CMSC 430, Practice Problems 1 (Solutions)

1. Describe the languages denoted by the following regular expressions:
   a. $0(0|1)^*0$
      strings beginning and ending in 0
   b. $((\varepsilon | 0)1^*)^*$
      all strings of 0 and 1
   c. $(0|1)^*0(0|1)(0|1)$
      strings with 0 as third digit from right

2. Regular expressions and languages
   a. Give a regular expression for all binary numbers including the substring “101”.
      $(0|1)*101(0|1)*$
   b. Give a regular expression for all binary numbers with an even number of 1’s.
      $0^*(10*10*)^*$
   c. Give a regular expression for all binary numbers that don’t include “000”.
      $(01 | 001 | 1)^*(0 | 00 | \varepsilon)$

3. Finite automata
   a. Give a NFA that only accepts binary numbers including the substring “101”.
      \[\text{Diagram of NFA accepting 101}\]
   b. Give a NFA that only accepts binary numbers that include either “00” or “11”.
      \[\text{Diagram of NFA accepting 00 or 11}\]
c. Give a NFA that only accepts binary numbers that include both “00” and “11”.

\[
\begin{array}{ccc}
0,1 & 0 & 0 \\
2 & 3 & 4 \\
5 & 6 & 7 \\
8 & 9 & 10 \\
11 & 12 & \\
\end{array}
\]

\[\epsilon - \text{closure}(1) = \{1,2,5\}\]

\[\epsilon - \text{closure}(1) = \{1,2,8\}\]

d. Compute the \(\epsilon\)-closure of the start state for each of the NFA above.

- For NFA in (a) \(\epsilon\)-closure(1) = \{1,2\}
- For NFA in (b) \(\epsilon\)-closure(1) = \{1,2,5\}
- For NFA in (c) \(\epsilon\)-closure(1) = \{1,2,8\}

e. Give a DFA that only accepts binary numbers with an odd number of 1’s.

\[
\begin{array}{ccc}
0 & 0 \\
1 & 1 \\
\end{array}
\]

f. Give a DFA that only accepts binary numbers that include “000”.

\[
\begin{array}{ccc}
1 & 0 & 0 \\
2 & 3 & 4 \\
\end{array}
\]

g. Give a DFA that only accepts binary numbers that don’t include “000”.

\[
\begin{array}{ccc}
1 & 0 & 0 \\
0 & 0 & 1 \\
\end{array}
\]
4. Reducing REs to DFAs
   For each regular expression: 1*, (0|01)*0
   a. Reduce the RE to an NFA using the algorithm described in class.
   b. Reduce the resulting NFA to a 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

\[ 1^* \rightarrow \text{NFA} \rightarrow \text{DFA} \]

Accept / reject
- “1” \( \{3,1,4\} \rightarrow \{2,4,3,1\} \) accept
- “11” \( \{3,1,4\} \rightarrow \{2,4,3,1\} \rightarrow \{2,4,3,1\} \) accept
- “101” \( \{3,1,4\} \rightarrow \{2,4,3,1\} \rightarrow \text{reject} \)

Minimized DFA
Initial partitions: accept = \{ \{3,1,4\}, \{2,4,3,1\} \} = P1,
nonfinal = Ø
- \text{move}(\{3,1,4\}, 1) \rightarrow P1
- \text{move}(\{2,4,3,1\}, 1) \rightarrow P1
No need to split P1, minimization done. After cleanup, minimal DFA is
(0101)*0 → NFA

Accept / reject
- “1” \{9,7,1,3,10,11\} → reject
- “11” \{9,7,1,3,10,11\} → reject
- “101” \{9,7,1,3,10,11\} → reject

Minimized DFA
Initial partitions: accept =\{ {2,4…}\} = P1,
nonfinal =\{ {9,7…}, {6,8…}\} = P2
- move({9,7…}, 0) → P1
- move({6,8…}, 0) → P1
- move({9,7…}, 1) → reject
- move({6,8…}, 1) → reject
No need to split P2, minimization done. After cleanup, minimal DFA (different from previous minimal DFA) is
5. Describing grammars
a. Describe the language accepted by the following grammar:
   \[ S \rightarrow abS \mid a \]
   \[(ab)^*a \]
   Description:
   The grammar generates strings of the form \(ab\) with at least one \(a\) at the end and possibly some \(ab\) sequences in between.

