Chapitre cinq

Langages algébriques

MPI/MPI*, lycée Faidherbe

Résumé

On a étudié les langages rationnels qui sont reconnaissables par une structure simple, les automates. Cependant des langages aussi simples et importants que le langage des mots qui contiennent autant de lettre a que de lettre b n'est pas rationnel. Nous allons étudier une classe plus importante de langages qui auront besoin de techniques plus sophistiquées.

I Dérivations

I.1 Définition

Un exemple

On commence par décrire une définition possible d'une formule sous forme normale conjonctive :

- Une formule est une clause parenthésée ou la conjonction d'une clause parenthésée et d'une formule.
- Une clause parenthésée est une clause entre parenthèses.
- Une clause est un littéral ou la disjontion d'un littéral et d'une clause.
- Un littéral est une variable ou la négation d'une variable.
- Les variables sont des objets de base.

On a ici exclu les clauses et les formules vides.
On note F une formule, CP une clause parenthésée, C une clause, L un littéral et V une variable alors sur l'ensemble de variables $\{x_1, x_2, x_3, x_4\}$, on peut donner les règles de construction

- $V \to x_1|x_2|x_3|x_4$
- $L \to V|\neg V$
- $C \to L|L \vee C$
- $CP \to (C)$
- $F \to CP|CP \wedge F$.

Par exemple la première ligne signifie qu'on peut remplacer un symbole V par l'un des mots x_1, x_2, x_3 ou x_4.
On peut créer la formule $(x_1 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_4)$ en appliquant les règles de construction :

$$\begin{aligned}
&F \to CP \wedge F \to (C) \wedge F \to (L \vee C) \wedge F \to (V \vee C) \wedge F \to (x_1 \vee C) \wedge F \\
&\quad \to (x_1 \vee L) \wedge F \to (x_1 \vee \neg V) \wedge F \to (x_1 \vee \neg x_3) \wedge F \\
&\quad \to (x_1 \vee \neg x_3) \wedge CP \to (x_1 \vee \neg x_3) \wedge (C) \to (x_1 \vee \neg x_3) \wedge (L \vee C) \\
&\quad \to (x_1 \vee \neg x_3) \wedge (V \vee C) \to (x_1 \vee \neg x_3) \wedge (x_2 \vee C) \\
&\quad \to (x_1 \vee \neg x_3) \wedge (x_2 \vee L \vee C) \to (x_1 \vee \neg x_3) \wedge (x_2 \vee V \vee C) \to (x_1 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee C) \\
&\quad \to (x_1 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee L) \to (x_1 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg V) \\
&\quad \to (x_1 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_4)
\end{aligned}$$

On va formaliser les opérations vues ici.

Grammaire

Définition 1

Une *grammaire hors-contexte*est un quadruplet $G = (\Sigma, V, P, S)$ où :

- Σ (parfois noté T) est un alphabet de symboles dits *terminaux* ;
- V est un alphabet dit de *variables* tel que $\Sigma \cap V = \emptyset$;
- $P \subset V \times (\Sigma \cup V)^*$ est un ensemble dit de *règles de production* ;
- $S \in V$ est le *symbole de départ*.

Une règle de production (X, α) est souvent notée $X \to \alpha$. Si (X, α) et (X, β) sont deux règles de production, on notera $X \to \alpha \mid \beta$.

On dit aussi grammaire non contextuelle.
Dans l'exemple ci-dessus,

- $\Sigma = \{x_1, x_2, x_3, x_4, (,), \neg, \wedge, \vee\}$,
- $V = \{F, CP, C, L, V\}$,
- les règles de production sont ce qui a été appelé les règle de construction,
- F est le symbole de départ.

L'usage est d'employer des lettres majuscules pour les symboles de V et des lettres grecques (α, β, γ, ...) pour désigner des mots de $(\Sigma \cup V)^*$.
Le qualificatif de "hors contexte" signifie que les règles s'appliquent à une lettre de Vdans un mots de $(\Sigma \cup V)^*$ indépendamment des lettres qui l'entourent.

Définition 2

Un mot $\alpha \in (\Sigma \cup V)^*$ se *dérive immédiatement* en le mot $\beta \in (\Sigma \cup V)^*$ s'il existe une règle $X \to \gamma$ et deux mots $\alpha', \alpha'' \in (\Sigma \cup V)^*$ tels que $\alpha = \alpha' X \alpha''$ et $\beta = \alpha' \gamma \alpha''$.
Si α se dérive immédiatement en β, on le note $\alpha \Rightarrow \beta$.

