

Ambiguïté et analyse syntaxique

Quentin Fortier

October 7, 2025

Définition : Arbre de dérivation

Soient $G = (\Sigma, V, R, S)$ une grammaire et $u \in L(G)$. Un arbre de dérivation (ou : arbre syntaxique) pour u est un arbre tel que :

- la racine est étiquetée S
- chaque nœud interne est étiqueté par un élément de V
- chaque feuille est étiquetée par un élément de $\Sigma \cup \{\varepsilon\}$
- toute feuille étiquetée ε est fille unique
- si un nœud interne est étiqueté X et possède n fils étiquetés $\alpha_1, \dots, \alpha_n$, alors $X \longrightarrow \alpha_1 \dots \alpha_n \in R$

Remarque : les étiquettes des feuilles, lues de gauche à droite, forment le mot u .

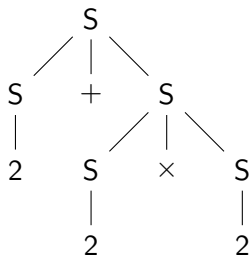
Arbre de dérivation

Soit G la grammaire définie par $S \rightarrow S + S \mid S \times S \mid 2$ avec $\Sigma = \{+, \times, 2\}$.

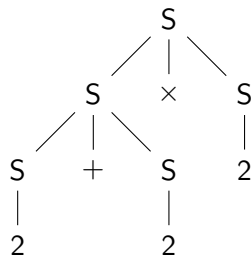
Il y a plusieurs façons d'engendrer $2 + 2 \times 2$, donnant des arbres de dérivation différents :

① $S \Rightarrow S + S \Rightarrow 2 + S \Rightarrow 2 + S \times S \Rightarrow 2 + 2 \times S \Rightarrow 2 + 2 \times 2$

② $S \Rightarrow S \times S \Rightarrow S + S \times S \Rightarrow 2 + S \times S \Rightarrow 2 + 2 \times S \Rightarrow 2 + 2 \times 2$



①



②

Définition : Grammaire ambiguë

Soit $G = (\Sigma, V, R, S)$ une grammaire.

- On dit que u est ambigu pour G s'il existe plusieurs arbres de dérivation distincts pour u .
- On dit que la grammaire G est ambiguë s'il existe au moins un mot u ambigu pour G .

Exemple : $S \rightarrow S + S \mid S \times S \mid 2$ est ambiguë.

Définition : Grammaire ambiguë

Soit $G = (\Sigma, V, R, S)$ une grammaire.

- On dit que u est ambigu pour G s'il existe plusieurs arbres de dérivation distincts pour u .
- On dit que la grammaire G est ambiguë s'il existe au moins un mot u ambigu pour G .

Exemple : $S \rightarrow S + S \mid S \times S \mid 2$ est ambiguë.

Exercice

Montrer que les grammaires suivantes sont ambiguës :

① $S \rightarrow S \mid \varepsilon$

② $S \rightarrow aXb, X \rightarrow a \mid b \mid \varepsilon \mid XX.$

Attention : Si $S \rightarrow SaS \mid b$, bab peut être engendré de deux façons ($S \Rightarrow SaS \Rightarrow baS \Rightarrow bab$ et $S \Rightarrow SaS \Rightarrow Sab \Rightarrow bab$) mais n'est pas ambigu (les deux arbres de dérivation sont les mêmes).

Grammaire ambiguë

Attention : Si $S \rightarrow SaS \mid b$, bab peut être engendré de deux façons ($S \Rightarrow SaS \Rightarrow baS \Rightarrow bab$ et $S \Rightarrow SaS \Rightarrow Sab \Rightarrow bab$) mais n'est pas ambigu (les deux arbres de dérivation sont les mêmes).

Définition : Dérivation gauche

Soit $G = (\Sigma, V, R, S)$ une grammaire.

Une dérivation gauche pour u est une dérivation où, à chaque étape, on remplace la variable la plus à gauche.

De même pour une dérivation droite, où on remplace la variable la plus à droite.

Exemple : $S \Rightarrow S + S \Rightarrow 2 + S \Rightarrow 2 + S \times S \Rightarrow 2 + 2 \times S \Rightarrow 2 + 2 \times 2$
est une dérivation gauche pour $2 + 2 \times 2$.

Théorème

Soient G une grammaire et $u \in L(G)$. Il y a une bijection entre les dérivations gauches de u et les arbres de dérivation de u .

Preuve :

Théorème

Soient G une grammaire et $u \in L(G)$. Il y a une bijection entre les dérivations gauches de u et les arbres de dérivation de u .

