bool =
45   try
46     let arbre, reste = parse_s m
47     in reste = []
48   with
49   | NonValide -> false
```