Fiche de TD-TP no. 3

Analyse lexicale : lex

Exercice 1. Expliquez la signification des symboles suivants dans le contexte d'une expression régulière Lex :

`*` `+` `|` `?` `[` `]` `{` `}` `(` `)` `\` `"` `.` `^` `$` `/`

Pour chaque symbole proposez un cas d'utilisation.

Exercice 2.

1. Lequel des ces mots

   `ba`, `b+b+`, `(ab))`,

   est filtré par le motif `[(ab)+]+` ?
2. Soit `ident` defini comme d'habitude :

   ```
   ident [a-zA-Z][a-zA-Z0-9_]+
   ```

   Lequel des ces mots

   `then`, `ma_variable`, `net`, `i|e`, {}

   est filtré par le motif `[{ident}|then]{1,3}` ?

Exercice 3. La compilation du fichier lex suivant est sensée produire un analyseur lexical qui élimine les commentaires d'un fichier contenant du code source en langage C :

```
%option main
%%
"/*"(.|\n)*"*/" {}
.|\n {printf("%s",yytext);}
%%
```

Expliquez pourquoi ce code n'est pas satisfaisant. Proposez une correction.

Exercice 4. Considérez le ficher lex suivant :

```
%option main
%%
ab {printf("court\n");}
(ab)*aba {printf("long\n");}
%%
```

1. Construisez deux AFN $\mathcal{A}_1, \mathcal{A}_2$ pour reconnaître, respectivement, les langages des expression régulières ab et $(ab) * aba$.
2. Construisez un AFD capable de reconnaître l'expression régulière $ab|(ab) * aba$. On étiquettera chaque état final par une priorité chosie entre 1 et 2. et, après re-nommage des états et ajout des priorités sur les états finaux :
3. A l'aide de cet automate et des pointeurs `yytex`, `fc` et `cc`, simulez un calcul d'un lexeur généré par `lex` sur l'entrée

   `ababacaaab`

Exercice 5. Dans cet exercice nous allons revenir sur l'exercice 8 du TP dernier. Cet exercice demandait d'écrire un analyseur lexicale capable d'extraire toutes les références contenues dans des balise de type `<a ...>` où `<img ...>` à partir d'un fichier html. L'exercice demandait aussi de destructurer une références en ses composantes structurelles.
Destructurez cet exercice : par exemple, on pourra écrire le code `lex` pour générer trois lexeurs qui seront utilisé en cascade de la façon suivante :

Écrivez une ligne de commande pour utiliser ces trois lexeurs.
Remarque : on pourra faire référence à la page web http://www.w3.org/Addressing/URL/5_BNF.html

Analyse syntaxique : grammaires et APs

Exercice 6. Exercice 3.5 de Amadio.

Exercice 7.

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 8.

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. Conbstruisez une dérivation et un arbre de dérivation du mot *abbccd*.
3. Montrez que la grammaire proposez est ambiguë.

En TP

Lex

Exercice 9. Modifiez votre solution de l'exercice 8 (TP 2) selon la de-structuration proposée par l'exercice 5.

Prise en main de l'outil yacc (bison)

Exercice 10. 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 11. Ajoutez des actions au fichier `parseur.y` de l'exercice 10 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.