Concepts of Programming Languages, CSCI 305, Fall 2020
Exam 2, Oct. 9

Name _____

This is a closed book exam. You may not use notes, the text book, Internet, etc.

1. Circle ALL of the strings which are in the language described by

   $P \rightarrow aaP'b$
   $P' \rightarrow aaP'b \mid X$
   $X \rightarrow aX \mid bX \mid \varepsilon$ (4 pts.)

   a. aab
   b. aaaab
   c. aaaabba
   d. aabbabbb
   e. aaababab

2. The grammar for the context-free language

   $L = \{w\#x \mid \text{where } w^R \text{ is w in reverse, } w^R \text{ is a substring of x for w,x} \in \{0,1\}^*\}$

   a. $S \rightarrow XT \quad T \rightarrow 0T0 \mid 1T1 \mid X\# \quad X \rightarrow 0X \mid 1X \mid \varepsilon$
   b. $S \rightarrow XT \quad T \rightarrow 0T0 \mid 1T1 \mid \#X \quad X \rightarrow 0X \mid 1X \mid \varepsilon$
   c. $S \rightarrow TX \quad T \rightarrow 0T0 \mid 1T1 \mid X\# \quad X \rightarrow 0X \mid 1X \mid \varepsilon$
   d. $S \rightarrow TX \quad T \rightarrow 0T0 \mid 1T1 \mid \#X \quad X \rightarrow 0X \mid 1X \mid \varepsilon$
   e. None of the above

   (4 pts.)

3. What best describes why the following context-free grammar is ambiguous. (4 pts.)

   $S \rightarrow aE$
   $S \rightarrow bE$
   $E \rightarrow aEa$
   $E \rightarrow bEb$
   $E \rightarrow \varepsilon$

   a. An expression in the language has both a right-most and a left-most derivation
   b. Multiple productions are defined for the variables S and E
   c. When parsing a sting it is not clear which production should be used
   d. An expression in the language has more than one parse tree
   e. This grammar is not ambiguous

4. The following code snippet in C contains an error in the second line.

```
main ( ) {
        int;
        if (range <= 0 )
                amount +-;
        else
                amount --;
}
```

This error is most likely to be classified as a: (4 pts.)

a. syntax error detected by the lexical analyzer
b. syntax error detected by the parser
c. static semantic analysis error
d. dynamic semantic analysis error
e. compiler can't catch

5. The following code snippet in C contains an error (use of an undeclared variable).

```
main ( ) {
        int sum = 5;
        sun++;
    }
```

This error is most likely to be classified as a: (5 pts.)

a. syntax error detected by the lexical analyzer
b. syntax error detected by the parser
c. static semantic analysis error
d. dynamic semantic analysis error
e. compiler can't catch

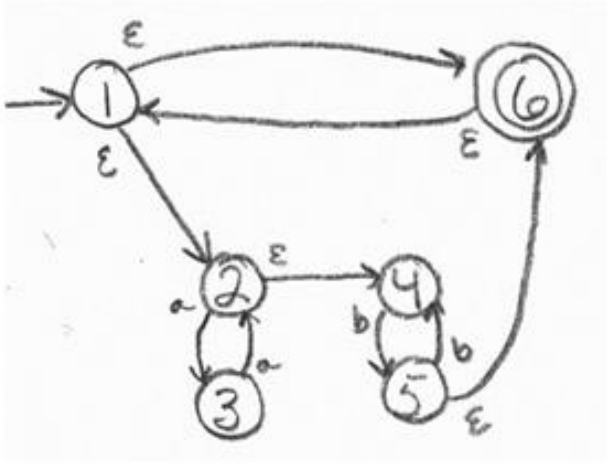
6. Write one or more regular expressions to identify the first three elements of URLs.

   URLs consist of an optional protocol indicator, IP address or hostname, optional port number, optional directory name, optional filename and optional query string.

