Coursework 3

Sunday, November 17, 2019 12:28 PM

Question 1:

Design a grammar for the WHILE language and give the grammar rules. The main categories of non-terminals should be:

- arithmetic expressions (with the operations from the previous coursework, that is +, -, *, / and %)
- boolean expressions (with the operations ==, <, >, >=, <=, !=, &&, ||, true and false)
- single statements (that is skip, assignments, ifs, while-loops, read and write)
- compound statements separated by semicolons
- blocks which are enclosed in curly parentheses

Make sure the grammar is not left-recursive.

A1:

```
AExp  ::= T · + · AExp | T · - · AExp | T
T     ::= F · * · T | F · / · T | F · % · T
F     ::= ( · AExp · ) | id | num

BExp  ::= AExp · == · AExp | AExp · != · AExp | AExp · < · AExp | AExp · > · AExp
      ::= AExp · <= · AExp | AExp · >= · AExp
      ::= ( · BExp · ) | ( · BExp · ) · && · BExp | ( · BExp · ) · || · BExp
      ::= true | false

Stmt  ::= skip | id · := · AExp | write · AExp | write · str | read · id
      ::= if · BExp · then · Block · else · Block
      ::= while · BExp · do · Block

Stmts ::= Stmt · ; · Stmts | Stmt

Block ::= { · Stmts · } | Stmt
```

Q2:

You should implement a parser for the WHILE language using parser combinators. Be careful that the parser takes as input a stream, or list, of tokens generated by the tokenizer from the previous coursework. For this you might want to filter out whitespaces and comments. Your parser should be able to handle the WHILE programs in Figures 2, 3 and 4 (if your lexer cannot deal with comments you can delete them from the prime number program). In addition give the parse tree for the statement:

if (a < b) then skip else a := a * b + 1

A2:

```
                    Block
                      |
                      if
                      |
            |----------|----------|
            |   (then)|          |(else)
          BExp     Block      Block
            |          |          |
          Bop       Skip       Assign
            |                       |
    |-------|------|            |---------|
   Var     <     Var         a        AExp
    |              |                     |
    a              b                    Aop
                                         |
                             |---------|---------|
                            Aop       +       Num
                             |                   |
                      |------|------|            1
                     Var    *     Var
                      |             |
                      a             b
```

Q3:
Implement an interpreter for the WHILE language you designed and parsed in Question 1 and 2. This interpreter should take as input a parse tree. However be careful because, programs contain variables and variable assignments. This means you need to maintain a kind of memory, or environment, where you can look up a value of a variable and also store a new value if it is assigned. Therefore an evaluation function (interpreter) needs to look roughly as follows

```
eval_stmt(stmt , env)
```

where stmt corresponds to the parse tree of the program and env is an environment acting as a store for variable values. Consider the Fibonacci program in Figure 2. At the beginning of the program this store will be empty, but needs to be extended in line 3 and 4 where the variables minus1 and minus2 are assigned values. These values need to be reassigned in lines 7 and 8. The program should be interpreted according to straightforward rules: for example an if-statement will “run” the if-branch if the boolean evaluates to true, otherwise the else-branch. Loops should be run as long as the boolean is true.

Give some time measurements for your interpreter and the loop program in Figure 3. For example how long does your interpreter take when start is initialised with 100, 500 and so on. How far can you scale this value if you are willing to wait, say 1 Minute?

A3:

Loop init value	Time (ns)	Time (ms)	Time (s)
100	108113600	108	0
200	882586900	882	0
500	12142977600	12142	12
800	50785834800	50785	50
1000	97913050400	97913	97

Note: I am running Scala on the WSL (Windows Subsystem for Linux) which is in itself an interpreter for Linux executables (more or less). As such, although I have a good relatively modern laptop, it performs pretty bad, some might argue even worse then how scala usually does.