Laboratoire "little langage" et "interpreter" - séance 4

Date de rentrée : 19 décembre 2014

1. Ecrire en Java un petit interpréteur pour des expressions booléennes en utilisant les opérateurs and et or. Exemples d'entrées pour l'interpréteur :

   ```
   (true and X) or (Y and Z)
   X and Y or Z
   ```

   Si X $\rightarrow$ true, Y$\rightarrow$false et Z$\rightarrow$true pour les deux exemples, alors dans les deux cas les expressions sont réduites à true.
   Comme point de départ, servez vous de la grammaire ci-dessous.

   ```
   <expression>::=<orexpr>

   <orexpr>::=<andexpr> 'or' <orexpr>
   <orexpr>::=<andexpr>

   <andexpr>::=<simpleexpr> 'and' <andexpr>
   <andexpr>::=<simpleexpr>

   <simpleexpr>::= 'A' | 'B' | ... | 'Z' | 'true' | 'false' | '('<expression> ')'
   ```

   Les principales étapes à réaliser sont les suivantes :

   (a) Ecrire une classe `LexicalAnalyser`.
   (b) Ecrire une classe `Parser`.
   (c) Ecrire les classes pour l'interpréteur(classes représentant des non terminaux et des terminaux).
   (d) Ces classes ont une méthode `interpret(Context c)` ou `execute(Context c)`.
   (e) Création de la source de données(par exemple un fichier contenant les expressions à interpréter).
   (f) Dans le `main`, vous créez un contexte(association variables - valeurs), vous instancier le parseur en lui donnant la source de données et vous récupérer l'objet non terminal qui est le point de départ pour lancer l'interprétation.

   La classe `LexicalAnalyser` :

   (a) "découpe" l'entrée (en utilisant par exemple un `StreamTokenizer`).
   (b) implémente la méthode Token nextToken() pour progresser dans l'entrée et donner le symbole sous le "curseur" pour la suite.

   La classe `Parser` :

   (a) La classe a une méthode `parse`.
   (b) Il y a une procédure pour chaque non terminal.
   (c) Le parsing commence par exemple par l'appel à parse qui appelle la procédure correspondant à l'axiome.
   (d) La procédure détermine la production à utiliser et développe la partie droite(cfr slides).
   (e) Un terminal correspondant avec l'unité lexicale en entrée provoque la lecture de la prochaine unité lexicale sur l'entrée. Ecrivez une méthode ayant le prototype `boolean match(Token expectedToken)`.
   (f) Chaque non terminal de la production résulte en l'appel de la procédure correspondante.
   (g) Chaque procédure peut éventuellement renvoyer un objet si vous voulez construire un arbre pour la construction. Autrement dit, pendant le parsing, on peut construire un AST.

2. Ecrire une grammaire et un interpréteur pour des expressions mathématiques(par exemple : $(x + 10) \times sin(x) - 3 \times cos(x \times x)$). Implémentez

- l'interprétation de l'expression. Les valeurs des différentes variables sont fournies par l'utilisateur.
- la dérivation de l'expression. Pour rappel
  - La dérivée d'un entier est 0.
  - La dérivée de x est 1.
  - La dérivée d'une somme est la somme des dérivées.
  - La dérivée de $x \times y = x' \times y + x \times y'$.
  - La dérivée de $f(x) = x' \times f'(x)$.