b. Describe the language accepted by the following grammar:
   \[ S \rightarrow aSb \mid \varepsilon \]
   \[a^n b^n, n \geq 0 \]
   Description:
   The grammar generates strings consisting of an equal number of \(a\)s and \(b\)s, where \(n\) is non-negative.

c. Describe the language accepted by the following grammar:
   \[ S \rightarrow bSb \mid A \]
   \[ A \rightarrow aA \mid \varepsilon \]
   \[b_n a_n b_n, n \geq 0 \]
   Description:
   The grammar generates strings of \(ab\) with the same number of \(a\)s and \(b\)s, where \(n\) is non-negative.

d. Describe the language accepted by the following grammar:
   \[ S \rightarrow AS \mid B \]
   \[ A \rightarrow aAc \mid Aa \mid \varepsilon \]
   \[B \rightarrow bBb \mid \varepsilon \]
   Strings of \(a\) & \(c\) with same or fewer \(c\)'s than \(a\)'s and no prefix has more \(c\)'s than \(a\)'s, followed by an even number of \(b\)'s
   Description:
   The grammar generates strings where the number of \(a\)s is greater than or equal to the number of \(b\)s, with an even number of \(b\)s at the end.

e. Describe the language accepted by the following grammar:
   \[ S \rightarrow S \text{ and } S \mid S \text{ or } S \mid (S) \mid \text{true} \mid \text{false} \]
   Boolean expressions of \text{true} & \text{false} separated by \& \text{&} or, with some expressions enclosed in parentheses
   Description:
   The grammar generates boolean expressions, possibly with nested parentheses.

f. Which of the previous grammars are left recursive?
   5d, 5e

g. Which of the previous grammars are right recursive?
   5a, 5c, 5d, 5e

h. Which of the previous grammars are ambiguous?  Provide proof.
   Examples of multiple left-most derivations for the same string
   5d:  \[ S \Rightarrow AS \Rightarrow AaS \Rightarrow aS \Rightarrow aB \Rightarrow a \]
   \[ S \Rightarrow AS \Rightarrow S \Rightarrow AS \Rightarrow AaS \Rightarrow aS \Rightarrow aB \Rightarrow a \]
   5e:  \[ S \Rightarrow S \text{ and } S \Rightarrow S \text{ and } S \Rightarrow S \Rightarrow \text{true} \text{ and } S \Rightarrow S \Rightarrow \text{true} \text{ and } S \Rightarrow S \Rightarrow \text{true} \text{ and } S \Rightarrow S \Rightarrow \text{true} \text{ and } \]
   \[ S \Rightarrow S \text{ and } S \Rightarrow \text{true} \text{ and } S \Rightarrow \text{true} \text{ and } S \Rightarrow \text{true} \text{ and } \]
   \[ S \Rightarrow S \Rightarrow \text{true} \text{ and } S \Rightarrow \text{true} \text{ and } S \Rightarrow \text{true} \text{ and } \]

6. Creating grammars
a. Write a grammar for \(a^x b^y\), where \(x = y\)
   \[ S \rightarrow aSb \mid \varepsilon \]

b. Write a grammar for \(a^x b^y\), where \(x > y\)
   \[ S \rightarrow aL \]
   \[ L \rightarrow aL \mid aLb \mid \varepsilon \]

c. Write a grammar for \(a^x b^y\), where \(x = 2y\)
   \[ S \rightarrow aaSb \mid \varepsilon \]

d. Write a grammar for all strings of \(a\) and \(b\) that are palindromes.
   \[ S \rightarrow aSa \mid bSb \mid L \]
   \[ L \rightarrow a \mid b \mid \varepsilon \]

e. Write a grammar for all strings of \(a\) and \(b\) that include the substring \(baa\).
   \[ S \rightarrow LbaaL \]
   \[ L \rightarrow aL \mid bL \mid \varepsilon \]
   // \(L = \text{any} \)

f. Write a grammar for all strings of \(a\) and \(b\) with an odd number of \(a\)'s and \(b\)'s.
   \[ S \rightarrow EaEbE \mid EbEaE \]
   \[ E \rightarrow EaEaE \mid EbEbE \mid \varepsilon \mid SS \]
   // \(E = \text{even} \ #s \)
7. Derivations and parse trees