Exercice 1 **Solution page 10**
Prouver que si on a $\alpha \Rightarrow \beta$ alors, pour tous $\gamma, \delta \in (\Sigma \cup V)^*$, $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$.

Langage algébrique

La relation de dérivation peut se prolonger

Définition 3

Un mot $\alpha \in (\Sigma \cup V)^*$ se *dérive* en le mot $\beta \in (\Sigma \cup V)^*$, que l'on note $\alpha \overset{*}{\Longrightarrow} \beta$, si $\alpha = \beta$ ou s'il existe une suite finie $(\beta_0, \beta_1, \ldots, \beta_n)$ de mots de $(\Sigma \cup V)^*$ tels que
$\beta_0 = \alpha$, $\beta_n = \beta$ et $\beta_{i-1} \Rightarrow \beta_i$ pour tout $i \in \{1, 2, \ldots, n\}$.
$\alpha \Rightarrow \beta_1 \Rightarrow \beta_2 \Rightarrow \cdots \Rightarrow \beta_{n-1} \Rightarrow \beta$ est une *dérivation* de α en β.
Une dérivation avec n dérivation immédiates, comme ci-dessus, est aussi notée $\alpha \overset{n}{\Longrightarrow} \beta$.

$\overset{*}{\Longrightarrow}$ est la clôture réflexive et transitive de $\Rightarrow$.

Exercice 2 **Solution page 10**
Prouver que si on a $\alpha \overset{*}{\Longrightarrow} \beta$ alors, pour tous $\gamma, \delta \in (\Sigma \cup V)^*$, $\gamma\alpha\delta \overset{*}{\Longrightarrow} \gamma\beta\delta$.

Définition 4

Pour $G = (\Sigma, V, P, S)$ une grammaire hors-contexte, on note $L(G) = \{u \in \Sigma^* \ ; \ S \overset{*}{\Longrightarrow} u\}$, c'est-à-dire l'ensemble des mots de Σ^* pour lesquels il existe une dérivation de S en u, on dit que G génère u.
Si $L = L(G)$, on dit que G *engendre* L.
Un langage est *algébrique* s'il est le langage d'une grammaire hors-contexte.

On note que les mots de $L(G)$ ne contiennent pas de variables.

Définition 5

Soient G_1 et G_2 deux grammaires hors-contexte.
On dit que G_1 et G_2 sont *faiblement équivalentes* si $L(G_1) = L(G_2)$.

I.2 Exemples

Un langage non rationnel

La grammaire $G = \big(\{a, b\}, \{S\}, \{S \to aSb|\varepsilon\}, S\big)$ engendre le langage $L = \{a^n b^n \ ; \ n \in \mathbb{N}\}$.

- On montre par récurrence sur n que $S \overset{*}{\Longrightarrow} a^n b^n$.
  Pour $n = 0$, on a $S \Rightarrow \varepsilon = a^0 b^0$.
  Si on a une dérivation $S \overset{*}{\Longrightarrow} a^n b^n$ alors on peut écrire la dérivation $S \Rightarrow aSb \overset{*}{\Longrightarrow} aa^n b^n b = a^{n+1} b^{n+1}$.
- Inversement on montre par récurrence sur n que si $S \Rightarrow^{n+1} \alpha$ avec $\alpha \in (\Sigma \cup V)^*$, alors $\alpha = a^n b^n$ ou $\alpha = a^{n+1} S b^{n+1}$. Ainsi tous les mots de $L(G)$ appartiennent à L.

Mots bien parenthésés

On considère les parenthésages possibles en omettant les expressions parenthésées.
Par exemple `()`, `((()))`, `(()())` ou `(((()(())))())`.
Pour faciliter la lecture, on remplace `(` par a et `)` par b.
Le langage correspondant, le langage de Dyck, LD est celui des mots $u \in \{a, b\}^*$ tels que

- $|u|_a = |u|_b$, il y a autant de parenthèses ouvrantes que de parenthèses fermantes,
- pour tout préfixe v de u, $|v|_a \geqslant |v|_b$, toute parenthèse doit être ouverte avant d'être fermée.

