CMSC 330 Practice Problem 4 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
   e. Describe an abstract syntax tree (AST)
      Compact representations of parse trees with only essential parts

2. Describing Grammars
   a. Describe the language accepted by the following grammar:
      $S \rightarrow abS \mid a$
      (ab)*a
   b. Describe the language accepted by the following grammar:
      $S \rightarrow aSb \mid \varepsilon$
      $\mathbf{a^n b^n, n \geq 0}$
   c. Describe the language accepted by the following grammar:
      $S \rightarrow bSb \mid A$ $\quad$ $A \rightarrow aA \mid \varepsilon$
      $\mathbf{b^n a^* b^n, n \geq 0}$
   d. Describe the language accepted by the following grammar:
      $S \rightarrow AS \mid B$ $\quad$ $A \rightarrow aAc \mid Aa \mid \varepsilon$ $\quad$ $B \rightarrow bBb \mid \varepsilon$
      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
   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 => AS => AaS => aS => aB => a
      S => AS => S => AS => AaS => aS => aB => a
      2e: S => S and S => S and S and S => true and S and S
      => true and true and S => true and true and true
      S => S and S => true and S => true and S and S
      => true and true and S => true and true and true

3. Creating Grammars
   a. Write a grammar for $a^x b^y$, where x = y
      S → aSb | ε
   b. Write a grammar for $a^x b^y$, where x > y
      S → aL   L → aL | aLb | ε
   c. Write a grammar for $a^x b^y$, where x = 2y
      S → aaSb | ε
   d. Write a grammar for $a^x b^y a^z$, where z = x+y
      S → aSa | L   L → bLa | ε
   e. Write a grammar for $a^x b^y a^z$, where z = x-y
      S → aSa | L   L → aLb | ε
   f. Write a grammar for all strings of *a* and *b* that are palindromes.
      S → aSa | bSb | L   L → a | b | ε
   g. Write a grammar for all strings of *a* and *b* that include the substring *baa*.
      S → LbaaL   L → aL | bL | ε   // L = any
   h. Write a grammar for all strings of *a* and *b* with an odd number of *a*'s and an odd number of *b*'s.
      S → EaEbE | EbEaE   E → EaEaE | EbEbE | ε | SS   // E = even #s
   i. Write a grammar for the "if" statement in OCaml
      S → if E then E else E | if E then E   E → S | expr
   j. Write a grammar for all lists in OCaml
      S → [] | [E] | E::S   E → 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 → S and S | true
   a. List 4 derivations for the string "true and true and true".
      i. S => <u>S</u> and S => <u>S</u> and S and S => true and <u>S</u> and S => true and true and S => true and true and true
      ii. S => <u>S</u> and S => true and S => true and <u>S</u> and S => true and true and S => true and true and true
      iii. S => S and <u>S</u> => S and true => S and <u>S</u> and true => S and true and true => true and true and true
      iv. S => S and <u>S</u> => S and S and <u>S</u> => S and <u>S</u> and true => S and true and true => true and true and true
      v. S => <u>S</u> and S => <u>S</u> and S and S => true and S and <u>S</u> => true and S and true => true and true and true

vi. **S => $\underline{S}$ and S => S and $\underline{S}$ and S => $\underline{S}$ and true and S => true and true and S => true and true and true**

vii. **S => $\underline{S}$ and S => S and $\underline{S}$ and S => S and true and $\underline{S}$ => S and true and true => true and true and true**

viii. **S => $\underline{S}$ and S => S and S and $\underline{S}$ => $\underline{S}$ and S and true => true and S and true => true and true and true**

ix. **S => $\underline{S}$ and S => S and S and $\underline{S}$ => S and $\underline{S}$ and true => S and true and true => true and true and true**

x. **S => $\underline{S}$ and S => true and S => true and S and $\underline{S}$ => true and S and true => true and true and true**

xi. **S => S and $\underline{S}$ => S and true => $\underline{S}$ and S and true => true and S and true => true and true and true**

xii. **S => S and $\underline{S}$ => $\underline{S}$ and S and S => true and $\underline{S}$ and S => true and true and S => true and true and true**

xiii. **S => S and $\underline{S}$ => $\underline{S}$ and S and S => true and S and $\underline{S}$ => true and S and true => true and true and true**

xiv. **S => S and $\underline{S}$ => S and $\underline{S}$ and S =>$\underline{S}$ and true and S => true and true and S => true and true and true**

xv. **S => S and $\underline{S}$ => S and $\underline{S}$ and S => S and true and $\underline{S}$ => S and true and true => true and true and true**

