CMSC 330, Practice Problem 3 Solutions

1. Context Free Grammars
   a. List the 4 components of a context free grammar.
      - Terminals, non-terminals, productions, start symbol
   b. Describe the relationship between terminals, non-terminals, and productions.
      - Productions are rules for replacing a single non-terminal with a string of terminals and non-terminals
   c. Define ambiguity.
      - Multiple left-most (or right-most) derivations for the same string
   d. Describe the difference between scanning & parsing.
      - Scanning matches input to regular expressions to produce terminals, parsing matches terminals to grammars to create parse trees

2. Describing Grammars
   a. Describe the language accepted by the following grammar:
      \[ S \rightarrow abS | a (ab)^n a \]
   b. Describe the language accepted by the following grammar:
      \[ S \rightarrow aSb | \epsilon \]
      \[ a^n b^n, n \geq 0 \]
   c. Describe the language accepted by the following grammar:
      \[ S \rightarrow bSb | A \]
      \[ A \rightarrow aA | \epsilon \]
      \[ b^n a^n b^n, n \geq 0 \]
   d. Describe the language accepted by the following grammar:
      \[ S \rightarrow AS | B \]
      \[ A \rightarrow aAc | Aa | \epsilon \]
      \[ B \rightarrow bBb | \epsilon \]
      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
   e. Describe the language accepted by the following grammar:
      \[ S \rightarrow S and S | S or S | (S) | true | false \]
      Boolean expressions of true & false separated by and & or, with some expressions enclosed in parentheses
   f. Which of the previous grammars are left recursive?
      2d, 2e
   g. Which of the previous grammars are right recursive?
      2a, 2c, 2d, 2e
   h. Which of the previous grammars are ambiguous? Provide proof.
      Examples of multiple left-most derivations for the same string
      2d:
      \[ S \Rightarrow AS \Rightarrow AaS \Rightarrow aS \Rightarrow aB \Rightarrow a \]
      \[ S \Rightarrow AS \Rightarrow S \Rightarrow AS \Rightarrow AaS \Rightarrow aS \Rightarrow aB \Rightarrow a \]
      2c:
      \[ S \Rightarrow S and S \Rightarrow S and S and S \Rightarrow true and S and S \]
      \[ \Rightarrow true and true and S \Rightarrow true and true and S \]
      \[ \Rightarrow true and true and S \Rightarrow true and true and true \]
3. Creating Grammars
   a. Write a grammar for $a^x b^y$, where $x = y$
      
      $S \rightarrow aSb | \varepsilon$
   
   b. Write a grammar for $a^x b^y$, where $x > y$
      
      $S \rightarrow aL$
      $L \rightarrow aL | aLb | \varepsilon$
   
   c. Write a grammar for $a^x b^y$, where $x = 2y$
      
      $S \rightarrow aaSb | \varepsilon$
   
   d. Write a grammar for $a^x b^y a^z$, where $z = x + y$
      
      $S \rightarrow aL$
      $L \rightarrow aL | aLb | \varepsilon$
   
   e. Write a grammar for $a^x b^y a^z$, where $z = x - y$
      
      $S \rightarrow aSb | \varepsilon$
   
   f. Write a grammar for all strings of $a$ and $b$ that are palindromes.
      
      $S \rightarrow aSa | bSb | L$
      $L \rightarrow a | b | \varepsilon$
   
   g. Write a grammar for all strings of $a$ and $b$ that include the substring $baa$.
      
      $S \rightarrow LbaaL$
      $L \rightarrow a | b | \varepsilon$
      // $L = \text{any}$
   
   h. Write a grammar for all strings of $a$ and $b$ with an odd number of $a$’s and $b$’s.
      
      $S \rightarrow EaEbE | EbEaE | E \rightarrow EaEaE | EbEbE | \varepsilon | SS$
      // $E$ = even #s
   
   i. Write a grammar for the “if” statement in OCaml
      
      $S \rightarrow \text{ if } E \text{ then } E \text{ else } E | \text{ if } E \text{ then } E$
      $E \rightarrow S | \text{ expr}$
   
   j. Write a grammar for all lists in OCaml
      
      $S \rightarrow | | \text{ E | E::S E } \rightarrow \text{ elem } | S$
      // Ignores types, allows lists of lists
   
   k. Which of your grammars are ambiguous? Can you come up with an
      unambiguous grammar that accepts the same language?
      