Soit $u = x_1 x_2 \cdots x_n$ un mot de Dyck, $u \in LD$.
On note $\kappa(x) = 1$ si $x = a$ et $\kappa(x) = -1$ si $x = b$ et, pour $0 \leqslant i \leqslant n = |u|$,
$\alpha_u(i) = \sum_{k=1}^{i} \kappa(x_k) = |v|_a - |v|_b$ si $v = x_1 x_2 \cdots x_i$ est le préfixe de u de longueur i.
La définition ci-dessus peut se traduire par $\alpha_u(n) = 0$ et $\alpha_u(i) \geqslant 0$ pour tout i.
On a aussi $\alpha_u(0) = 0$ et $\alpha_u(p+1) = \alpha_u(p) + \kappa(x_{p+1})$.

Lemme : si $\alpha_u(p) = 0$ alors $x_p = b$ pour $p > 0$ et $x_{p+1} = a$ pour $p < n$.
Démonstration : pour $p > 0$, $0 \leqslant \alpha_u(p-1) = \alpha_u(p) - \kappa(x_p)$ donc $\kappa(x_p) \leqslant 0 : x_p = b$,
pour $p < n$, $0 \leqslant \alpha_u(p+1) = \alpha_u(p) + \kappa(x_{p+1})$ donc $\kappa(x_{p+1}) \geqslant 0 : x_{p+1} = a$. . . . . ∎

En particulier $\alpha_u(0) = 0$ implique $x_1 = a$ et $\alpha_n(0) = 0$ implique $x_n = b$

- Pour $u \neq \varepsilon$ il existe au moins un entier $k > 0$ tel que $\alpha_u(k) = 0 : k = n$ convient.
  Soit p le plus petit entier $k > 0$ tel que $\alpha_u(k) = 0$; on pose $v = x_2 \cdots x_{p-1}$ et $w = x_{p+1} \cdots x_n$, on a $u = a \cdot v \cdot b \cdot w$, v ou w peut être le mot vide.

- On peut écrire $v = y_1 y_2 \cdots y_{p-1}$ avec $y_i = x_{i+1}$.
  Alors $\alpha_v(i) = \alpha_u(i+1) - \kappa(x_1) = \alpha_u(i+1) - 1 \geqslant 0$ car $\alpha_u(k) \neq 0$ pour $1 \leqslant k < p$.
  De plus $\alpha_v(p-2) = \alpha_u(p-1) - 1 = \alpha_u(p) - \kappa(x_p) - 1 = \alpha_u(p) = 0$ car $x_p = b$.
  Ainsi $v \in LD$.
- On peut écrire $w = z_1 z_2 \cdots z_{n-p}$ avec $z_i = x_{i+p}$.
  Alors $\alpha_w(i) = \alpha_u(i+p) - \alpha_u(p) = \alpha_u(i+p) \geqslant 0$ et $\alpha_w(n-p) = \alpha_u(n) = 0$.
  Ainsi $w \in LD$.

Tout mot $u \in LD \setminus \{\varepsilon\}$ peut donc s'écrire $u = a \cdot v \cdot b \cdot w$ avec $v, w \in LD$.

On peut alors considérer la grammaire G de règle $S \to aSbS|\varepsilon$; G génère ε.
Supposons que G génère tous les mots de LD de longueur $2n$ au plus.
Soit $u \in LD$ avec $|u| = 2n+2$. D'après le résultat ci-dessus, on peut écrire $u = avbw$ avec $v, w \in LD$. De plus $|u| = 2 + |v| + |w|$ donc $|v| \leqslant 2n$ et $|w| \leqslant 2n$. D'après l'hypothèse de récurrence, il existe des dérivations $S \overset{*}{\Longrightarrow} v$ et $S \overset{*}{\Longrightarrow} v$. L'exercice 1 permet alors de construire la dérivation $S \Rightarrow aSbS \overset{*}{\Longrightarrow} avbS \overset{*}{\Longrightarrow} avbw = u$.
Par récurrence, on a prouvé que tout mot de LD est généré par la grammaire.

Inversement si u est généré par la grammaire alors chaque dérivation immédiate ajoute un a et un b avec b placé après a. u contient donc autant de a que de b et chaque a est associé à un b d'indice supérieur donc u est bien parenthésé.
G engendre LD.

II Langages rationnels

II.1 Propriétés des langages algébriques

Théorème 1 Stabilités des langages algébriques

Si L_1 et L_2 sont des langages algébriques alors $L_1 \cup L_2$, $L_1 \cdot L_2$ et L_1^* sont algébriques.

