

CMSC 330, Fall 2009, 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:
 $S \rightarrow abS \mid a$
 $(ab)^*a$
 - b. Describe the language accepted by the following grammar:
 $S \rightarrow aSb \mid \epsilon$
 $a^n b^n, n \geq 0$
 - c. Describe the language accepted by the following grammar:
 $S \rightarrow bSb \mid A \quad A \rightarrow aA \mid \epsilon$
 $b^n a^* b^n, n \geq 0$
 - d. Describe the language accepted by the following grammar:
 $S \rightarrow AS \mid B \quad A \rightarrow aAc \mid Aa \mid \epsilon \quad B \rightarrow bBb \mid \epsilon$
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:
 $S \rightarrow S \text{ and } S \mid S \text{ or } S \mid (S) \mid \text{true} \mid \text{false}$
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: $S \Rightarrow AS \Rightarrow AaS \Rightarrow aS \Rightarrow aB \Rightarrow a$
 $S \Rightarrow AS \Rightarrow S \Rightarrow AS \Rightarrow AaS \Rightarrow aS \Rightarrow aB \Rightarrow a$
2e: $S \Rightarrow S \text{ and } S \Rightarrow S \text{ and } S \text{ and } S \Rightarrow \text{true and } S \text{ and } S$
 $\Rightarrow \text{true and true and } S \Rightarrow \text{true and true and true}$
 $S \Rightarrow S \text{ and } S \Rightarrow \text{true and } S \Rightarrow \text{true and } S \text{ and } S$
 $\Rightarrow \text{true and true and } S \Rightarrow \text{true and true and true}$

3. Creating Grammars

- Write a grammar for $a^x b^y$, where $x = y$
 $S \rightarrow aSb \mid \epsilon$
- Write a grammar for $a^x b^y$, where $x > y$
 $S \rightarrow aL \quad L \rightarrow aL \mid aLb \mid \epsilon$
- Write a grammar for $a^x b^y$, where $x = 2y$
 $S \rightarrow aaSb \mid \epsilon$
- Write a grammar for $a^x b^y a^z$, where $z = x+y$
 $S \rightarrow aSa \mid L \quad L \rightarrow bLa \mid \epsilon$
- Write a grammar for $a^x b^y a^z$, where $z = x-y$
 $S \rightarrow aSa \mid L \quad L \rightarrow aLb \mid \epsilon$
- Write a grammar for all strings of a and b that are palindromes.
 $S \rightarrow aSa \mid bSb \mid L \quad L \rightarrow a \mid b \mid \epsilon$
- Write a grammar for all strings of a and b that include the substring baa .
 $S \rightarrow LbaaL \quad L \rightarrow aL \mid bL \mid \epsilon \quad // L = \text{any}$
- Write a grammar for all strings of a and b with an odd number of a 's and an odd number of b 's.
 $S \rightarrow EaEbE \mid EbEaE \quad E \rightarrow EaEaE \mid EbEbE \mid \epsilon \mid SS \quad // E = \text{even \#s}$
- Write a grammar for the “if” statement in OCaml
 $S \rightarrow \text{if } E \text{ then } E \text{ else } E \mid \text{if } E \text{ then } E \quad E \rightarrow S \mid \text{expr}$
- Write a grammar for all lists in OCaml
 $S \rightarrow [] \mid [E] \mid E::S \quad E \rightarrow \text{elem} \mid S \quad // \text{Ignores types, allows lists of lists}$
- 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 \rightarrow RE \rightarrow 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.

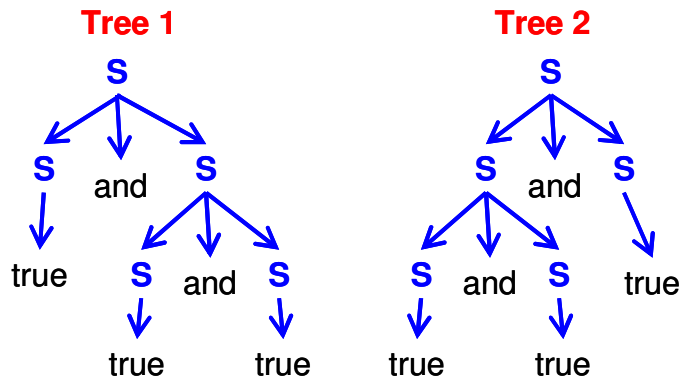
4. Derivations, Parse Trees, Precedence and Associativity

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

- List 4 derivations for the string “true and true and true”.
 - $S \Rightarrow \underline{S} \text{ and } S \Rightarrow \underline{S} \text{ and } S \text{ and } S \Rightarrow \text{true and } \underline{S} \text{ and } S \Rightarrow \text{true and true and } S \Rightarrow \text{true and true and true}$
 - $S \Rightarrow \underline{S} \text{ and } S \Rightarrow \text{true and } S \Rightarrow \text{true and } \underline{S} \text{ and } S \Rightarrow \text{true and true and } S \Rightarrow \text{true and true and true}$
 - $S \Rightarrow S \text{ and } \underline{S} \Rightarrow S \text{ and true} \Rightarrow S \text{ and } \underline{S} \text{ and true} \Rightarrow S \text{ and true and true} \Rightarrow \text{true and true and true}$
 - $S \Rightarrow S \text{ and } \underline{S} \Rightarrow S \text{ and } S \text{ and } \underline{S} \Rightarrow S \text{ and } \underline{S} \text{ and true} \Rightarrow S \text{ and true and true} \Rightarrow \text{true and true and true}$
 - $S \Rightarrow \underline{S} \text{ and } S \Rightarrow \underline{S} \text{ and } S \text{ and } S \Rightarrow \text{true and } S \text{ and } \underline{S} \Rightarrow \text{true and } S \text{ and true} \Rightarrow \text{true and true and true}$

