Quiz solutions

Compiler Construction, Fall 2012

Wednesday, December 19, 2012

Last Name : ______________________________

First Name : ______________________________

Exercise	**Points**	**Achieved Points**
1	10	
2	10	
3	20	
Total	40	

Problem 1: Lexical Analysis (10 points)

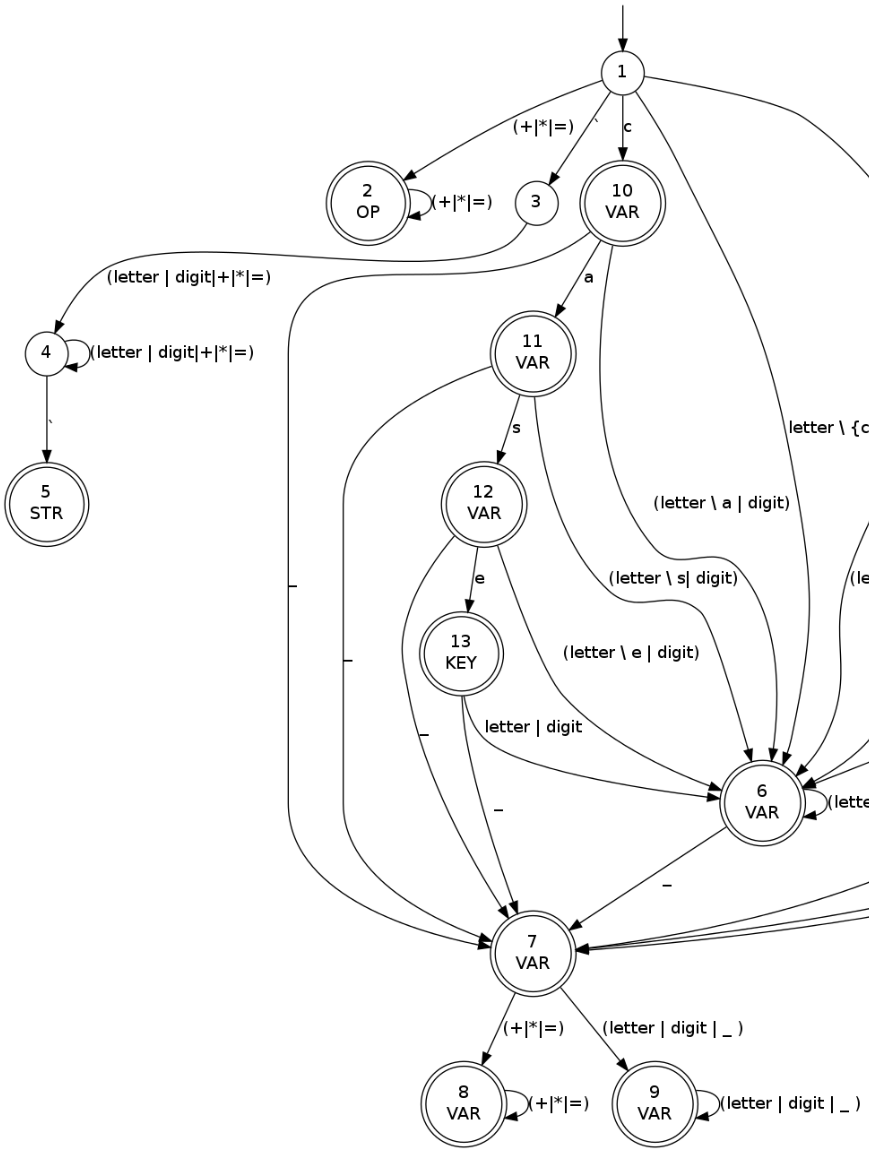
a) [2 pts]

```
big_bob | ++= | ‘def‘
VARID      OP    STRINGID

+* | ‘case‘ | type_x | == | func123_def | =+= | case | ** | def_77
OP   STRING    VARID   OP    VARID         OP    KEY    OP    VARID
```

b) [8 pts]

In the automaton, VAR stands for VARID, and STR for STRINGID. All missing links go to the error state.



Problem 2: Grammars (10 points)

a) [8 pts]

E_1 -> E_2 : E_1 | E_2
E_2 -> E_3 < E_3 | E_3
E_3 -> E_3 + E_4 | E_4
E_4 -> E_4 * E_5 | E_5
E_5 -> (E_1) | num

b) [2 pts]

```
2:  ( (3 < (4+5)) :  ((6*7)*8) )


         E1
        / | \
      E2  :   E1
     /      / | \
    E3    E2  :  E1
   /     / | \     \
  E4   E3  < E3     E2
  |    |     /|\      \
  E5   E4  E3 + E4     E3
  |    |    |   |       \
  2    E5   E4  E5       E4
       |    |   |       / | \
       3    E5  5     E4  *  E5
            |        / | \   |
            4       E4 * E5  8
                    |    |
                    E5   7
                    |
                    6
```

Problem 3: Parsing (20 points)

a) **[5 pts]**

The CYK algorithm runs as follows:

```
 _________________E_____________
|                              |
     _________E___________
    |                     |
     ______E_____
    |            |
 E    E    E         E
 2    3    4    +    5    -    *
```

Thus we get the following triples:
(E, 0, 1), (E, 1, 2), (E, 2, 3), (E, 4, 5), (E, 1, 4), (E, 1, 6), (E, 0, 7)

The only parse that we get is the following:

```
      E
    / | \
  E   E  *
 /  / | \
2  E  E  -
 / |\  \
E  E +  5
|  |
3  4
```

b) **[15 pts]**

We claim that the grammar is not ambiguous. Proof: consider parsing the language in reverse. For this we reverse all the grammar rules such that `E -> EE+` becomes `E -> +EE`. The resulting grammar is LL(1) as each grammar rule starts with a distinct nonterminal. Thus, we can parse the reverse string unambiguously with an LL(1) parser, and reverse the resulting parse tree to obtain the parse tree of the original grammar.