Démonstration Soient $G_1 = (\Sigma, V_1, P_1, S_1)$ et $G_2 = (\Sigma, V_2, P_2, S_2)$ des grammaires engendrant respectivement L_1 et L_2. On suppose que $V_1 \cap V_2 = \emptyset$, on renommera les variables si besoin.

- $L_1 \cup L_2$ est algébrique car il est engendré par la grammaire $(\Sigma, V_1 \cup V_2 \cup \{S\}, P, S)$ où $P = P_1 \cup P_2 \cup \{S \to S_1 | S_2\}$.
- $L_1 \cdot L_2$ est algébrique car il est engendré par la grammaire $(\Sigma, V_1 \cup V_2 \cup \{S\}, P, S)$ où $P = P_1 \cup P_2 \cup \{S \to S_1 S_2\}$.
- L_1^* est algébrique car il est engendré par la grammaire $(\Sigma, V_1 \cup \{S\}, P, S)$ où $P = P_1 \cup \{S \to S_1 S \mid \varepsilon\}$. ∎

Exercice 3 **Solution page 10**
Prouver que les langages élémentaires sont algébriques.

Théorème 2

Tout langage rationnel est algébrique. algébriques.

Démonstration On montre le résultat par induction structurelle en utilisant les deux résultats ci-dessus. ∎

Le résultat ci-dessus ne donne pas directement une grammaire pour un langage rationnel mais on peut en calculer une en partant d'une expression régulière dénotant le langage et en utilisant les construction du théorème 1.
Cependant le résultat peut être rapidement lourd.

Exemple

On considère le langage L des mots finissant par la lettre a.
Il peut être dénoté par $\mathtt{r} = (\mathtt{a}|\mathtt{b})*\mathtt{a}$.

- $(\Sigma, \{S_1\}, \{S_1 \to a\}, S_1)$. engendre $L[\mathtt{a}]$.
- $(\Sigma, \{S_2\}, \{S_2 \to b\}, S_2)$. engendre $L[\mathtt{b}]$.
- $(\Sigma, \{S_1, S_2, S_3\}, \{S_1 \to a, S_2 \to b, S_3 \to S_1|S_2\}, S_3)$. engendre $L[\mathtt{a}|\mathtt{b}]$.
- $(\Sigma, \{S_1, S_2, S_3, S_4\}, \{S_1 \to a, S_2 \to b, S_3 \to S_1|S_2, S_4 \to \varepsilon|S_3S_4\}, S_2)$ engendre $L[(\mathtt{a}|\mathtt{b})*]$.
- $(\Sigma, \{S_5\}, \{S_5 \to a\}, S_5)$. engendre $L[\mathtt{a}]$.
- $(\Sigma, \{S_1, S_2, S_3, S_4, S_5, S_6\}, P, S_6)$ engendre $L[(\mathtt{a}|\mathtt{b})*\mathtt{a}]$
  avec $P = \{S_1 \to a, S_2 \to b, S_3 \to S_1|S_2, S_4 \to \varepsilon|S_3S_4, S_5 \to a, S_6 \to S_4S_5\}$.

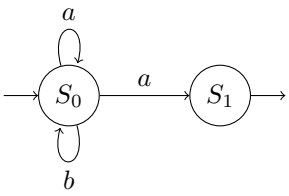
II.2 Grammaire associée à un automate

Le résultat ci-dessus est vite pléthorique. Il existe des procédés de simplification des grammaires, qui sont hors programme. On a déjà cherché des automates simples ; il peut être utile de déterminer une grammaire à partir d'un automate.
On notera ici $\mathbb{S} = \{S_0, S_1, \ldots, S_{n-1}\}$ l'ensemble des états de l'automate et suppose que S_0 est le seul état initial.

Définition 6

Si $Q = \big(\Sigma, \mathbb{S}, \Delta, \{S_0\}, T\big)$ est un automate, la *grammaire associée* est $G(Q) = \big(\Sigma, \mathbb{S}, P, S_0\big)$ avec $P = \{S_j \to \varepsilon \ ; \ S_j \in T\} \cup \big\{S_i \to xS_j \ ; S_j \in \Delta(S_i, x)\big\}$.

Si on revient à l'exemple du langage des mots finissant par a, il est reconnu par l'automate ci-contre. On lui associe la grammaire $\big(\{a, b\}, \{S_0, S_1\}, P, S_0\big)$ avec $P = \{S_0 \to aS_0|bS_0|aS_1, S_1 \to \varepsilon\}$.



