CMPSC 160 – Practice Problems 1

1. Write regular expressions for the following languages over \( \Sigma = \{0, 1\} \):
   
   a) All strings containing exactly one 0
   
   b) All strings where each 0 is followed by at least one 1
   
   c) All strings containing no more than three 0s
   
   d) All strings not ending in 01
   
   e) All strings with an odd number of 0s
   
   f) All strings with an odd number of 0s and an odd number of 1s
   
   g) All strings that contain 101
   
   h) All strings that do not contain 101

2. Write a regular expression for numerical constants which conform to the following list of restrictions:

   - At least one digit precedes the decimal point
   - The decimal point is optional, but if present, it is followed by at least one digit.
   - Except as demanded by (a) and (b), leading and trailing zero digits are not permitted.

3. Construct a nondeterministic finite automata for the regular expression \( b(a|b)^* \) using Thompson’s construction, show the steps of the algorithm.

4. Show that the grammar

   \[
   \begin{align*}
   S & \rightarrow \text{Adjective Noun} \\
   \text{Adjective} & \rightarrow \text{good} \mid \text{bad} \mid \text{Noun} \\
   \text{Noun} & \rightarrow \text{weather} \mid \text{report} \mid \text{Adjective Noun}
   \end{align*}
   \]

   is ambiguous by finding two different parse trees for the phrase “bad weather report”. Give the leftmost and rightmost derivations corresponding to one of the parse trees.

5. Consider the following grammar:

   \[
   \begin{align*}
   S & \rightarrow \text{while } E \text{ do } S \\
   & \mid \text{id = id} \\
   & \mid \text{id ( } E \text{ )} \\
   E & \rightarrow E \text{ and bool} \\
   & \mid \text{bool}
   \end{align*}
   \]

   First transform the above grammar to an LL(1) grammar using left-factoring and immediate left-recursion elimination algorithms and then write a recursive descent parser for the resulting grammar. Use the token definitions and the functions given below:
```c
int WHILE=..., DO=..., ID=..., EQ=..., LPAREN=..., RPAREN=..., AND=..., BOOL=...;

void match(int token) {
    if (lookahead == token)
        lookahead = getNextToken();
    else
        error();
}

void main() {
    lookahead = getNextToken();
    S();
    match(EOF);
}
```

6. Consider the following grammar:

```
S → N R
R → dot dot dot
    | dot dot dot N
N → N digit
    | digit
```

(a) Is the above grammar LL(k) for some k? Explain. Eliminate the left recursion from the above grammar. Is the resulting grammar LL(k) for some k? Explain.

(b) Convert the grammar you constructed in part (a) to an LL(1) grammar by left factoring.

7. Consider the following LL parse table:

```
  | id      | −      | /      | $      |
---|---------|--------|--------|--------|
E | E → T R | −      | /      | $      |
R | R → − T R | R → ε  |
T | T → id P | P → ε  | P → / id P | P → ε |
```

Show the contents of the stack and the input buffer during the parse of the input string:

```
  id − id / id
``` 

8. Consider the following grammar:

```
S → ( L )
  | a
L → S R
R → , S R
    | ε
```

(a) Compute the FIRST and FOLLOW sets for the nonterminals S, L and R.

(b) Construct the LL(1) parse table for the above grammar.