CMSC 430, Practice Problems 1

For these practice problems, when creating NFAs from REs, use Thompson’s construction algorithm, but with a modified algorithm for computing closure. Instead of adding an epsilon edge from the final state back to the start state of the original NFA, add the epsilon edge from the final state back to the start state of the new NFA instead. The resulting NFA should appear as follows:

1. Describe the languages denoted by the following regular expressions:
   a. $0(01)^*0$
   b. $((\varepsilon | 0)1)^*$
   c. $(01)^*0(01)(011)$

2. Regular expressions
   a. Give a regular expression for all binary numbers including the substring “101”.
   b. Give a regular expression for all binary numbers with an even number of 1’s.
   c. Give a regular expression for all binary numbers that don’t include “000”.

3. Finite automata
   a. Give a NFA that only accepts binary numbers including the substring “101”.
   b. Give a NFA that only accepts binary numbers that include either “00” or “11”.
   c. Give a NFA that only accepts binary numbers that include both “00” and “11”.
   d. Compute the $\varepsilon$-closure of the start state for each of the NFA above.
   e. Give a DFA that only accepts binary number with an odd number of 1’s.
   f. Give a DFA that only accepts binary numbers that include “00”.
   g. Give a DFA that only accepts binary numbers that don’t include “000”.

4. Reducing REs to DFAs
   For each regular expression: $1^*, (0|01)^*0$
   a. Reduce the RE to an NFA using the modified Thompson’s algorithm above.
   b. Reduce the resulting NFA to an DFA using the subset algorithm.
   c. Show whether the DFA accepts / rejects the strings “1”, “11”, “101”
   d. Minimize the resulting DFA using Hopcroft reduction
5. Describing grammars
   a. Describe the language accepted by the following grammar:
      \[ S \rightarrow abS \mid a \]
   b. Describe the language accepted by the following grammar:
      \[ S \rightarrow aSb \mid \epsilon \]
   c. Describe the language accepted by the following grammar:
      \[ S \rightarrow bSb \mid A \quad A \rightarrow aA \mid \epsilon \]
   d. Describe the language accepted by the following grammar:
      \[ S \rightarrow AS \mid B \quad A \rightarrow aAc \mid Aa \mid \epsilon \quad B \rightarrow bBb \mid \epsilon \]
   e. Describe the language accepted by the following grammar:
      \[ S \rightarrow S \text{ and } S \mid S \text{ or } S \mid (S) \mid \text{true} \mid \text{false} \]
   f. Which of the previous grammars are left recursive?
   g. Which of the previous grammars are right recursive?
   h. Which of the previous grammars are ambiguous? Provide proof.

6. Creating grammars
   a. Write a grammar for a<b,y>, where x = y
   b. Write a grammar for a<b,y>, where x > y
   c. Write a grammar for a<b,y>, where x = 2y
   d. Write a grammar for all strings of a and b that are palindromes.
   e. Write a grammar for all strings of a and b that include the substring baa.
   f. Write a grammar for all strings of a and b with an odd number of a’s and b’s.

7. Derivations and parse trees
   For the following grammar: \[ S \rightarrow S \text{ and } S \mid \text{true} \]
   a. List 4 possible derivations for the string “true and true and true”.
   b. Label each derivation as left-most, right-most, or neither.
   c. List the parse tree for each derivation

8. Consider the following grammar \[ S \rightarrow aSbS \mid bSaS \mid \epsilon \]
   a. Prove grammar is ambiguous by finding 2 left-most derivations for ”abab”
   b. Prove grammar is ambiguous by finding 2 right-most derivations for ”abab”

9. Consider the following grammar: \[ S \rightarrow S \mid S \mid S \mid (S) \mid \text{true} \mid \text{false} \]
   a. Compute First sets for each production and nonterminal
   b. Explain why the grammar cannot be parsed by a LL(1) parser

10. Consider the following grammar: \[ S \rightarrow abS \mid acS \mid c \]
    a. Compute First sets for each production and nonterminal
    b. Show why the grammar cannot be parsed by a LL(1) parser.
    c. Rewrite the grammar so it can be parsed by a LL(1) parser.
    d. Write a recursive descent parser for the rewritten grammar.
11. Consider the following grammar: \[ S \rightarrow Sa \mid Sc \mid c \]
a. Show why the grammar cannot be parsed by a LL(1) parser.
b. Rewrite the grammar so it can be parsed by a LL(1) parser.
c. Using FIRST and FOLLOW, build the LL(1) table for the new grammar
d. Using the LL(1) table, show how a table-driven parser parses the string “caca”
e. Write a recursive descent parser for your new grammar

12. Consider the following grammar

\[
\begin{align*}
\text{Goal} & \rightarrow \text{Expr} \\
\text{Expr} & \rightarrow \text{Term} + \text{Expr} \\
& \mid \text{Term} - \text{Expr} \\
& \mid \text{Term} \\
\text{Term} & \rightarrow \text{Factor} \times \text{Term} \\
& \mid \text{Factor} / \text{Term} \\
& \mid \text{Factor} \\
\text{Factor} & \rightarrow \text{num} \\
& \mid \text{id}
\end{align*}
\]
a. Apply left factoring to create a LL(1) grammar
b. Compute FIRST and FOLLOW for the modified grammar
c. Using FIRST and FOLLOW, build the LL(1) table for the new grammar