- vi. $S \Rightarrow \underline{S}$ and $S \Rightarrow S$ and \underline{S} and $S \Rightarrow \underline{S}$ and true and $S \Rightarrow$ true and true and $S \Rightarrow$ true and true and true
- vii. $S \Rightarrow \underline{S}$ and $S \Rightarrow S$ and \underline{S} and $S \Rightarrow S$ and true and $\underline{S} \Rightarrow S$ and true and true \Rightarrow true and true and true
- viii. $S \Rightarrow \underline{S}$ and $S \Rightarrow S$ and S and $\underline{S} \Rightarrow \underline{S}$ and S and true \Rightarrow true and S and true \Rightarrow true and true and true
- ix. $S \Rightarrow \underline{S}$ and $S \Rightarrow S$ and S and $\underline{S} \Rightarrow S$ and \underline{S} and true $\Rightarrow S$ and true and true \Rightarrow true and true and true
- x. $S \Rightarrow \underline{S}$ and $S \Rightarrow$ true and $S \Rightarrow$ true and S and $\underline{S} \Rightarrow$ true and S and true \Rightarrow true and true and true
- xi. $S \Rightarrow S$ and $\underline{S} \Rightarrow S$ and true $\Rightarrow \underline{S}$ and S and true \Rightarrow true and S and true \Rightarrow true and true and true
- xii. $S \Rightarrow S$ and $\underline{S} \Rightarrow \underline{S}$ and S and $S \Rightarrow$ true and \underline{S} and $S \Rightarrow$ true and true and $S \Rightarrow$ true and true and true
- xiii. $S \Rightarrow S$ and $\underline{S} \Rightarrow \underline{S}$ and S and $S \Rightarrow$ true and S and $\underline{S} \Rightarrow$ true and S and true \Rightarrow true and true and true
- xiv. $S \Rightarrow S$ and $\underline{S} \Rightarrow S$ and \underline{S} and $S \Rightarrow \underline{S}$ and true and $S \Rightarrow$ true and true and $S \Rightarrow$ true and true and true
- xv. $S \Rightarrow S$ and $\underline{S} \Rightarrow S$ and \underline{S} and $S \Rightarrow S$ and true and $\underline{S} \Rightarrow S$ and true and true \Rightarrow true and true and true
- xvi. $S \Rightarrow S$ and $\underline{S} \Rightarrow S$ and S and $\underline{S} \Rightarrow \underline{S}$ and S and true \Rightarrow true and S and true \Rightarrow true and true and true

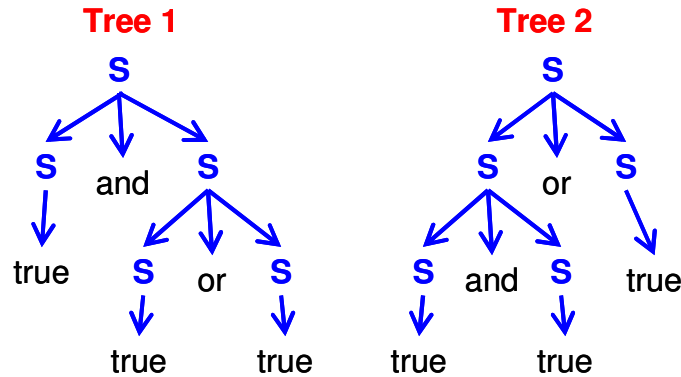
- b. Label each derivation as left-most, right-most, or neither.
i and ii are left-most derivations, iii and iv are right-most derivations, remaining derivations are neither
- c. List the parse tree for each derivation
Tree 1 = ii, iii, x, xi, Tree 2 = rest



- d. What is implied about the associativity of “and” for each parse tree?
Tree 1 \Rightarrow and is right-associative, Tree 2 \Rightarrow and is left-associative

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

- 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 predictive parser by eliminating left recursion and applying left factoring where necessary

a. $S \rightarrow S + a \mid b$

↓
 $S \rightarrow b L$
 $L \rightarrow + a L \mid \epsilon$

b. $S \rightarrow S + a \mid S + b \mid c$

↓
 $S \rightarrow c L$
 $L \rightarrow + a L \mid + b L \mid \epsilon$

↓
 $S \rightarrow c L$
 $L \rightarrow + M \mid \epsilon$
 $M \rightarrow a L \mid b L$

c. $S \rightarrow a b c \mid a c$

↓
 $S \rightarrow a L$
 $L \rightarrow b c \mid c$

d. $S \rightarrow a a \mid a b \mid a$

↓
 $S \rightarrow a L$
 $L \rightarrow a \mid b \mid \epsilon$

e. $S \rightarrow a S c \mid a S b \mid b$

↓
 $S \rightarrow a S L \mid b$
 $L \rightarrow c \mid b$

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: $S \rightarrow S \text{ and } S \mid S \text{ or } S \mid (S) \mid \text{true} \mid \text{false}$

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 \emptyset

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.

```
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: $S \rightarrow Sa \mid Sc \mid c$
- Show why the grammar cannot be parsed by a predictive parser.
First sets of productions intersect, grammar is left recursive
 - Rewrite the grammar so it can be parsed by a predictive parser.

$S \rightarrow cL$ $L \rightarrow aL \mid cL \mid \epsilon$

- 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
}

```

- Describe an abstract syntax tree (AST)

Compact representations of parse trees with only essential parts

7. Automata

- Compare finite automata and pushdown automata

Pushdown automata are finite automata that can use 1 stack.

Pushdown automata > finite automata in terms of computing power.