Indian Association for the Cultivation of Science

(Deemed to be University under *de novo* Category)

Master's/Integrated Master's-PhD Program/ Integrated Bachelor's-Master's Program/PhD Course

Mid-Semester (Sem-IV) Examination-Spring 2026

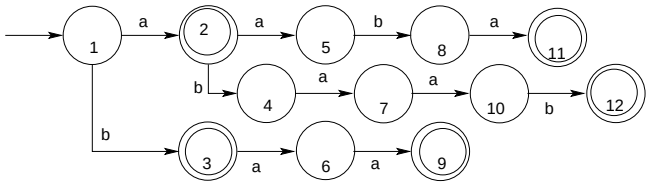
Subject: **Compiler Construction** *Subject Code:* **COM 5202**

Full Marks: **25** *Time Allotted:* **2 hours**

Q 1. Answer five (5) questions with brief justifications. [5 × 2 = 10]

(a) Draw a *deterministic state transition diagram* to generate tokens for the patterns { `a, b, baa, aaba, abaab` }, input $x \in \{\texttt{a,b}\}^*$. Show roll-backs from reachable non-final states for maximum length matching:

Ans.



Roll-backs: $4:1, 5:1, 6:1, 7:2, 8:2, 10:3$.

(b) Give an unambiguous *context-free grammar* of integer arithmetic expressions with binary operators '`+`', '`*`', and '`**`' (power, `2**3 = 8`) incorporating proper precedence and associativity of operators.

Ans. The production rules of the grammar are,

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * P \mid P \\ P &\rightarrow F * *P \mid F \\ F &\rightarrow (E) \mid ic \end{aligned}$$

Non-terminals: $\{E, T, P, F\}$, start symbol: E, terminals: $\{+, *, **, (,), ic\}$, the start symbol is E.

(c) Give a regular expression of a signed $(+/-/\ )$ binary floating point number. It must have a binary dot ('$\cdot$'), mantissa must have some digit, and the exponent is optional. Use *name definitions*.

Ans.

$$\begin{aligned} S &: (+|-) \\ D &: (0|1) \\ M &: \{S\}?((\{D\}*\backslash\cdot\{D\}+)|(\{D\}+\backslash\cdot\{D\}*)) \\ E &: \{S\}?(e|E)\{D\}+ \\ BF &: M\{E\}? \end{aligned}$$

(d) Give a *context-free grammar* for variable declarations of simple variables and multi-dimensional arrays of types `int` and `double`.

Ans.

$$\begin{aligned} DLO &\rightarrow DLO\ DL \mid DL \\ DL &\rightarrow TY\ VL \\ TY &\rightarrow \texttt{int} \mid \texttt{double} \\ VL &\rightarrow VL\ V \mid V \\ V &\rightarrow id\ IL \\ IL &\rightarrow IDX\ IL \mid \varepsilon \\ IDX &\rightarrow [ic] \end{aligned}$$

(e) The system call `int fork()` creates a child process. It returns the child process-id (`pid`) to the parent. The system call code for `fork()` is **57**. It has no other parameters. Create an equivalent `myFork()` system call by filling the (`????`) with inline x86-64 code.
`int myFork(){ int pid; __asm__volatile__ ( ???? ); return pid}`

Ans.

```
int myfork(){
    int pid;
    __asm__ __volatile__ (
        "movl $57, eax  \t\n"
        "syscall          \t\n"
     : "=a" (pid)
    );
    return pid;
}
```

(f) Following C code prints the value of `a+1` as `0x7ffc3005b1c4` (GCC x86-64). What is the value of `a` in hex?
`int main(){ int a[4][5]; printf("a+1: p", a+1); return 0; }`

Ans. Each row has 5 elements of type `int` and the stack grows towards the lower address. So the address of `a` is $5 \times 4 = 20 = 0x14$ locations below `a+1`. It is `0x7ffc3005b1c4 - 0x14 = 0x7ffc3005b1b0`.

(g) In a *flex* specification the terminals are {`if, else, while, for, int, (, ), {, }, :. ;, =, >, <. +, -, *, /, id, ic`}. What will be the effect if the rule for pattern 'dot(·)' is placed before all other rules?

Ans. Each single character terminal will be matched with the pattern '·' and corresponding appropriate tokens cannot be generated.

(h) GCC x86-64 uses the register `rdi/edi` to pass the first parameter of a function call (size 64/32-bits). Suggest a scheme to pass a larger size parameter such as a structure.

