Université Gustave Eiffel

Travaux Dirigés d'analyse syntaxique n°2

Licence d'informatique

———

Analyse lexicale avec `flex`

Ce TD a plusieurs objectifs :

- approfondir l'analyse lexicale en `flex`, y compris les conditions de démarrage et l'opérateur /,
- prendre en main `bison`, dans le cas simple où on l'utilise avec `flex` et où les lexèmes sont des caractères,
- se familiariser avec les grammaires algébriques.

▸ Exercice 1. *Conditions de démarrage et opérateur /*

1. *Écrire un programme* `flex` *qui prend un programme C et affiche tous les noms de fonctions utilisés dans ce programme. Si on prend en entrée le programme suivant :*

```
int plus(int a,int b) {
   return a+b;
}
int main
        (void) {
   printf ("%d\n",plus(4,7));
   getchar();
   return 0;
}
```

   on devra obtenir la liste suivante : `plus main printf plus getchar`

2. *Écrire une nouvelle version qui saute les commentaires : si un nom apparait dans une chaîne entre guillemets ou dans un commentaire, que ce soit avec la syntaxe /* ... */ ou avec la syntaxe //, ce n'est* **pas** *un nom de fonction. Si on prend en entrée le programme suivant :*

```
/* la fonction plus(int,int) renvoie
   la somme de ses paramètres */
int plus(int a,int b) {
   return a+b;
}
int main
       (void) {
   printf ("%d\n",plus(4,7));
   getchar(); // getchar() attend un retour chariot
   return 0;
}
```

on devra obtenir seulement : `plus main printf plus getchar`. *Indication : pour la syntaxe /* ... */, utilisez une condition de démarrage (*start condition*).*

▸ **Exercice 2.**

1. *Faire un programme* `flex` *pour un analyseur lexical dans lequel chaque lettre est un lexème. Pour chaque lettre, l'analyseur doit renvoyer le caractère comme valeur de retour de* `yylex()`. *Il doit renvoyer* 0 *quand il rencontre la fin de la ligne (*`bison` *interprètera ce* 0 *comme lexème de fin de fichier), et ignorer les autres caractères. Attention,* ***on ne peut pas tester*** *cet analyseur lexical avant d'avoir fait la question 2.*
2. *Faire avec* `bison` *un analyseur syntaxique pour la grammaire suivante et l'interfacer avec votre analyseur lexical :*

   $S \rightarrow aS \mid b$

   Compiler et tester l'analyseur syntaxique sur quelques fichiers d'entrée. Vérifier que l'analyseur termine l'exécution tout de suite chaque fois que le fichier d'entrée contient un mot engendré par la grammaire, et qu'il produit un message d'erreur sinon.
3. *Ouvrir le programme en C engendré par* `bison` *et trouver l'appel de* `yylex()` *par* `yyparse()`.

▸ **Exercice 3.** *Pour chacune des grammaires suivantes, caractériser le langage engendré, soit par une expression régulière sur* a *et* b*, soit d'une autre façon mathématique.*

1. $S \rightarrow aS \mid b$
2. $S \rightarrow Sa \mid b$
3. $S \rightarrow aaS \mid b$
4. $S \rightarrow aSa \mid b$
5. $S \rightarrow aSa$
6. $\begin{cases} S \rightarrow aS \mid T \\ T \rightarrow Ta \mid b \end{cases}$
7. $S \rightarrow aS \mid Sa \mid aaS \mid aSa \mid b$