Langages et compilation

Analyse syntaxique

Exercice 1.
On considère le vocabulaire $V = \{a, b, c\}$. Proposez une grammaire décrivant chacun des quatre langages suivants :

1. $L1 = a^*.b.c^*$
2. $L2 = \{a^n.b.c^n \mid n \geq 0\}$
3. $L3 = \{a^n.b.c^m \mid 0 < n < m\}$
4. $L4 = \{w \mid w \text{ est un palindrome}\}$

Exercice 2.
On souhaite définir une grammaire G destinée à décrire des *expressions arithmétiques* construites à partir des opérateurs binaires soustraction $(-)$ et multiplication $(*)$ et dont les opérandes sont des entiers (e).

1. On considère tout d'abord la grammaire G_0 dont les productions sont les suivantes :

$$\begin{aligned} Z &\longrightarrow E \\ E &\longrightarrow E - E \\ E &\longrightarrow e \end{aligned}$$

   En prenant pour exemple la séquence "10 - 2 - 3", montrez que cette grammaire est *ambiguë*.
2. Pour éliminer l'ambiguité de G_0, on considère la grammaire G_1 suivante :

$$\begin{aligned} Z &\longrightarrow E \\ E &\longrightarrow E - T \\ E &\longrightarrow T \\ T &\longrightarrow e \end{aligned}$$

   Dessinez l'arbre de dérivation correspondant à l'exemple de la question précédente. On dit ici que la soustraction est "associative à gauche".
3. Comment faut-il modifier G_1 pour introduire un opérateur de multiplication tel que :
   – la multiplication soit associative à gauche
   – la multiplication soit plus prioritaire que la soustraction
   Justifiez votre réponse en construisant les arbres syntaxiques des expressions suivantes : "$10 - 2 * 3$", "$10 * 2 - 3$".
4. On veut ajouter la possibilité de mettre une sous-expression entre parenthèses, pour pouvoir écrire par exemple : "$(10 - 2) * 3$" ou "$10 * (2 - 3)$". Modifiez la grammaire obtenue précédemment. Construisez les arbres syntaxiques des expressions données en exemple.
5. On ajoute un opérateur "moins unaire" (noté $-$), plus prioritaire que $-$ et $*$. Modifiez la grammaire précédente. Construisez l'arbre syntaxique de "$10 * -2 - -3$".

Exercice 3.
On considère la grammaire suivante qui décrit les instructions d'un langage de programmation :

$$\begin{aligned} Z &\longrightarrow I \\ I &\longrightarrow \text{if } e \text{ then } I \\ I &\longrightarrow \text{if } e \text{ then } I \text{ else } I \\ I &\longrightarrow a \end{aligned}$$

1. Montrez que cette grammaire est *ambigüe* : trouvez une phrase du langage qui admet deux arbres de dérivations distincts.
2. Proposez une solution pour rendre cette grammaire non ambiguë (par exemple en modifiant le langage décrit).
3. Est-il possible de rendre cette grammaire non ambiguë sans modifier le langage décrit ?