Preuve :

On associe à chaque arbre de dérivation une dérivation gauche en parcourant l'arbre dans le parcours préfixe.

D'où :

Théorème

Soit G une grammaire et $u \in L(G)$.

u est ambigu pour G si et seulement si u possède plusieurs dérivations gauches.

Exercice

Montrer qu'un langage régulier est engendré par une grammaire non ambiguë.

Définition : Faible équivalence

Deux grammaires G_1 et G_2 sont dites faiblement équivalentes si $L(G_1) = L(G_2)$.

Remarque : le terme « faiblement » vient du fait qu'on ne demande que l'égalité sur les langages et pas sur les arbres de dérivation. Ainsi, une grammaire ambiguë peut être faiblement équivalente à une grammaire non ambiguë.

Il peut être utile de chercher à trouver une grammaire équivalente non ambiguë.

Exemple : la grammaire définie par $S \rightarrow S + S \mid S \times S \mid (S) \mid 2$ est ambiguë mais on admet qu'elle est faiblement équivalente à la grammaire non-ambiguë suivante :

$$S \rightarrow S + T \mid T$$

$$T \rightarrow T \times F \mid F$$

$$F \rightarrow (S) \mid 2$$

On force ici la priorité des opérations.

Remarques :

- Il est indécidable de savoir si une grammaire est ambiguë, c'est-à-dire qu'il n'existe pas d'algorithme qui, étant donnée une grammaire, détermine si elle est ambiguë.
- Un langage non-contextuel qui ne peut être engendré que par une grammaire ambiguë est dit intrinsèquement ambigu.

Dangling else

Il y a deux façons d'interpréter le code suivant :

```
if (...)
    if (...)
        ...
    else
        ...
```

```
if (...)
    if (...)
        ...
else
    ...
```

La grammaire $S \rightarrow \text{if } S \text{ else } S \mid \text{if } S \mid \dots$ est ambiguë car `if ... if ... else ...` peut être dérivé de deux façons.

En C, on résout l'ambiguïté en associant le `else` au `if` le plus proche. On peut aussi modifier la grammaire pour la rendre non ambiguë (exemple : ajouter des accolades).

La compilation d'un code source permet de passer d'un langage à un autre et comporte deux grandes étapes :

- 1 Analyse lexicale : découpe le code source en une liste de lexèmes (mots-clés, identifiants, opérateurs...) en vérifiant que le code est bien formé.

Utilise un automate fini déterministe.

- 2 Analyse syntaxique (*parsing*) : construit l'arbre de dérivation du code source.

Utilise un algorithme *top-down* ou *bottom-up*.

Exemple de création d'un langage de programmation simple en OCaml (voir aussi [ce cours de compilation](#)).

Analyse syntaxique

Une expression arithmétique postfixe (ou : polonaise inverse) consiste à écrire les opérateurs après les opérandes : $34 + 5 \times$ au lieu $(3 + 4) \times 5$.

Grammaire (non ambiguë) :

Analyse syntaxique

Une expression arithmétique postfixe (ou : polonaise inverse) consiste à écrire les opérateurs après les opérandes : $34 + 5 \times$ au lieu $(3 + 4) \times 5$.

Grammaire (non ambiguë) :

$$S \rightarrow SSB \mid n \in \mathbb{N}$$

$$B \rightarrow + \mid \times \mid -$$

On simplifiera l'écriture d'un arbre syntaxique :



On utilise le type suivant :

```
type lexeme = I of int | B of char
type arbre = F of int | N of lexeme * arbre * arbre
```

Ainsi, $34 + 5 \times$ est représenté par :

- la liste de lexèmes [I 3; I 4; B '+'; I 5; B '*']
- l'arbre N (B '*', N (B '+', F 3, F 4), F 5).

Exercice

Écrire une fonction `parse : lexeme list -> arbre` qui prend une liste de lexemes et renvoie l'arbre de dérivation correspondant.

Exercice

Écrire une fonction `parse : lexeme list -> arbre` qui prend une liste de lexemes et renvoie l'arbre de dérivation correspondant.

On parcourt la liste de lexèmes en utilisant une pile pour stocker les arbres en cours de construction.

- Si on rencontre un entier k , on ajoute un arbre feuille $F\ k$ à la pile.
- Si on rencontre un opérateur $+$, on extrait les deux arbres $a1$ et $a2$ de la pile et on ajoute l'arbre $N(+, a1, a2)$.
- Quand on a fini de parcourir la liste, le seul arbre restant dans la pile est l'arbre de dérivation.