INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
Department of Computer Science & Engineering
End-Term Examination (ETE)
Subject: Compiler Design (CSN-352)
B. Tech. (CSE) III Year, Spring (2023-2024)

Time: 9.15-12:30 AM (3 hours) **[Full Marks: 100]** **[Date: April 29, 2024]**

Read the following instructions before you start.

- There are 5 Questions. All of them should be answered.
- Write your Enrollment number, Full Name and Page Number on all the extra sheets you will be using to write your answers.
- For each question, marks will be deducted for illegible handwriting.

1. (a) Consider the following grammar.

$$S \rightarrow Sa \mid b \mid \textbf{\textit{error}}\ a$$

(i) Show the DFA for recognizing viable prefixes of this grammar. Use LR(0) items, and treat ***error*** as a terminal.

(ii) Tools such as **bison** use error productions in the following way. When a parsing error is encountered (i.e., the machine cannot shift, reduce, or accept):

- First, pop and discard elements of the stack one at a time until a stack configuration is reached where the **error** terminal of an error production can be shifted on to the stack.
- Second, discard tokens of the input one at a time until one is found that can be shifted on to the stack.
- Third, resume normal execution of the parser.

Show the sequence of stack configurations of an SLR(1) parser for the grammar above on the following input: `bacadfa`. Be sure to show all shift, reduce, and discard actions (for both stack and input).

(b) Which of LL(1), SLR(1), and LR(1) can parse strings in the following grammar, and why?

$$\begin{aligned} E &\rightarrow A \mid B \\ A &\rightarrow a \mid c \\ B &\rightarrow b \mid c \end{aligned}$$

[(10+8)+2 = 20 Marks]

2. (a) Tokenize the following C code segment. List out and state the number of tokens:

```
switch(inputvalue)
{
        case 1: b = c * d; break;
        Default: b = b++; break;
}
```

(b) Derive and write a grammar with the following `First` and `Follow` sets. Your grammar

should have exactly two productions per non-terminal and no epsilon productions. The non-terminals are X, Y, Z and the terminals are a, b, c, d, e, f.

First(X)	=	$\{b, d, f\}$	Follow(X)	=	$\{\$\}$
First(Y)	=	$\{b, d\}$	Follow(Y)	=	$\{c, e\}$
First(Z)	=	$\{c, e\}$	Follow(Z)	=	$\{a\}$
Follow(d)	=	$\{c, e\}$	Follow(a)	=	$\{\$\}$
Follow(e)	=	$\{a\}$	Follow(b)	=	$\{b, d\}$
Follow(f)	=	$\{\$\}$	Follow(c)	=	$\{c, e\}$

(c) Explain how an LALR(1) parser DFA is derived from an LR(1) parser DFA. What is the advantage of an LALR(1) parser compared to an LR(1) parser?

(d) Given the following grammar:

$S \rightarrow AaAb \mid BbBa$

$A \rightarrow \epsilon$

$B \rightarrow \epsilon$

Draw the state transition diagram and Action-Goto table for CLR(1) parser. What is the number of states in the state transition table? Is this grammar in CLR(1)?

[5+5+2+(3+3+1+1) = 20 Marks]

3. (a) (i) Let synthesized attribute F.val gives the value of the binary fraction generated by F in the grammar that follows:

$$\begin{aligned} &F \rightarrow .L \\ &L \rightarrow LB \mid B \\ &B \rightarrow 0 \mid 1 \end{aligned}$$

For instance, on input .101 we have that F.val = .625. Using only synthesized attributes, give a translation scheme (Syntax Directed Translation, SDT) for the above grammar.

(ii) Show the translation of the input string .101 by decorating its parse tree. Make sure your tree is drawn clearly.

(b) Consider the SDT given below:

$$\begin{aligned} E \rightarrow E \uparrow E \ \{print(' \uparrow ');\} \\ \mid E * E \ \{print(' * ');\} \\ \mid id \ \{print(id.name);\} \end{aligned}$$

Explain whether the above SDT is S-attributed or L-attributed.

(c) Consider the SDT given below:

$$S \rightarrow BC \ \{B.val = C.val\}$$

Explain whether the above SDT is S-attributed or L-attributed.

(d) A shift-reduce parser carries out the actions specified within braces immediately after reducing with the corresponding rule of grammar.

X → a A { print "3"} | c { print "5"}

A → b B { print "2"} | a { print "4"}

B → X c { print "1"} | c { print "5"}

Draw the parse tree and write down what will be output for the translation of the input string **ababccc** using the SDT described in the above rules.

[(6+4)+3+3+(3+1) = 20 Marks]

4. (a) Given the following C program snippet:

```
i=1; s=0;
for (i=1; i<=3; i++)
   for (j=1;j<=3;j++)
      c[i][j]=c[i][j] + a[i][j] + b[i][j];
```

Generate intermediate code as 3AC (three address codes) for the above code segment using backpatching.

(b) Optimize the following 3AC by applying common sub-expression elimination.

```
1. r1 = 2
2. r2 = r1 + 2
3. r3 = r1 + r2
4. r4 = r1 + 2
5. r2 = load(r5)
6. r7 = r1 + 2
7. r1 = r4 * r3
8. r8 = r1 + 2
```

(c) Consider expression (x + y) * (y + z) + (x + y + z). Write down the three-address code (3AC) for this expression evaluation. Draw the tables for storing the 3AC following three different representations: Quadruple, Triples, and Indirect Triples.

(d) Consider the following basic block-

B10:

S1 = 4 x I

S2 = addr(A) – 4

S3 = S2[S1]

S4 = 4 x I

S5 = addr(B) – 4

S6 = S5[S4]

S7 = S3 x S6

S8 = PROD + S7

PROD = S8

S9 = I + 1

I = S9

If I <= 20 goto L10

Draw a directed acyclic graph (DAG) and identify local common sub-expressions. After eliminating the common sub-expressions, re-write the basic block.

[6+2+(2+2+2+2)+(2+2) = 20 Marks]

5. (a) Given the following piece of code:

```
1:  A = 7
2:  B = 8
3:  C = A + B
4:  D = A + C
5:  B = C - D
6:  E = A + B
7:  A = D + C
8:  F = A + C
9:  G = E + F
10: WRITE(G)
```

(i) Show which variables are live after each instruction.

(ii) How many registers are needed to perform register allocation on this code without spills (temporarily storing in memory)?

(b) Given the following piece of code:

```
x = a + b
y = x + 1
z = x
if (x > 0) {
    z = x + 1
} else {
    x = 2
}
y = z + x
```

(i) Draw the program flow graph using basic blocks for this program. Label each basic block.

(ii) Perform liveness analysis of three variables x, y and z in the basic blocks and show the liveness status table.

(iii) What is the required number of registers required for this code snippet as obtained from the liveness analysis?

(iv) Do the results of liveness analysis on this code enable any optimization opportunities? If so, describe the optimization. If not, describe an optimization that uses liveness analysis and explain why it's not applicable.

[(5+1)+(5+5+2+2) = 20 Marks]