      Grammar for 3h is ambiguous. An unambiguous grammar must exist
      since the language can be recognized by a deterministic finite automaton,
      and DFA -> RE -> Regular Grammar.
      Grammar for 3i is ambiguous. Multiple derivations for “if expr then if
      expr then expr else expr”. It is possible to write an unambiguous
      grammar by restricting some $S$ so that no unbalanced if statement can be
      produced.

4. Derivations, Parse Trees, Precedence and Associativity
   For the following grammar: $S \rightarrow S$ and $S | \text{ true}$
   
   a. List 4 derivations for the string “true and true and true”.
      
      i. $S => \underline{S}$ and $S => \underline{S}$ and $S$ and $S => \text{ true and } S$ and $S$ => $\text{ true and true}$
         and $S$ => $\text{ true and true and true}$
      
      ii. $S => \underline{S}$ and $S => \text{ true and } S$ => $\text{ true and true}$ and $S$ => $\text{ true and true}$
      
      iii. $S$ => $\underline{S}$ and $S$ => $\text{ true and } S$ => $\text{ true and true}$ and $S$ => $\text{ true and true}$
      
      iv. $S$ => $\underline{S}$ and $S$ => $\text{ true and } S$ => $\text{ true and true}$ and $S$ => $\text{ true and true}$
      
      v. $S$ => $\underline{S}$ and $S$ => $\text{ true and } S$ => $\text{ true and true}$ and $S$ => $\text{ true and true}$
      
      vi. $S$ => $\underline{S}$ and $S$ => $\text{ true and true}$ and $S$ => $\text{ true and true}$ and $S$ => $\text{ true and true}$
      
      and $S$ => $\text{ true and true and true}$
vii. $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and true and $S \Rightarrow S$ and true and true

viii. $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and true and $S \Rightarrow S$ and true and true

ix. $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and true and true

x. $S \Rightarrow S$ and $S \Rightarrow true$ and $S \Rightarrow true$ and true and true

xi. $S \Rightarrow S$ and $S \Rightarrow S$ and true and $S \Rightarrow true$ and true and true

xii. $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow true$ and $S \Rightarrow S$ and true and true

xiii. $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow true$ and true and true

xiv. $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and true and true

xv. $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and true and true

xvi. $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow S$ and $S \Rightarrow true$ 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

\[
\text{Tree 1}
\]

\[
\text{Tree 2}
\]

d. What is implied about the associativity of “and” for each parse tree?

Tree 1 => and is right-associative, Tree 2 => and is left-associative

For the following grammar: $S \rightarrow S \ and \ S \mid S \ or \ S \mid true$

e. List all parse trees for the string “true and true or true”
f. What is implied about the precedence/associativity of “and” and “or” for each parse tree?
   
   Tree 1 => or has higher precedence than and
   Tree 2 => and has higher precedence than or

   g. Rewrite the grammar so that “and” has higher precedence than “or” and is right associative

   \[ S \rightarrow S \text{ or } S \mid L \] // op closer to Start = lower precedence op
   \[ L \rightarrow \text{true and } L \mid \text{true} \] // right recursive = right associative

5. Left factoring & eliminating left recursion
   Rewrite the following grammars so they can be parsed by a predicative parser by eliminating left recursion and applying left factoring where necessary

   a. \[ S \rightarrow S + a \mid b \]
      \[ S \rightarrow b \ L \]
      \[ L \rightarrow + a L \mid \varepsilon \]

   b. \[ S \rightarrow S + a \mid S + b \mid c \]
      \[ S \rightarrow c \ L \]
      \[ L \rightarrow + a L \mid + b L \mid \varepsilon \]

   c. \[ S \rightarrow S + a \mid S + b \mid \varepsilon \]
      \[ S \rightarrow L \]
      \[ L \rightarrow + a L \mid + b L \mid \varepsilon \]