Théorème 3

Le langage engendré par la grammaire associée à un automate est le langage reconnu par l'automate : $L\big(G(Q)\big) = L(Q)$.

Démonstration

- Si $u = x_1x_2\cdots x_n \in L(Q)$, il ;existe un calcul réussi
  $S_0 \xrightarrow{x_1} E_1 \xrightarrow{x_2} \cdots \xrightarrow{x_{n-1}} E_{n-1} \xrightarrow{x_n} E_n$ avec $E_i \in \mathbb{S}$ et $E_n \in T$.
  On a alors $u \in L\big(G(Q)\big)$ car on peut définir la dérivation
  $S_0 \Rightarrow x_1E_1 \Rightarrow x_1x_2E_2 \Rightarrow \cdots \Rightarrow x_1\cdots x_{n-1}E_{n-1} \Rightarrow x_1\cdots x_{n-1}x_nE_n \Rightarrow x_1\cdots x_n\varepsilon = u$.
- Inversement si $u = x_1x_2\cdots x_n \in L\big(G(Q)\big)$ on a une dérivation $S_0 \Rightarrow \alpha_1 \Rightarrow \cdots \Rightarrow \alpha_{p-1} \Rightarrow u$
  On montre alors que $\alpha_{p-1} = uE_{p-1}$ avec $E_{p-1} \in T$ puis $\alpha_{p-2} = x_1\cdots x_{n-1}E_{p-2}$ et (E_{p-2}, x_n, E_{p-1}) transition de Q. On remonte ainsi jusqu'à S_0 avec des transitions $(E_{p-n+i-2}, x_i, E_{p-n+i-1})$. On en déduit un calcul réussi pour u donc $u \in L(Q)$.

