Fiche de TD-TP no. 5

Grammaires

Exercice 1. Définissez, par une grammaire, la syntaxe des expressions régulières de lex.

Analyse descendante

Exercice 2. Exemple 4.5 de Amadio :

1. pour chaque production $A \to \alpha$, calculez $FIRST(\alpha)$,
2. construisez l'APD pour la grammaire $LL(1)$,
3. en utilisant l'APD, montrez que le mot $aabb\$$ appartient au langage de la grammaire.

Exercice 3. Exercices 4.6, 4.7, 4.8 de Amadio.

Analyse ascendante

Exercice 4. Considérez le fichier yacc suivant :

```
%token ID  LPAR RPAR
%token PLUS
%%
S:expr;
expr: ID | expr PLUS expr | LPAR expr RPAR;
%%
```

La compilation de ce fichier donne le résultat suivant :

```
>> bison expr.y
expr.y: conflits: 1 décalage/réduction
```

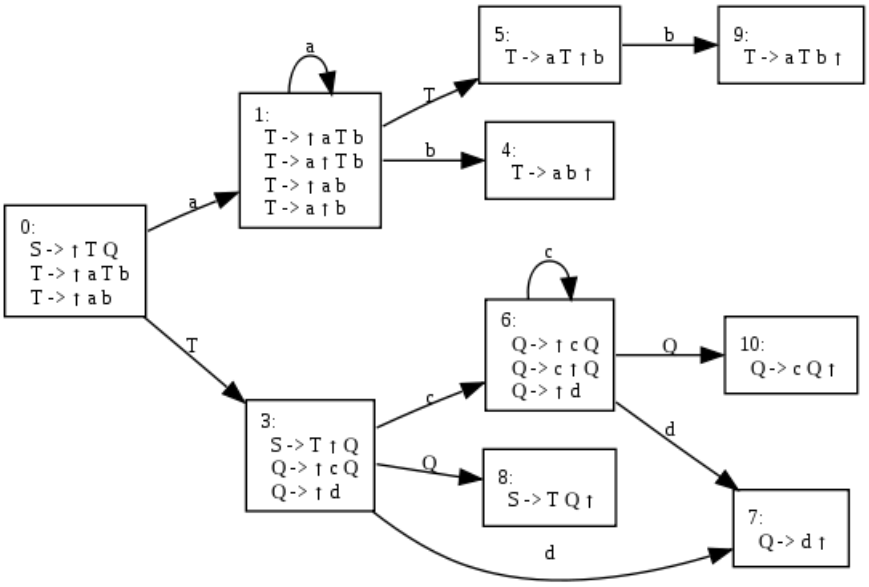
Expliquez pourquoi :

1. montrez que la grammaire est ambiguë,
2. simulez un calculs sur un mot ambigu,
3. montrez que le choix d'une alternative dans un conflit amène à construire deux arbres de dérivation différents pour ce mot.

Exercice 5. Considérez la grammaire suivante :

$$\begin{array}{rcl} S & \to & TQ \\ T & \to & aTb \mid ab \\ Q & \to & cQ \mid d \end{array}$$

L'AFD des items est le suivant :



Simulez un calcul de l'APD sur les entrées *aabbcd* et *abccd*.

Exercice 6. Exercices 5.11, 5.12, 5.13, 5.14 de Amadio.

En TP

Exercice 7. Ajoutez des actions au fichier `parseur.y` de l'exercice 6 (Td-Tp 4) 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.

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 | _)
```