Université Gustave Eiffel

Travaux Dirigés d'analyse syntaxique n°9

Licence d'informatique

———

Interface Flex-Bison et récupération sur erreur

A part un exercice facultatif pour approfondir l'analyse ascendante SLR, le but de ce TD est d'analyser du code source qui ressemble à du C en utilisant `bison` et `flex`. En particulier, on transmet des informations de `flex` à `bison` par l'intermédiaire de la variable `yylval`. Puis on expérimente la récupération sur erreur.

▸ **Exercice 1.** *(Facultatif) On rappelle qu'une grammaire SLR ne peut pas être ambiguë. La grammaire ci-dessous n'est pas ambiguë, est-elle SLR ?*

$$\begin{cases} S & \to & G = D \\ S & \to & D \\ G & \to & *D \\ G & \to & \texttt{id} \\ D & \to & G \end{cases}$$

▸ **Exercice 2.** ***Coupler Flex et Bison***

1. *Faites une nouvelle version du programme* `flex` *de votre projet pour produire un analyseur lexical qu'on puisse coupler avec* `exp-tpc.y`*. L'analyseur lexical doit aussi accepter les commentaires entre* `/*` *et* `*/` *et après* `//`*.*

   Pour vos tests, le fichier `exp.tpc` *contient une expression conforme à la grammaire de* `exp-tpc.y`*.*

2. *Modifiez* `exp-tpc.y` *et votre programme* `flex` *pour que l'analyseur syntaxique affiche des traces indiquant les opérateurs* `+` *et* `-` *(binaires comme dans* `a+b` *ou unaires comme dans* `a=-b`*),* `*`*,* `/`*,* `\%`*,* `==`*,* `!=`*,* `<`*,* `<=`*,* `>`*,* `>=` *et les appels de fonctions, avec le nom de la fonction. Par exemple, quand on analyse l'expression* `distance(value((-b)/(2*a)))`*, on veut voir apparaitre :*

   ```
   unary -
   *
   /
   fn value
   fn distance
   ```

   Pour transmettre les informations de l'analyseur lexical à l'analyseur syntaxique, utilisez la variable `yylval` *définie automatiquement par* `bison`*.*

▸ **Exercice 3.** ***Récupération sur erreur***

1. *Complétez le programme* `bison` *de l'exercice 2 pour qu'en cas d'erreur de syntaxe dans une expression parenthésée, il continue à analyser la donnée après l'erreur. Pour vos tests, adaptez* `exp.tpc` *en introduisant une erreur.*
2. *Même exercice pour les erreurs de syntaxe qui se produisent :*
   — *dans un appel de fonction,*
   — *dans un appel de fonction, mais en supposant que l'erreur n'est pas dans le dernier paramètre mais dans l'un des précédents, dans le but de redémarrer à temps pour pouvoir analyser le dernier paramètre.*