Les deux inclusions donnent l'égalité des langages. . . . . ∎

III Arbre de dérivation

III.1 Définition

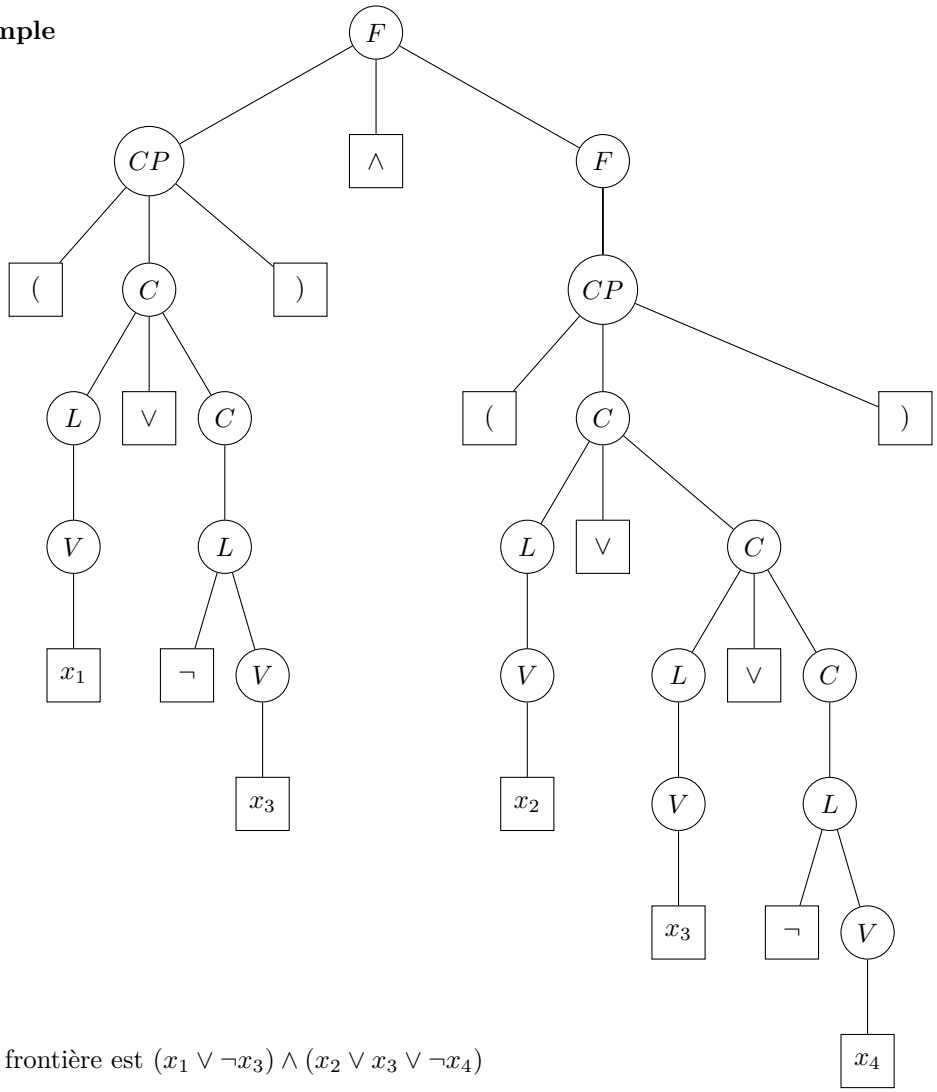
Définition 7

Un *arbre de dérivation* pour une grammaire $G = (\Sigma, V, P, S)$ est un arbre général dont les fils sont ordonnés tel que

- les nœuds sont étiquetés par des lettres de V,
- les feuilles sont étiquetées par des lettres de Σ ou par ε,
- la racine est étiquetée par S,
- si les fils d'un nœud d'étiquette X sont $\alpha_1, \alpha_2, \ldots, \alpha_p$, alors $X \to \alpha_1\alpha_2\cdots\alpha_p$ est une règle de production,
- une feuille d'étiquette ε est le seul fils de son père.

Le mot u obtenu par la lecture des feuilles de gauche à droite est la *frontière* de l'arbre.

Exemple



La frontière est $(x_1 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_4)$

Théorème 4

Les frontières des arbres de dérivation pour une grammaire G sont les mots de $L(G)$.

Démonstration On considère un arbre de dérivation ; chaque nœud est associé à une règle de production. Si on part de S et qu'on applique les règles de production associées aux nœuds dans l'ordre de parcours en largeur de l'arbre, on aboutit à la frontière de l'arbre. Ainsi la frontière est générée par la grammaire.
Inversement une dérivation d'un mot de $L(G)$ permet de construire un arbre à partir de la racine S ; chaque dérivation immédiate définit les fils d'un nœud et, àla fin de la dérivation, il n'y a plus de nœud et le mot généré est la frontière de l'arbre. . . . . ∎

L'arbre de dérivation associé à une dérivation est unique mais il y a souvent plusieurs dérivation qui sont portées par un arbre. Elles diffèrent par l'ordre des dérivations.

Définition 8

Si $\alpha \in (\Sigma \cup V)^*$, on écrit $\alpha = uX\beta Yv$ avec $u, v \in \Sigma^*$, $X, Y \in V$ et $\beta \in (\Sigma \cup V)^*$.
Une dérivation immédiate *droite* (resp. *gauche*) est une dérivation obtenue par une règle de production $X \to \gamma$ (resp. $Y \to \gamma'$) : $uX\beta Yv \Rightarrow u\gamma\beta Yv$ (resp. $uX\beta Yv \Rightarrow uX\beta\gamma' v$).
Une suite de dérivations immédiates gauches (resp. droites) est appelée une *dérivation gauche* (resp. *droite*), notée par $\overset{*}{\Longrightarrow}_g$ (resp. $\overset{*}{\Longrightarrow}_d$).

III.2 Ambiguïté

Des dérivations différentes peuvent être considérées comme semblables si elles définissent le même arbre de dérivation. Mais il peut exister des dérivations qui ont des arbres de dérivations différents.

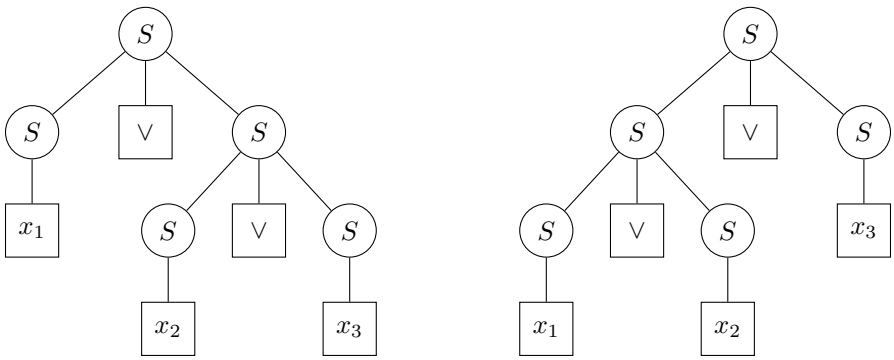
Définition 9

Soit $G = (\Sigma, V, P, S)$ une grammaire.
Un mot $u \in L(G)$ est *ambigu pour* G s'il existe plusieurs arbres de dérivation de u.
La grammaire G est **ambiguë** s'il existe des mots ambigu pour G.

