Fiche de TD-TP no. 4

Analyse syntaxique : grammaires et APs

Exercice 1. Exercice 3.5 de Amadio.

Exercice 2.

1. Proposez une grammaire $\mathcal{G}$ telle que
$$L(\mathcal{G}) = \{\, a^n b^n \mid n \geq 0 \,\}.$$
2. Construisez une dérivation et un arbre de dérivation du mot $aabb \in L(\mathcal{G})$.
3. Proposez un AP $\mathcal{A}$ tel que
$$L(\mathcal{A}) = \{\, a^n b^n \mid n \geq 0 \,\}.$$

Exercice 3.

1. Proposez une grammaire $\mathcal{G}$ telle que
$$L(\mathcal{G}) = \{\, a^n b^n c^m d^m \mid n, m \geq 0 \,\} \cup \{\, a^n b^m c^m d^n \mid n, m \geq 0 \,\}.$$
2. Construisez une dérivation et un arbre de dérivation du mot *abbccd*.
3. Montrez que la grammaire proposée est ambiguë.

Exercice 4. Définissez, par une grammaire, la syntaxe des expressions régulières de lex.

Exercice 5. Exemples 4.4, et 4.5 de Amadio. Exercices 4.6, 4.7, 4.8 de Amadio.

En TP

Prise en main de l'outil yacc (bison)

Exercice 6. Modifiez le fichier `lexeur.lex` de l'exercice 8 (TP 1) et intégrez-le avec un fichier `parseur.y` qui fasse l'analyse syntaxique d'un fichier contenant un programme `lse`. On produira un programme `lseparseur` qui affichera à l'écran si un programme est syntaxiquement correcte ou non.

Exercice 7. Nous souhaitons controller un thermostat, par des suites des commandes. Ainsi, nous définissons formellement ce que veut dire « suite de commandes » par la grammaire suivante :

$$\begin{aligned} \mathit{commandes} &\rightarrow \epsilon \mid \mathit{commandes}\ \ \mathit{commande} \\ \mathit{commande} &\rightarrow \mathit{changer\_etat} \mid \mathit{choisir\_temp} \\ \mathit{changer\_etat} &\rightarrow \mathit{MOTCLES\_ETAT}\ \ \mathit{ETAT} \\ \mathit{chosir\_temp} &\rightarrow \mathit{MOTCLES\_TEMP}\ \ \mathit{NOMBRE} \end{aligned}$$

Écrivez un analyseur syntaxique permettant de dire si une chaîne de caractères est un commande.

Exercice 8. Pour cet exercice faite référence à la page web http://www.graphviz.org/doc/info/lang.html.
Écrivez un analyseur syntaxique permettant de dire si un fichier contient un description correcte d'un graphe dans le langage dot. La grammaire du langage est la suivante :

```
graph    :      [ strict ] (graph | digraph) [ ID ] '{' stmt_list '}'
stmt_list       :      [ stmt [ ';' ] [ stmt_list ] ]
stmt     :      node_stmt
         |      edge_stmt
         |      attr_stmt
         |      ID '=' ID
         |      subgraph
attr_stmt       :      (graph | node | edge) attr_list
attr_list       :      '[' [ a_list ] ']' [ attr_list ]
a_list   :      ID [ '=' ID ] [ ',' ] [ a_list ]
edge_stmt       :      (node_id | subgraph) edgeRHS [ attr_list ]
edgeRHS         :      edgeop (node_id | subgraph) [ edgeRHS ]
node_stmt       :      node_id [ attr_list ]
node_id         :      ID [ port ]
port     :      ':' ID [ ':' compass_pt ]
         |      ':' compass_pt
subgraph        :      [ subgraph [ ID ] ] '{' stmt_list '}'
compass_pt      :      (n | ne | e | se | s | sw | w | nw | c | _)
```

Exercice 9. Ajoutez des actions au fichier `parseur.y` de l'exercice 6 pour générer votre premier compilateur. Le programme affichera sur la sortie standard une traduction d'un programme `lse` dans le langage de la machine virtuelle.