

# CMSC 430

## Practice Problems

---

1. Describe the languages denoted by the following regular expressions:

- (a)  $0(0|1)^*0$
  - (b)  $((\epsilon|0)1^*)^*$
  - (c)  $(0|1)^*0(0|1)(0|1)$
- 

2. Write regular expressions for the following languages.

- (a) All strings of 0's and 1's that do not contain the substring 011.
- (b) All strings of 0's and 1's that do not contain the subsequence 011.

Substring is any contiguous sequence of characters found in a string. Subsequence is any sequence of characters (contiguous or non-contiguous) found in a string. For instance, the string "110010" contains the substrings "110" (first 3 characters), "100" (2nd, 3rd, 4th characters), and "010" (last 3 characters). It contains the subsequences "111" (1st, 2nd, 5th characters) and "000" (3rd, 4th, 6th characters), though neither are substrings.

---

3. For the regular expressions  $(a|b)^*$  and  $(a^*|b^*)^*$

- (a) Construct an NFA using Thompson's construction algorithm
  - (b) Show the sequence of moves in parsing "ababab"
  - (c) Convert the NFA to a DFA using subset construction algorithm
  - (d) Minimize the DFA using the partitioning algorithm
- 

4. Write a grammar for the following languages. Are the grammars ambiguous?:

- (a) All strings of 0's and 1's that have the same number of 0's and 1's.
- (b) All strings of 0's and 1's that have more 0's than 1's.
- (c) All balanced pairs of left and right parentheses (e.g., "()", "()()").

---

5. Consider the grammar

$S ::= ( L ) \mid a$   
 $L ::= L , S \mid S$

For each of the following strings "(a,a)" and "(a,(a,a))"

- (a) Find the parse tree
  - (b) Find the leftmost derivation
  - (c) Find the rightmost derivation
- 

6. Consider the grammar

$S ::= aSbS \mid bSaS \mid \epsilon$

- (a) Show grammar is ambiguous by constructing two leftmost derivations for "abab"
  - (b) Show grammar is ambiguous by constructing two rightmost derivations for "abab"
- 

7. Consider the grammar

$S ::= ( L ) \mid a$   
 $L ::= L , S \mid S$

- (a) Eliminate left recursion from the grammar
- (b) Compute FIRST and FOLLOW for each non-terminal
- (c) Build a non-backtracking recursive descent parser, given the following code:

```
tok; // current token

match(x) { // matches token
    if (tok != x) // if wrong token
        error(); // exit with error
    tok = getToken(); // get new token
}

parser() {
    tok = getToken(); // initialize
    S( ); // start symbol
    match("$"); // match EOF
}
```

- (d) Show an example parse of the string "(a,a)"

---

8. Consider the grammar

$S ::= AS \mid b$

$A ::= SA \mid a$

- Compute FIRST and FOLLOW for each non-terminal
- Construct the set of LR(0) items for the grammar
- Construct the set of LR(1) items for the grammar
- Construct the LR(1) action/goto tables for the grammar
- List any shift/reduce or reduce/reduce conflicts. What is the effect if we always shift for a shift/reduce conflict? What is the effect if we always reduce for a shift/reduce conflict?
- Show an example parse of the string "abab"

---

9. Consider the grammar

$S ::= Aa \mid bAc \mid Bc \mid bBa$

$A ::= d$

$B ::= d$

- Show that the grammar is not LALR(1)

---

10. Attribute grammars and syntax-directed translation

- What are the advantages of syntax-directed translation over context-free parsing?
- What are the differences between context-free and context-sensitive parsing?
- Rank the following techniques in terms of the languages (sets of strings) they can recognize: syntax-directed translation, attribute grammars, context-free grammars, context-sensitive grammars.
- In an attribute grammar, what are the differences between inherited and synthesized attributes?
- Write a small attribute grammar with one inherited and one synthesized attribute, labeling each.

