CS 5318 (Spring 2019)

Principles of Programming Languages

Programming assignment 3

Due on May 9 (Thursday) at 7:30pm

Consider the following grammar for applied Lambda expressions:

```
<expr> ::= <num> | "+" | "-" | "*" | "/"                // constants
         | <var>                                        // identifier
         | "(" <expr> <expr>* ")"                       // application
         | "(" "L" <var>+ "." <expr>+ ")"               // abstraction
```

Main part: Do the following

1. Write a JJTree specification `Prog3.jj` to generate the AST for any applied Lambda expression.
2. Implement the method `String astToString()` to convert the AST of a Lambda expression into its concrete syntax form.
3. Implement the method `Set<String> freeVars()` to find the set of free variables of a Lambda expression AST.
4. Implement the method `void dumpFV()` to dump a Lambda expression AST with free variables.

Bonus part: Write a separate program `Bonus.java` for this part and implement

1. the method `SimpleNode substitute(String var, SimpleNode expr)` for substituting all the free occurrences of the variable `var` with a copy of the Lambda expression AST `expr` in a Lambda expresson AST
2. the method `SimpleNode normalOrderEvaluate()` for performing a normal order evaluation of a Lambda expression

Create a tar file named Prog3.tar from your programs (including the bonus part) for this assignment and submit it by the due date through TRACS.

Here is a sample execution for the main part:

```
[hs@zeus Prog3]$ java Prog3
>>> Lambda Expression Evaluator <<<
Enter an applied Lambda expression:
((L f x1 . f (f x1)) (L n . * 2 (- n 1)) 3)

The abstract syntax tree:
appl
 appl
  lamb
   f
   lamb
    x1
    appl
     f
     appl
```

```
      f
      x1
  lamb
   n
   appl
    appl
     mul
     2
    appl
     appl
      sub
      n
     1
 3

The Lambda expression in the concrete syntax:
(((L f . (L x1 . (f (f x1)))) (L n . ((* 2) ((- n) 1)))) 3)

The abstract syntax tree with free variables:
appl   []
 appl   []
  lamb   []
   f   [f]
   lamb   [f]
    x1   [x1]
    appl   [f, x1]
     f   [f]
     appl   [f, x1]
      f   [f]
      x1   [x1]
  lamb   []
   n   [n]
   appl   [n]
    appl   []
     mul   []
     2   []
    appl   [n]
     appl   [n]
      sub   []
      n   [n]
     1   []
 3   []
```

Here is a sample execution for the bonus part:

```
[hs@zeus Prog3]$ java Bonus
>>> Lambda Expression Evaluator <<<
Enter an applied Lambda expression:
(L x1 . f (f x1))

The abstract syntax tree:
lamb
 x1
 appl
  f
  appl
   f
   x1

Enter the variable to be substituted:
f

Enter the substituting applied Lambda expression:
(L n . * 2 (- n 1))
```

```
The abstract syntax tree:
lamb
 n
 appl
  appl
   mul
   2
  appl
   appl
    sub
    n
   1

The substitution result:
lamb
 x1
 appl
  lamb
   n
   appl
    appl
     mul
     2
    appl
     appl
      sub
      n
     1
  appl
   lamb
    n
    appl
     appl
      mul
      2
     appl
      appl
       sub
       n
      1
   x1

Enter an applied Lambda expression:
((L f x1 . f (f x1)) (L n . * 2 (- n 1) ) 3)

The normal order evaluation result:
6
```