Page 2: $\rightarrow$

Answer any three (3) of the following questions.

[3 × 5 = 15]

Q 2. **[4+1]**

Consider the following C program:

```
int main(){int a=10, b, c; b=6*a; c=a+b; printf("%d\n", c); return 0;}
```

GNU x86-64 compiler translates the C program to the following *non-optimized* (sequence: 1 - 22) and *optimized* (seq: 1-7) assembly language code of x86-64.

(a) Explain the *non-optimized* code (seq: 1-22) and show its connection with the C program. Use the sequence numbers.

(b) Explain code improvements performed in both the codes.

```
pushq %rbp            # 1 movl  -8(%rbp), %eax #12
movq  %rsp, %rbp      # 2 addl  %edx, %eax     #13
subq  $16, %rsp       # 3 movl  %eax, -4(%rbp) #14
movl  $10, -12(%rbp)  # 4 movl  -4(%rbp), %eax #15
movl -12(%rbp), %edx  # 5 movl  %eax, %esi     #16
movl  %edx, %eax      # 6 leaq .LC0(%rip), %rdi#17
addl  %eax, %eax      # 7 movl  $0, %eax       #18
addl  %edx, %eax      # 8 call  printf@PLT     #19
addl  %eax, %eax      # 9 movl  $0, %eax       #20
movl  %eax, -8(%rbp) #10 leave                 #21
movl -12(%rbp), %edx #11 ret                   #22
```

Optimized Code

```
subq  $8, %rsp        # 1
movl  $70, %esi       # 2
leaq .LC0(%rip), %rdi# 3
call  printf@PLT      # 4
xorl  %eax, %eax      # 5
addq  $8, %rsp        # 6
ret                   # 7
```

Ans.

(a) **Non-optimized code**

```
 1: Save the base pointer of the caller stack.
 2: rbp <-- rsp, set base pointer of callee stack
 3: rsp <-- rsp-16, create 16 byte callee stackspace.
 4: Mem[rbp-12] <-- 10 (a = 10)
 5: edx <-- Mem[rbp-12] (a = 10)
 6: eax <-- edx (a = 10)
 7: eax <-- eax + eax (2ax = 20)
 8: eax <-- eax + edx (3a = 30)
 9: eax <-- eax + eax (6a = 60)
10: Mem[rbp-8] <-- eax (b = 6*a = 60)
11: edx <-- Mem[rbp-12] (a = 10)
12: eax <-- Mem[rbp-8] (b = 60)
13: eax <-- eax + edx (a + b = 70)
14: Mem[rbp-4] <-- eax (c = a+b = 70)
15: eax <-- Mem[rbp-4] (c = 70)
16: esi <-- eax (c = 70) (second param.)
17: rdi <-- rip + .LC0 (first param.)
18: eax <-- 0
19: call printf (through PLT)
20: destroy the stack
21: return to caller
```

Optimized code:

```
1: rsp <-- rsp-8, create 8 byte callee stackspace.
2: esi <-- 70 (second param)
3: rdi <-- rip + .LC0 (first param.)
4: call printf (through PLT)
5: eax <-- 0 (xor)
6: rsp <-- rsp+8 (reset callee stack)
7: return to caller.
```

(b) Optimizations:

- Constant propagation: `a=10` to `b=6*a` and `c=a+b`
- Constant folding at `b=6*a (6*10)`
- Constant propagation: `b=60` to `c=a+b`
- Constant folding at `c=a+b (10+60)`

Q 3. $[\mathbf{2+1+2}]$

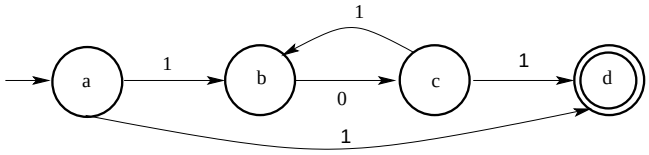
(a) Design an NFA with at least one nondeterministic transition and no more than 4 states for the language of the regular expression $\mathbf{1}(\mathbf{01})^*$.

(b) Use subset construction to convert the NFA to a DFA.

