Université Joseph Fourier **Master Informatique**
UFR IM^{2}AG

Sémantique des Langages de Programmation et Compilation

Séances 1 et 2 : grammaires et syntaxe abstraite

Exercice 1.
Donnez des exemples de fragments de programmes Ada ou C ou de tout autre langage comportant respectivement une erreur :

1. lexicale
2. syntaxique
3. de sémantique statique
4. de sémantique dynamique

Exercice 2.
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 3.
On considère le langage régulier $L0 = a^*.b^*$. La grammaire $G0 = S \rightarrow A\ B; \quad A \rightarrow a\ A \mid \epsilon; \quad B \rightarrow b\ B \mid \epsilon$ où S est l'axiome décrit le langage $L0$. L'automate d'états fini $A0 = (Q, V, \delta, q0, F)$ permet de reconnaître ce langage.

$$\begin{array}{rcl} Q & = & \{q0, q1\} \text{ est l'ensemble d'états} \\ V & = & \{a, b\} \text{ est l'alphabet d'entrée} \\ q0 & \in & Q \text{ est l'état initial} \\ F & = & \{q1\} \text{ est l'ensemble des états accepteurs} \\ \delta & = & \{\ (q0, a, q0), (q0, \epsilon, q1), (q1, b, q1)\ \} \text{ est la relation de transition} \end{array}$$

1. donner une dérivation de la grammaire $G0$ pour la chaine *aaaabb*.
2. donner la succession des configurations de l'automate $A0$ permettant de reconnaître la chaine *aabb*.

Exercice 4.
On considère le langage hors-contexte $L1 = \{a^n.b^n \mid n \geq 0\}$. La grammaire $G1 = S \rightarrow a\ S\ b \quad S \rightarrow \epsilon$ où S est l'axiome décrit le langage $L1$. l'automate à pile $A1 = (Q, V, \Gamma, \Delta, Z, q0, F)$ permet de reconnaître ce langage.

$$\begin{array}{rcl} Q & = & \{q0, q1, q2, q3\} \text{ est l'ensemble d'états} \\ V & = & \{a, b\} \text{ est l'alphabet d'entrée} \\ \Gamma & = & \{Z, A\} \text{ est l'alphabet de la pile} \\ Z & \in & \Gamma \text{ est le symbole initial de pile} \\ q0 & \in & Q \text{ est l'état initial} \end{array}$$

$$
\begin{array}{rcl}
F & = & \{q3\} \text{ est l'ensemble des états accepteurs} \\
\Delta & = & \{\ (q0, a\ \omega, Z) \rightarrow (q1, \omega, Z\ A), \\
 & & (q1, a\ \omega, u) \rightarrow (q1, \omega, u\ A), \\
 & & (q1, b\ \omega, u\ A) \rightarrow (q2, \omega, u), \\
 & & (q2, b\ \omega, u\ A) \rightarrow (q2, \omega, u), \\
 & & (q2, \epsilon, Z) \rightarrow (q3, \epsilon, \epsilon)\ \} \\
 & & \Delta \text{ est l'ensemble des transitions, avec } \omega \in V^* \text{ et } u \in \Gamma^*
\end{array}
$$

1. donner la dérivation de la grammaire $G1$ pour la chaine *aabb*.
2. donner la succession des configurations de l'automate $A1$ permettant de reconnaître la chaine *aabb*.

Exercice 5.
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 :
   $Z \longrightarrow E \qquad E \longrightarrow E - E \qquad E \longrightarrow e$
   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 :
   $Z \longrightarrow E \qquad E \longrightarrow E - T \qquad E \longrightarrow T \qquad T \longrightarrow e$
   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 6.
On considère la grammaire suivante qui décrit des instructions d'un langage de programmation :

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

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 ?

Exercice 7.
Soit G une grammaire décrivant des affectations dans laquelle les terminaux `c` et `i` désignent respectivement une constante entière et un identificateur :

```
Z -> A ;              E -> E + T | T        F -> c | i | (E)
A -> i := E           T -> T * F | F
```

On considère les deux affectations suivantes (`x` et `y` sont des identificateurs) :
(1) $x := 5 + x * 2 + y;$ (2) $y := (5 + x) * 2 + y;$

1. Construisez les arbres de dérivation correspondant aux affectations (1) et (2).
2. On suppose que l'on dispose d'un processeur comportant des registres notés `Ri` et disposant des instructions suivantes :
   — `LD Ri,op` charge la valeur de `op` dans le registre `Ri`, `op` désigne une variable ou une constante.
   — `ST Ri,x` donne à `x` la valeur contenue dans le registre `Ri`, `x` désigne une variable.
   — `ADD Ri,op1,op2` range dans `Ri` la somme `op1+op2`,
   — `MULT Ri,op1,op2` range dans `Ri` le produit `op1*op2`,
   avec `op1` désigne un registre et `op2` désigne un registre ou une constante.
   Ecrivez les séquences d'instructions correspondant à la traduction de (1) et (2).
3. Nous souhaitons produire le code pour notre machine à partir de l'arbre de dérivation. A quel noeud de l'arbre peut-on associer chacune des instructions machine de la séquence produite dans la question précédente ? Décorez ainsi les arbres de dérivation de (1) et (2) avec les séquences d'instructions obtenues : chaque instruction doit décorer un noeud de l'arbre, certains noeuds ne sont pas décorés.
4. En vous inspirant du résultat de la question 3, dessinez pour les arbres (1) et (2) un arbre "simplifié" (i.e. arbre abstrait) ne comportant que les noeuds décorés. Proposez une syntaxe abstraite pour G (sous la forme d'une grammaire). Etudiez différentes façons d'écrire cette grammaire.

Exercice 8.
Les fragments de programme ci-dessous donnent respectivement des exemples d'instructions "for" en ADA et C :

```
for i in 1..N loop            for (i=0 ; i<N ; i++) {
   -- instructions               /* instructions */
end loop ;                    }
```

1. Proposez une syntaxe concrète (éventuellement très simplifiée) de l'instruction "for" pour chacun de ces deux langages.
2. Proposez une syntaxe abstraite correspondante.