Homework Assignment #2

Due: Wednesday, October 19

1. Write a grammar for English sentences using the words

   `time`, `arrow`, `banana`, `flies`, `like`, `a`, `an`, `the`, and `fruit`

   and the semicolon. Be sure to include all the senses (noun, verb, etc.) of each word. Then show that this grammar is ambiguous by exhibiting more than one parse tree for "time flies like an arrow; fruit flies like a banana." (Note: this is one reason that computerized natural language understanding is hard!)

2. Write an unambiguous grammar for each of the following languages:

   a. Balanced parentheses and brackets, e.g. `([[](()[()][])])`.

   b. Balanced parentheses and brackets, where a closing bracket also closes all outstanding open parentheses (up to the previous open bracket), e.g. `[([](((()[(][])]`.

   c. Statement blocks as in ML where semicolons *separate* statements, e.g. `( beep ; ( beep ; beep ) ; beep )`. (You may assume that `beep` is the only kind of statement other than a statement block.)

   d. Statement blocks as in C where semicolons *terminate atomic* statements, e.g. `{ beep ; { beep ; beep ; } beep ; }`.

3. Rewrite the following grammar (terminals in boldface font) so that it becomes LL(1), i.e., eliminate any ambiguities, left-recursion, and left-factors.

   ```
   S ::= S ; S | id := E
   E ::= id | num | E + E | ( S , E ) | id ( L )
   L ::= E | L , E
   ```

   (In the above grammar, **+** and **;** are left-associative.)

4. Consider the following grammar (terminals in boldface font):

   ```
   S ::= N V
   N ::= time flies | bananas
   V ::= like arrows | ripen
   ```

   Using the LR(0) construction algorithm described in class:

   a. Construct the states (each defined by a set of items) and show each state's shift transitions and/or reduce and/or accept actions.

   b. Give the LR(0) action and goto tables.

   c. Illustrate the execution of this machine to parse the input string "`time flies ripen`".

5. Consider the following MiniJava statement:

```
while (x < 3+4*5) {
   System.out.println(x);
   y = new C().test(a,b,c);
}
```

   a. Draw the concrete syntax tree for this statement. Use the MiniJava grammar embedded in the `Parser/minijava.cup` file. (You may abbreviate non-terminal names if unambiguous, e.g. `Id` for `Identifier`.)

   b. Draw the abstract syntax tree for this statement, labelling each AST node with the name of an AST node class from the MiniJava compiler and labelling each child edge with the name of an instance variable of the parent node's class. (Child edges that lead to non-AST nodes, such as integers or strings, can be drawn to lead to a particular integer or string value.)

Produce a hard-copy of your answers and turn them in to the TA by the start of class on the due date.

Do these exercises individually, not with your project partner.