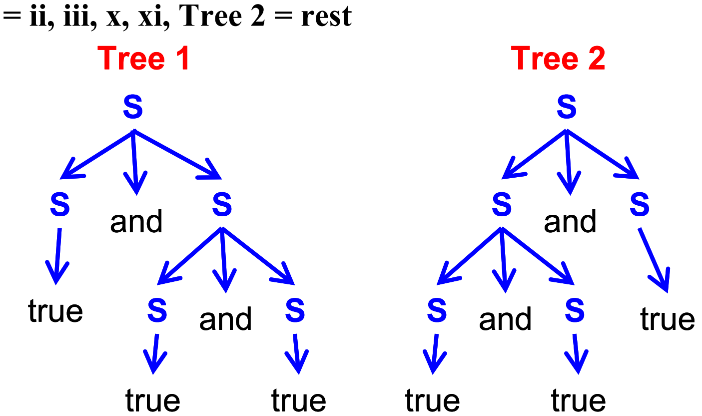
xvi. **S => S and $\underline{S}$ => S and S and $\underline{S}$ => $\underline{S}$ and S and true => true and S and true => true and true and true**

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

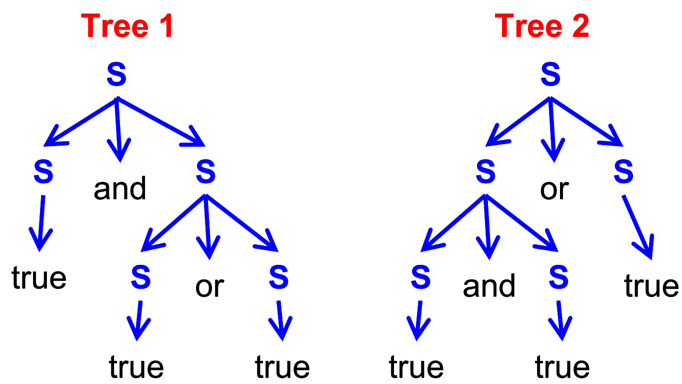


d. What is implied about the associativity of "and" for each parse tree?

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

For the following grammar: S → S and S | S or S | 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 → S or S | L **// op closer to Start = lower precedence op**
   L → true and L | true **// right recursive = right associative**

5. Left factoring

   Rewrite the following grammars so they can be parsed by a predicative parser by applying left factoring where necessary

   a. S → a b c | a c
      ↓
      S → a L
      L→ b c | c

   b. S → a a | a b | a
      ↓
      S → a L
      L→ a | b | ε

   c. S → a b A c | a b B a
      ↓
      S → a b L
      L→ A c | B a

   d. S → a a A | a a a B | a c
      ↓
      S → a L
      L→ a A | a a B | c
      ↓
      S → a L
      L→ a M | c
      M → A | a B

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 → S and S | S or S | (S) | true | 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 → abS | acS | 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) ∩ First(acS) = { a } ∩ { a } = { a } ≠ □

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

         S → aL | c   L → bS | cS

      iv. Write a predictive parser for the rewritten grammar.

```
parse_S( ) {
   if (lookahead == "a") {
      match("a");   // S → aL
      parse_L( );
   }
   else if (lookahead == "c")
      match("c");   // S → c
   }
   else error( );
}
parse_L( ) {
   if (lookahead == "b") {
      match("b");   // L → bS
      parse_S( );
   }
   else if (lookahead == "c") {
      match("c");   // L → cS
      parse_S( );
   }
   else error( );
}
```

c. Consider the following grammar: S → Sa | Sc | c

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

First sets of productions intersect, grammar is left recursive

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

S → c L **L → aL | cL | ε**

iii. Write a recursive descent parser for your new grammar

```
parse_S( ) {
   if (lookahead == "c") {
      match("c");   // S → cL
      parse_L( );
   }
   else error( );
}
parse_L( ) {
   if (lookahead == "a") {
      match("a");   // L → aL
      parse_L( );
   }
   else if (lookahead == "c") {
      match("c");   // L → cL
      parse_L( );
   }
   else ;            // L →ε
}
```