   - Protocol indicator, if specified, is either http:// or https://
   - IP address is four numbers separated by three periods (The numbers should be between 0 and 255, but you just need to check for one to three digits.)
   - Hostnames are case-insensitive but allow them to be a series of names separated by periods (.). Each name starts with an upper or lower case character, followed by any number of upper or lower case characters or digits.

   (10 pts.)

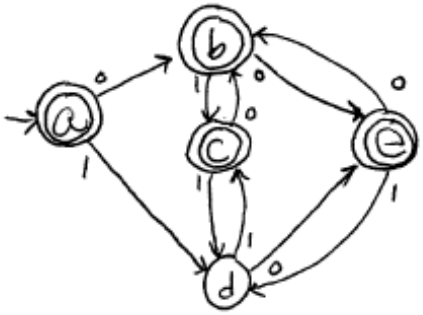
7. Use the construction described in class and the text to convert the following nondeterministic finite automaton (NFA) to an equivalent deterministic finite automaton (DFA). (10 pts.)



Create a table for this DFA, using made up variable names. (5 pts.)

8. Using the construction described in class and the text to convert the following DFA to a minimum DFA, if the DFA is not already a minimum DFA.

(10 pts.)



9. Create a grammar for the context-free language (5 pts.)

$L = \{w\# x_1 w^R x_2 \mid x_1, x_2, w \in \{0,1\}^* \}$

where w^R is the reverse of w

Strings in L	Strings not in L
#	ε
1#000100	1
011#000110000	0
1101#11011111	11#00
	10#10
	101#0000010

10. Consider a language of compound propositional statements that allows the propositions p, q, r and s, and the connectives $\wedge$, $\vee$, $\neg$, and $\rightarrow$. As usual, the connectives $\wedge$, $\vee$ and $\rightarrow$ are binary connectives (i.e. they have two operands), and the $\neg$ connective is a unary connective (it operates on one operand). Include parentheses in the language to force early evaluation.

Legal Compound Propositional Statements	**Illegal Compound Propositional Statements**
p	ε
$\neg$p	p q
$\neg\neg$p	$\neg\neg$
$\neg\neg\neg\neg$p	p$\vee\rightarrow$r
p$\vee\neg$q	p($\vee\rightarrow$r)
p$\vee\neg$q $\rightarrow$ r	()
p$\vee\neg$(q $\rightarrow$ r)	p$\vee\neg$q $\rightarrow$ r)

a. Create a grammar for the language of compound propositional statements. When giving the rules of your grammar, use $\alpha \Rightarrow \beta$, rather than $\alpha \rightarrow \beta$, to avoid confusion with the connective "$\rightarrow$". This grammar may be ambiguous. In part c, you are to create an unambiguous grammar. (5 pts.)

b. Show that your grammar can generate the following compound proposition

p$\vee\neg$(q $\rightarrow$ r)

by drawing a tree for it. (5 pts.)

c. Rewrite your grammar, if necessary, to adhere to the following order of operations:

$\neg$ - highest precedence
$\wedge$
$\vee$
$\rightarrow$

When an operator is repeated, use right associativity so the expression

$p \vee b \vee r$

is evaluated $p \vee (b \vee r)$ rather than $(p \vee b) \vee r$.

(10 pts.)

11. Show all pushes and pops which occur to a stack when parsing the code,
write (A*B) $$
given the following productions and parse table. (5 pts.)