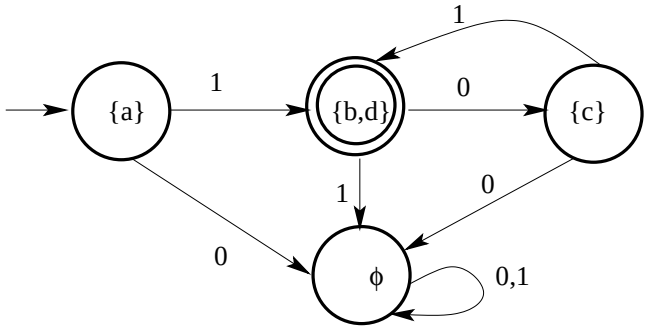
(c) Associate set of *dotted items* of the regular expression to every state of the DFA (except the dead/trap state).

Ans

(a) An NFA is



(b) Equivalent DFA is



(c) The *dotted items* associated with the states of the DFA are as follows.

$$\begin{aligned} \{a\} &: \{\bullet 1(01)^*\} \\ \{b,d\} &: \{1(\bullet 01)^*, 1(01)^*\bullet\} \\ \{c\} &: \{1(0\bullet 1)^*\} \end{aligned}$$

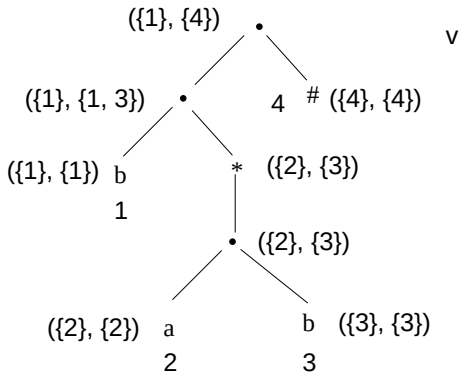
Q 4. **[3 + 2]**

(a) Augment the regular expression $\mathbf{b}(\mathbf{ab})^*$ with # and draw the *syntax tree*. Identify the *nullable nodes* of the tree, label the leaf nodes with $\{1, 2, \cdots\}$, attach *firstpos* and *lastpose* data with every node.

(b) Find *followpos* corresponding to different positions of the syntax tree and draw the NFA using the *followpos* table.

Ans.

(a) The syntax tree with the labels, *firstpos* and *lastpose*.

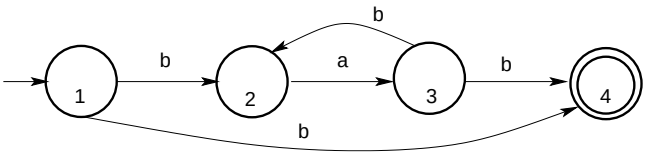


The node labeled with '$*$' is *nullable*.

(b) For different positions *followpos* are as follows:.

Position	Followpos
1	$2, 4$
2	3
3	$2, 4$

In the NFA, the *start* state is 1, the *firstpos* of root. The final state is 4, the position of #.



Q 5. **[1 + 2 + 2]**

Consider the context-free grammar G_5 with S as the *start symbol*. The *producion rules* are, $S \to A\ a \mid b$ and $A \to A\ c \mid S\ d \mid \varepsilon$. The language is $L(G_5)$.

(a) Remove ε-rule from G_5 to get a grammar G_{5a} ($L(G_5) = L(G_{5a})$.

(b) Remove *left-recursion* from G_{5a} to get an equivalent grammar G_{5b}.

(c) Is $L(G_5)$ a regular language? Justify your answer.

Ans.

(a) The grammar G_{5a} after the removal of ε-production.

$$\begin{aligned} S &\to A\ a \mid a \mid b \\ A &\to A\ c \mid S\ d \mid c \end{aligned}$$

(b) We substitute S-rules in A-rules.

$$\begin{aligned} S &\to A\ a \mid a \mid b \\ A &\to A\ c \mid A\ a\ d \mid a\ d \mid b\ d \mid c \end{aligned}$$

Now we remove direct left-recursion from A-rules.

$$\begin{aligned} S &\to A\ a \mid a \mid b \\ A &\to a\ d\ A' \mid b\ d\ A' \mid c\ A' \\ A' &\to c\ A' \mid a\ d\ A' \mid \varepsilon. \end{aligned}$$

(c) If we substitute A-rules in S-rule, we get

$$\begin{aligned} S &\to a\ d\ A'\ a \mid b\ d\ A'\ a \mid c\ A'\ a \mid a \mid b \\ A' &\to c\ A' \mid a\ d\ A' \mid \varepsilon. \end{aligned}$$

So the language is

$$(a \mid b \mid (ad\ (c \mid (ad))^*\ a) \mid (bd\ (c \mid (ad))^*\ a) \mid (c\ (c \mid (ad))^* a))$$

So it is a regular language.

$* * *$ **End** $* * *$