---

11. Type expressions

Given the following C declarations:

```
int num(char *x, int chuk[4]);
typedef struct {
    int a, b;
} CELL;
CELL foo[100];
CELL *bar(int x, CELL y) { ... }
```

- write the type expression for "num"
- write the type expression for "foo"
- write the type expression for "bar"

---

12. Syntax-directed translation

Consider the following grammar productions. Assume you have an attribute E.odd which can be set to either true, false, or unknown, and an attribute CONST.val which is the value of the constant.

E	→	CONST	{ E.odd = ?? }
		ID	{ E.odd = unknown }
		E <sub>1</sub> + E <sub>2</sub>	{ E.odd = ?? }
		E <sub>1</sub> - E <sub>2</sub>	{ E.odd = ?? }
		E <sub>1</sub> * E <sub>2</sub>	{ E.odd = ?? }
		( E <sub>1</sub> )	{ E.odd = ?? }
		- E <sub>1</sub>	{ E.odd = ?? }

- Add rules to the attribute grammar to calculate E.odd for each grammar production.
- Provide the parse tree for the expression  $(5 + 2) - (4 * -x)$  and show the calculation for E.odd at each point.

---

13. Syntax-directed translation and type checking

Consider the following grammar, which generates expressions for a number of operators. Assume you have the attributes E.min and E.max which should be set to the minimum and maximum values for E, and an attribute CONST.val which is the value of the constant.

E	→	CONST	{ E.min = ?? ; E.max = ?? }
		ID	{ E.min = ID.min ; E.max = ID.max }
		E <sub>1</sub> + E <sub>2</sub>	{ E.min = ?? ; E.max = ?? }
		E <sub>1</sub> - E <sub>2</sub>	{ E.min = ?? ; E.max = ?? }
		E <sub>1</sub> * E <sub>2</sub>	{ E.min = ?? ; E.max = ?? }
		( E <sub>1</sub> )	{ E.min = ?? ; E.max = ?? }
		- E <sub>1</sub>	{ E.min = ?? ; E.max = ?? }

- Write the type-checking rules which calculates the range for each subexpression.
- Provide the parse tree for the expression  $(5 + 2) - (4 * -x)$  and show the calculation for E.min and E.max at each point, assuming you know x.min = -3 and x.max = 10.

---

#### 14. Syntax-directed translation and type checking

Consider the following grammar, which generates expressions formed by applying "+" to integer and floating point constants. When two integers are added, the result is integer, otherwise, it is a float.

$$E \rightarrow E + T \mid T$$
$$T \rightarrow \text{num} . \text{num} \mid \text{num}$$

- Give a syntax-directed definition to determine the type of each subexpression. Assign each symbol an attribute "type".
- Provide the parse tree for the express  $5 + 4 + 3.2 + 1$  and show the computation E.type and T.type at each point.

---

#### 15. Symbol table

Consider the following program in a lexically-scoped language such as C.

```
int x, y;
int foo( )
  { int z; { int y; { int x; } } }
int bar( )
  { int z; { int x; } { int y; // HERE } }
```

- What is nested lexical scoping?
- How can symbol tables handle nested lexical scoping?
- How does the compiler determine where each variable encountered during compilation is actually declared when handling nested lexical scoping?
- Use your answer to construct the logical state of the symbol table(s) for the example when the compiler reaches the point marked HERE.

---

#### 16. Run-time environment

- What is a frame used for?
- Name six items of of run-time information stored in a frame. For each item, identify whether its value is set before the procedure is called, during procedure execution, or right before procedure return.
- Name one type additional type of information only stored in a frame for Java programs.
- When can storage for a variable be allocated in a frame?
- What advantage is obtained when allocating variables in a frame?
- Name two advantages of managing memory allocation manually in the code.
- Name two advantages of managing memory allocation automatically in the run-time system.
- Name two methods of managing memory allocation automatically in the run-time system.

---

#### 17. Intermediate representations.

Consider the statements:

- $x := a + ( b*a )$
- $x := a - (( b + a ) * c )$

- Translate each into an AST
- Translate each into 3-address code
- Translate each into Java stack code
- Which representation is the most compact? Why?
- Which representation is easy to manipulate? Why?
- Which representation is hard to manipulate? Why?
- Which representation is closest to the input program? Why?