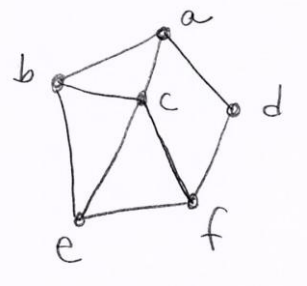
1. program → stmt_list $$
2. stmt_list → stmt stmt_list
3. stmt_list → ε
4. stmt → id := expr
5. stmt → read id
6. stmt → write expr
7. expr → term term_tail
8. term_tail → add_op term term_tail
9. term_tail → ε
10. term → factor factor_tail
11. factor_tail → mult_op factor factor_tail
12. factor_tail → ε
13. factor → (expr)
14. factor → id
15. factor → number
16. add_op → +
17. add_op → -
18. mult_op → *
19. mult_op → /

	id	**number**	**read**	**write**	**:=**	**(**	**)**	**+**	**-**	*****	**/**	**$**
program	1	-	1	1	-	-	-	-	-	-	-	1
stmt_list	2	-	2	2	-	-	-	-	-	-	-	3
stmt	4	-	5	6	-	-	-	-	-	-	-	-
expr	7	7	-	-	-	7	-	-	-	-	-	-
term_tail	9	-	9	9	-	-	9	8	8	-	-	9
term	10	10	-	-	-	10	-	-	-	-	-	-
factor_tail	12	-	12	12	-	-	12	12	12	11	11	12
factor	14	15	-	-	-	13	-	-	-	-	-	-
add_op	-	-	-	-	-	-	-	16	17	-	-	-
mult_op	-	-	-	-	-	-	-	-	-	18	19	-

Write programs in Scheme for the following. Do not use the computer to write these programs, simply write your program onto the test page.

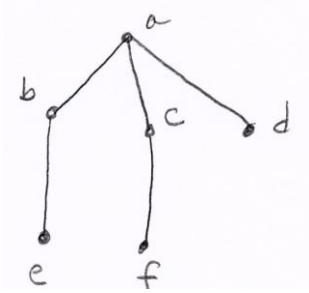
12. Sanjay plans to write a spanning tree function in Scheme and has decided that he needs some helper functions.

    Recall that a spanning tree of a graph, is a graph which contains all of the vertices in the original graph, however, it is also a tree. For example, the following graph

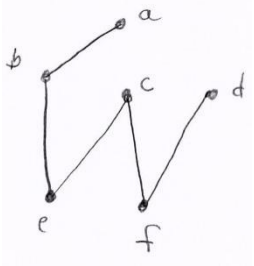
    

    has several spanning trees.

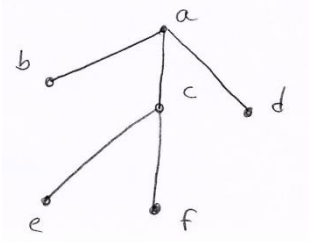
    The following spanning tree was gotten via a breadth-first search starting from vertex a.

    The next spanning tree was gotten via a depth-first search starting from vertex a.

    The next spanning tree is neither breadth-first, nor depth-first.

a. The first helper function Sanjay decides to define is called ‘edge-search’. The edge-search function takes a vertex and a graph, given as a list of edges. It returns the list of edges, beginning with the first edge which contains the given vertex.

For example:
(edge-search 'a '((a b) (a c) (b c))) returns '((a b) (a c) (b c))
(edge-search 'c '((a b) (a c) (b c))) returns '((a c) (b c))
(edge-search 'x '((a b) (a c) (b c))) returns '()

Write the function edge-search. (5 pts.)

b. The helper function ‘subsetOf?’ takes two lists and returns true if all of the elements in the first list are contained in the second list.

For example:
(subsetOf? '(a b c) '(x y b c a)) returns #t
(subsetOf? '(a a b c a a) '(x y b c a)) returns #t
(subsetOf? '(a b c) '(x y b c a a b c)) returns #t
(subsetOf? '() '(a b c)) returns #t
(subsetOf? '(a b x) '(a b c)) returns #f

Write the function subsetOf? (5 pts.)

c. The helper function ‘sameElements?’ takes two lists and returns true if the lists contain the same elements, ignoring repetition.

For example:
(sameElements '(a b c a a a) '(c b b b a)) returns #t
(sameElements '() '()) returns #t
(sameElements '(a b c) '(b a a a)) returns #f
(sameElements '(b a a a) '(a b c)) returns #f

Write the function sameElements? (5 pts.)