L'ambiguïté est celle de la grammaire, il est parfois possible de trouver une grammaire non ambiguë qui engendre la même langage qu'une grammaire ambiguë : une grammaire faiblement équivalente.

Exemple simple

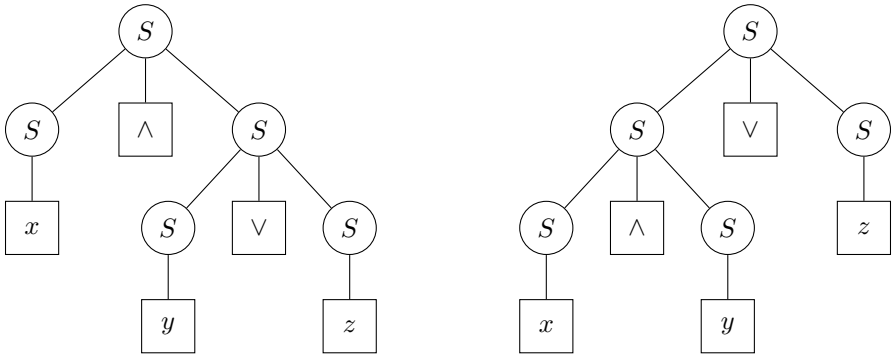
En reprenant l'exemple initial en se restreignant aux clauses positives, on peut les définir par la grammaire $\big(\{x_1, x_2, x_3, x_4, \vee\}, \{S\}, \{S \to S \vee S | x_1, x_2, x_3, x_4\}, S\big)$.
Dans ce cas $x_1 \vee x_2 \vee x_3$ admet deux arbres de dérivation



Ici l'ambiguïté reflète l'associativité. On peut la lever en imposant l'ordre de composition avec la grammaire $\big(\{x_1, x_2, x_3, x_4, \vee\}, \{S, V\}, \{S \to V | V \vee S, V \to x_1, x_2, x_3, x_4\}, S\big)$.

Problème de priorité

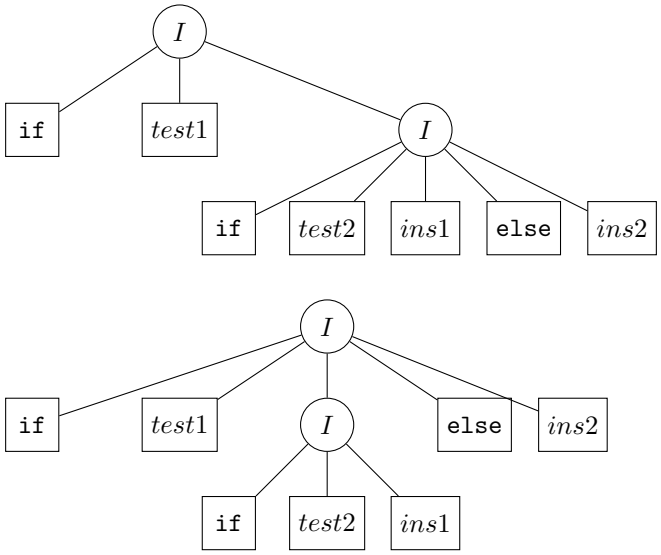
Le langage des formules booléennes simples à 3 variables x, y et z est engendré par la grammaire $\big(\{x, y, z, \vee, \wedge, \neg\}, \{S\}, \{S \to S \vee S | S \wedge S | \neg S | x | y | z\}, S\big)$ (on admet ce résultat).
Dans cette grammaire $x \wedge y \vee z$ admet 2 arbres de dérivations



Ici le problème est plus important car la fonction booléenne associée n'est pas la même. Si on veut une grammaire non ambiguë on peut ajouter un parenthésage ou donner une priorité distincte aux opérations

