Ambiguïté et analyse syntaxique

Quentin Fortier

October 3, 2024

Arbre de dérivation

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 $\alpha_1, \ldots, \alpha_n$, alors $X \longrightarrow \alpha_1 \ldots \alpha_n \in R$

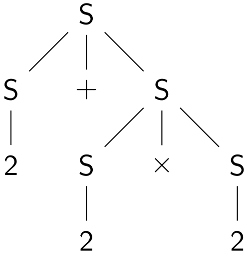
Remarque : les étiquettes des feuilles, lues de gauche à droite, forment le mot u.

Arbre de dérivation

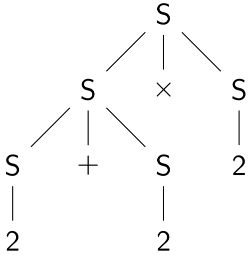
Soit G la grammaire définie par $S \to S + S \mid S \times S \mid 2$ avec $\Sigma = \{+, \times, 2\}$.

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

Grammaire ambiguë

Définition : Grammaire ambigüe

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 : $S \to S + S \mid S \times S \mid 2$ est ambigü.

Grammaire ambiguë

Définition : Grammaire ambigüe

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 : $S \to S + S \mid S \times S \mid 2$ est ambigü.

Exercice

Montrer que les grammaires suivantes sont ambiguës :

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

Grammaire ambiguë

Attention : Si $S \to SaS \mid b$, 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 ambigü (les deux arbres de dérivation sont les mêmes).

Grammaire ambiguë

Attention : Si $S \to SaS \mid b$, 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 ambigü (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 $2 + 2 \times 2$.

Grammaire ambiguë

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 :

Grammaire ambiguë

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 $u \in L(G)$.
G est ambiguë si et seulement si u possède plusieurs dérivations gauches.

Grammaire 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.

Grammaire ambiguë

Il peut être utile de chercher à éliminer l'ambiguïté d'une grammaire.

Exemple : la grammaire définie par $S \to S + S \mid S \times S \mid (S) \mid 2$ est ambiguë mais est faiblement équivalente à la grammaire non-ambiguë suivante :

$$\begin{aligned} S &\to S + T \mid T \\ T &\to T \times T \mid F \\ F &\to (S) \mid 2 \end{aligned}$$

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

Dangling else

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

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

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

La grammaire $S \to$ `if` S `else` S |`if` S |... 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).

Analyse syntaxique

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

Grammaire (non ambiguë) :

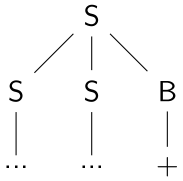
Analyse syntaxique

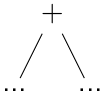
Une expression arithmique postfixe (ou : polonaise inverse) consiste à écrire les opérateurs après les opérandes : $34 + 5\times$ au lieu $(3+4)\times 5$.

Grammaire (non ambiguë) :

$$S \to SSB \mid n \in \mathbb{N}$$
$$B \to + \mid \times \mid -$$

On simplifiera l'écriture d'un arbre syntaxique :





Analyse syntaxique

On utilise le type suivant :

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

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

- la liste de tokens `[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 : token list -> arbre` qui prend une liste de tokens et renvoie l'arbre de dérivation correspondant.

Analyse syntaxique

Exercice

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

On parcourt la liste de tokens 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.