   d. \[ S \rightarrow a \ b \ c \mid a \ c \]
      \[ S \rightarrow a \ L \]
      \[ L \rightarrow b \ c \mid c \]
e. \[ S \rightarrow a \, b \, c \mid a \, b \, b \]  
\[ \downarrow \]  
\[ S \rightarrow a \, b \, L \]  
\[ L \rightarrow c \mid b \]  
f. \[ S \rightarrow a \, b \, c \mid a \, b \]  
\[ \downarrow \]  
\[ S \rightarrow a \, b \, L \]  
\[ L \rightarrow c \mid \varepsilon \]  
g. \[ S \rightarrow a \, a \mid a \, b \mid a \, c \]  
\[ \downarrow \]  
\[ S \rightarrow a \, L \]  
\[ L \rightarrow a \mid b \mid c \]  
h. \[ S \rightarrow a \, a \mid a \, b \mid a \]  
\[ \downarrow \]  
\[ S \rightarrow a \, L \]  
\[ L \rightarrow a \mid b \mid \varepsilon \]  
i. \[ S \rightarrow a \, a \mid a \, b \mid \varepsilon \]  
\[ \downarrow \]  
\[ S \rightarrow a \, L \mid \varepsilon \]  
\[ L \rightarrow a \mid b \]  
j. \[ S \rightarrow a \, S \, c \mid a \, S \, b \mid b \]  
\[ \downarrow \]  
\[ S \rightarrow a \, S \, L \mid b \]  
\[ L \rightarrow c \mid b \]  
k. \[ S \rightarrow a \, S \, c \mid a \, S \, b \mid a \]  
\[ \downarrow \]  
\[ S \rightarrow a \, L \]  
\[ L \rightarrow S \, c \mid S \, b \mid \varepsilon \]  
\[ \downarrow \]  
\[ S \rightarrow a \, L \]  
\[ L \rightarrow S \, M \mid \varepsilon \]  
\[ M \rightarrow c \mid \varepsilon \]  
l. \[ S \rightarrow a \, S \, c \mid a \, S \mid a \]  
\[ \downarrow \]  
\[ S \rightarrow a \, L \]  
\[ L \rightarrow S \, c \mid S \mid \varepsilon \]  
\[ \downarrow \]  
\[ S \rightarrow a \, L \]  
\[ L \rightarrow S \, M \mid \varepsilon \]  
\[ M \rightarrow c \mid \varepsilon \]
6. Parsing

For the problem, assume the term “predictive parser” refers to a top-down, recursive descent, non-backtracking predictive parser.

a. Consider the following grammar: $S \rightarrow S$ and $S \mid S \mid (S) \mid$ true $\mid$ false

   i. 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” }

      ii. Explain why the grammar cannot be parsed by a predictive parser

         **First sets of productions intersect, grammar is left recursive**

b. Consider the following grammar: $S \rightarrow abS \mid acS \mid c$

   i. Compute First sets for each production and nonterminal
      
      First(abS) = { a }
      First(acS) = { a }
      First(c) = { c }
      First(S) = { a, c }

   ii. Show why the grammar cannot be parsed by a predictive parser.

      **First sets of productions overlap**
      
      First(abS) $\cap$ First(acS) = { a } $\cap$ { a } = { a } $\neq$

   iii. Rewrite the grammar so it can be parsed by a predictive parser.

      $S \rightarrow aL \mid c$    
      $L \rightarrow bS \mid cS$

   iv. Write a predictive parser for the rewritten grammar.

   ```java
   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();
   }
   ```
c. Consider the following grammar: 
\[ S \rightarrow Sa | Sc | c \]
   i. Show why the grammar cannot be parsed by a predictive parser.  
   \textbf{First sets of productions intersect, grammar is left recursive}
   ii. Rewrite the grammar so it can be parsed by a predictive parser.
\[ S \rightarrow cL \quad L \rightarrow aL | cL | \varepsilon \]
   iii. Write a recursive descent parser for your new grammar

```c
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 ; // L \rightarrow \varepsilon
}
```

d. Describe an abstract syntax tree (AST)
   \textbf{Compact representations of parse trees with only essential parts}

7. Automata
   a. Describe regular grammars.  
      \textbf{Grammars where all productions are of the form X \rightarrow a or X \rightarrow aY}
   b. Describe the relationship between regular grammars and regular expressions.  
      \textbf{Regular grammars are exactly as powerful as regular expressions (and one can be converted to the other)}
   c. Name features needed by automata to recognize
      i. Regular languages (i.e., languages recognized by regular grammars)  
         \textbf{DFA (automaton with finite # of states and transitions)}
      ii. Context-free languages  
         \textbf{NFA and 1 stack}
      iii. All binary numbers  
         \textbf{DFA (binary #s can be recognized by RE)}
      iv. All binary numbers divisible by 2  
         \textbf{DFA (binary #s ending in 0 can be recognized by RE)}
   d. Compare finite automata and pushdown automata  
      \textbf{Pushdown automata are finite automata that can use 1 stack, Pushdown automata > finite automata in terms of computing power.}