For the following grammar:  $S \rightarrow S \text{ and } S \mid \text{true}$

a. List 4 derivations for the string “true and true and true”.
   i. $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow true$ and $S$ and $S \Rightarrow true$ and $true$ and $true$ and $true$
   ii. $S \Rightarrow S$ and $S \Rightarrow true$ and $S \Rightarrow true$ and $S$ and $S \Rightarrow true$ and $true$ and $true$ and $true$
   iii. $S \Rightarrow S$ and $S \Rightarrow true$ and $S \Rightarrow S$ and $true \Rightarrow true$ and $true$ and $true$ and $true$
   iv. $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and $true \Rightarrow S$ and $true$ and $true$ and $true$
   v. $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow true$ and $S$ and $S \Rightarrow true$ and $true$ and $true$ and $true$
   vi. $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and $true \Rightarrow true$ and $true$ and $true$ and $true$
   vii. $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow true$ and $true$ and $true$ and $true$
   viii. $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and $true \Rightarrow true$ and $true$ and $true$ and $true$
   ix. $S \Rightarrow S$ and $S \Rightarrow S$ and $true$ and $true$ and $true$ and $true$
   x. $S \Rightarrow S$ and $S \Rightarrow S$ and $true$ and $true$ and $true$ and $true$
   xi. $S \Rightarrow S$ and $S \Rightarrow S$ and $true$ and $true$ and $true$ and $true$
   xii. $S \Rightarrow S$ and $S \Rightarrow S$ and $true$ and $true$ and $true$ and $true$
   xiii. $S \Rightarrow S$ and $S \Rightarrow S$ and $true$ and $true$ and $true$ and $true$
   xiv. $S \Rightarrow S$ and $S \Rightarrow S$ and $true$ and $true$ and $true$ and $true$
   xv. $S \Rightarrow S$ and $S \Rightarrow S$ and $true$ and $true$ and $true$ and $true$
   xvi. $S \Rightarrow S$ and $S \Rightarrow S$ and $true$ and $true$ and $true$ and $true$

b. Label each derivation as left-most, right-most, or neither.
   i and ii are left-most derivations, iii and iv are right-most derivations, remaining derivations are neither

c. List the parse tree for each derivation
   Tree 1 = ii, iii, x, xi, Tree 2 = rest
8. Consider the following grammar: $S \rightarrow aSb \mid bSa \mid \epsilon$
   a. Prove grammar is ambiguous by finding 2 left-most derivations for "abab"
      
      $S \Rightarrow aSbS \Rightarrow abS \Rightarrow abaSbS \Rightarrow ababS \Rightarrow abab$
      $S \Rightarrow aSbS \Rightarrow abSaSbS \Rightarrow abaSbS \Rightarrow ababS \Rightarrow abab$
   b. Prove grammar is ambiguous by finding 2 right-most derivations for "abab"
      
      $S \Rightarrow aSbS \Rightarrow aSaSbS \Rightarrow aSaSb \Rightarrow aSbab \Rightarrow abab$
      $S \Rightarrow aSbS \Rightarrow aSb \Rightarrow abSaSb \Rightarrow abSaSb \Rightarrow abab$

9. Consider the following grammar: $S \rightarrow S \mid S \mid S \mid (S) \mid true \mid false$
   a. Compute First sets for each production and nonterminal
      
      $FIRST(true) = \{ "true" \}$
      $FIRST(false) = \{ "false" \}$
      $FIRST((S)) = \{ "(" \}$
      $FIRST(S and S) = FIRST(S or S) =$
            $FIRST(S) = \{ "(", "true", "false" \}$
   b. Explain why the grammar cannot be parsed by a LL(1) parser
      
      First sets of productions intersect, grammar is left recursive
10. Consider the following grammar: $S \rightarrow abS \mid acS \mid c$

a. Compute First sets for each production and nonterminal

First sets of productions overlap

b. Show why the grammar cannot be parsed by a LL(1) parser.
   