Le problème des `else`

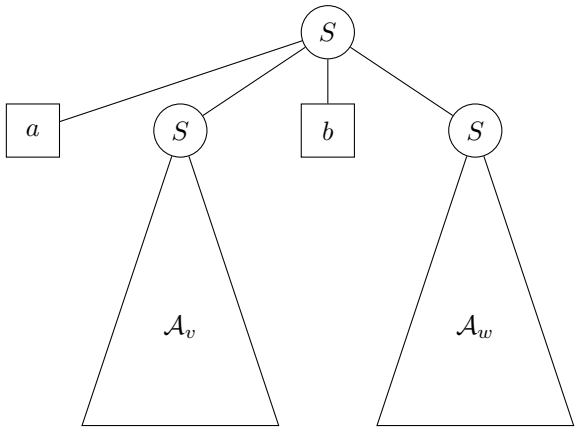
Les langages de programmation n'imposent pas toujours la présence d'un `else` après une un test `if`. L'absence de la partie `else` est traduite par "sinon ne rien faire". Cela signifie que, dans le langage OCaml, le bloc d'instructions qui suit la condition doit avoir un résultat `unit`.
Si on donne une partie de la grammaire définissant un langage on a les règles
$I \to \texttt{if}BI|\texttt{if}BI\texttt{else}I| \dots$ où B représente une expression booléenne et I un bloc d'instruction.
Alors la portion de mot $\texttt{if}(test_1)\texttt{if}(test2)ins1\texttt{else}ins2$ rend le mot total ambigu



Les instructions $ins2$ sont effectuées seulement si $test1$ et $test2$ sont tous deux faux ou si seulement $test1$ est faux ? La plupart des compilateurs choisissent le premier cas, ils attribuent le `else` aux dernier `if`. Il est recommandé d'ajouter des accolades ou des structures de bloc pour marquer son intention, les indentations ne sont pas prises en comptes.

Les mots bien parenthésés

La grammaire qui reconnaît l'ensemble LD des mots de Dyck, vue dans la partie I.2, est non ambiguë. Pour le prouver on montre par récurrence sur la demi-longueur d'un mot $u \in LD$ que u admet un unique arbre de dérivation (tous les mots de LD sont de longueur paire).
C'est vrai pour $u = \varepsilon$ qui ne peut admettre que $S \Rightarrow \varepsilon$ comme dérivation le générant.
On suppose que tous les mots de LD de longueur majorée par $2n$ admettent un unique arbre de dérivation dans G. On considère $u \in LD$ tel que $|u| = 2n + 2$.
On a vu qu'on peut écrire $u = avbw$ avec $u, v \in LD$.
On montre que cette décomposition est unique : si on a une autre décomposition $u = av'bw'$ avec $u', v' \in LD$. Alors av et av' sont des préfixes de u donc l'un est un préfixe de l'autre, par exemple av est préfixe de av', donc v est un préfixe de v'.
Si $v' \neq v$ alors, comme la lettre qui suit v dans u est b, vb est un préfixe de v'.
Or v contient autant de b que de a donc vb contient un b de plus que de a, ce qui est impossible pour un préfixe d'un mot de LD.
Ainsi on doit avoir $v = v'$ d'où $u = avbw = avbw'$ puis $w = w'$: la décomposition est unique.
Si $\mathcal{A}_v$ et $\mathcal{A}_w$ sont les arbres de dérivation de v et w respectivement, ils sont uniques par hypothèse de récurrence , et u ne peut admettre comme arbre de dérivation que



Solutions

Exercice 1
Si $\alpha \Rightarrow \beta$ alors il existe une règle $X \to \beta'$ et des décompositions $\alpha = \alpha' X \alpha''$ et $\beta = \alpha' \beta' \alpha''$.
On a alors les décompositions $\gamma\alpha\delta = \gamma\alpha' X \alpha''\delta$ et $\gamma\beta\delta = \gamma\alpha'\beta'\alpha''\delta$. Si on note $\gamma' = \gamma\alpha'$ et $\delta' = \alpha''\delta$, on a $\gamma\alpha\delta = \gamma' X \delta'$ et $\gamma\beta\delta = \gamma'\beta'\delta'$ et la règle $X \to \beta'$ donc $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$.

Exercice 2
On le prouve par récurrence sur le nombre de dérivations immédiates en appliquant l'exercice 1

Exercice 3

- $\emptyset$ est engendré par la grammaire $(\Sigma, \{S\}, \emptyset, S)$.
- $\{\varepsilon\}$ est engendré par la grammaire $(\Sigma, \{S\}, \{S \to \varepsilon\}, S)$.
- Pour $x \in \Sigma$, $\{x\}$ est engendré par la grammaire $(\Sigma, \{S\}, \{S \to x\}, S)$.