   First sets of productions overlap
   
   c. Rewrite the grammar so it can be parsed by a LL(1) parser.
   
   d. Write a recursive descent parser for the rewritten grammar.

```
parse_S( ) {
    if (lookahead == "a") {
        match("a");  // S \rightarrow aL
        parse_L( );
    }
    else if (lookahead == "c")
        match("c");  // S \rightarrow c
    else error( );
}

parse_L( ) {
    if (lookahead == "b") {
        match("b");  // L \rightarrow bS
        parse_S( );
    }
    else if (lookahead == "c") {
        match("c");  // L \rightarrow cS
        parse_S( );
    }
    else error( );
}
```
11. Consider the following grammar: $S \rightarrow Sa \mid Sc \mid c$
   a. Show why the grammar cannot be parsed by a predictive parser.
      **First sets of productions intersect, grammar is left recursive**
   b. Rewrite the grammar so it can be parsed by a predictive parser.
      $$ S \rightarrow cL \quad L \rightarrow aL \mid cL \mid \epsilon $$
   c. Using \textsc{First} and \textsc{Follow}, build the \textsc{LL}(1) table for the new grammar
      $$ \textsc{First}^+(L \rightarrow \epsilon) = \textsc{Follow}(L) = \{$$
   d. Using the \textsc{LL}(1) table, show how a table-driven parser parses the string “caca”
      \begin{tabular}{|c|c|c|}
      \hline
      Stack & Input & Action \\
      \hline
      [\$, S] & caca$ & S \rightarrow cL, pop S + push L c \\
      [\$, L, c] & caca$ & next input + pop c \\
      [\$, L] & aca$ & L \rightarrow aL, pop L + push L a \\
      [\$, L, a] & aca$ & next input + pop a \\
      [\$, L] & ca$ & L \rightarrow cL, pop L + push L c \\
      [\$, L, c] & ca$ & next input + pop c \\
      [\$, L] & a$ & L \rightarrow aL, pop L + push L a \\
      [\$, L, a] & a$ & next input + pop a \\
      [\$, L] & $ & L \rightarrow \epsilon, pop L \\
      [\$] & $ & accept \\
      \hline
      \end{tabular}
   e. Write a recursive descent parser for your new grammar
      ```
      parse_S( ) {
          if (lookahead == “c”) {
              match(“c”); // S \rightarrow cL
              parse_L();
          }
          else error();
      }
      parse_L( ) {
          if (lookahead == “a”) {
              match(“a”); // L \rightarrow aL
              parse_L();
          }
          else if (lookahead == “c”) {
              match(“c”); // L \rightarrow cL
              parse_L();
          }
          else if (lookahead == $)
              ; // L \rightarrow \epsilon since \textsc{Follow}(L) = \$
          else error();
      }
      ```
12. Consider the following grammar

```
Goal    →  Expr
Expr    →  Term + Expr
|       Term – Expr
|       Term
Term    →  Factor * Term
|       Factor / Term
|       Factor
Factor  →  num
|       id
```

a. Apply left factoring to create a LL(1) grammar

b. Compute FIRST and FOLLOW for the modified grammar

<table>
<thead>
<tr>
<th>Left Factored Grammar</th>
<th>FIRST</th>
<th>FOLLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal</strong> → Expr</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Expr</strong> → Term Expr’</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Expr’</strong> → + Expr’</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>– Expr’</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ε</td>
<td></td>
</tr>
<tr>
<td><strong>Term</strong> → Factor Term’</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Term’</strong> → * Term</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>/ Term</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ε</td>
<td></td>
</tr>
<tr>
<td><strong>Factor</strong> → num</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>id</td>
<td></td>
</tr>
</tbody>
</table>

c. Using FIRST and FOLLOW, build the LL(1) table for the new grammar

<table>
<thead>
<tr>
<th></th>
<th>num</th>
<th>id</th>
<th>+</th>
<th>–</th>
<th>*</th>
<th>/</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>Goal → Expr</td>
<td>Goal → Expr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expr</td>
<td>Expr → Term Expr’</td>
<td>Expr → Term Expr’</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expr’</td>
<td>Expr’ → + Expr’</td>
<td>Expr’ → – Expr’</td>
<td>Expr’ → ε</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Term</td>
<td>Term → Factor Term’</td>
<td>Term → Factor Term’</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Term’</td>
<td>Term’ → ε</td>
<td>Term’ → ε</td>
<td>Term’ → * Term</td>
<td>Term’ → / Term</td>
<td>Term’ → ε</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Factor</td>
<td>Factor → num</td>
<td>Factor → id</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>