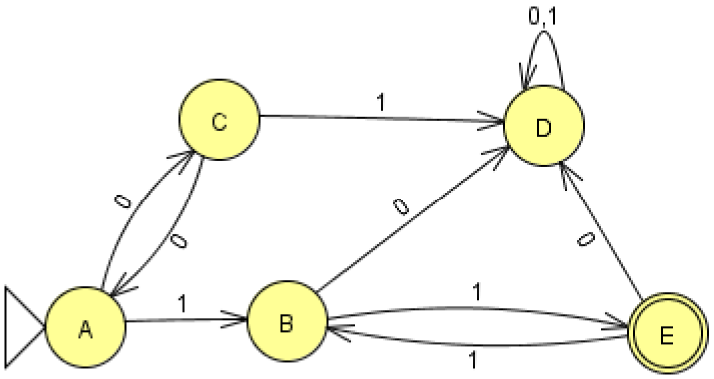
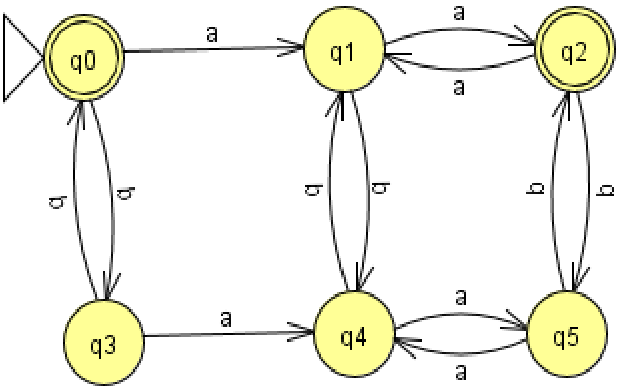
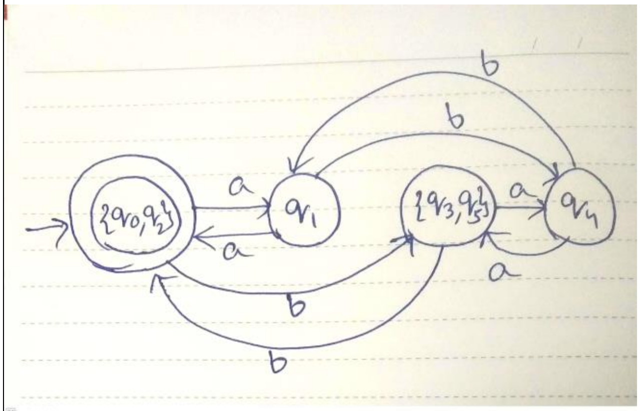
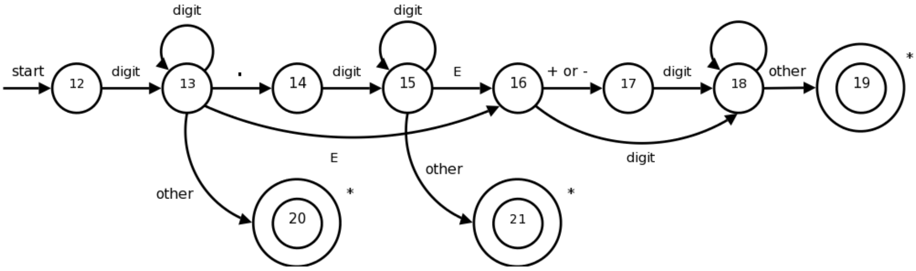
USN

Internal Assessment Test 1 – February 2024

Sub:	AUTOMATA THEORY AND COMPILER DESIGN				Sub Code:	21CS51	Branch:	CSE
Date:	01/02/2024	Duration:	90 mins	Max Marks:	50	Sem / Sec:	V/ A, C	OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1 (a)	Write regular expression for the following language: (i) All strings containing no more than 3 a's over Σ = {a,b}. **Ans:** b*+b*ab*+ b*ab*ab*+ b*ab*ab*ab* (ii) $\{w\in\{0,1\}^*$: w has 101 or 011 as a substring.} **Ans:** (0+1)*(101+011)(0+1)* (iii) $\{w\in\{a,b\}^*$: w doesn't end with bb or aa} **Ans:** (a+b)*(ab+ba) (iv) $\{ w\in\{a,b\}^*$: |w| mod 5 =0 } **Ans:** $((a+b)^5)$* (v) $\{a^nb^m$: n<=3 and m>=4} **Ans:** (Ɛ+a+aa+aaa)bbbb(b)*	[5]	CO3	L3
(b)	Convert the following FSM into a regular expression by state elimination method. Show the steps.  0,1 C 1 D 0 0 0 0 A 1 B 1 E 1 **Ans:** Step1: D is error state. Remove it Step2: Remove C and redraw the diagram Step3: Remove B and redraw the diagram R (00)*11(11)*= (00)*(11)+	[5]	CO3	L3
2 (a)	Define distinguishable and indistinguishable states. Minimize the following DFSM.  q0 a q1 a q2 a b b b b b b q3 a q4 a q5 a **Ans:** If X and Y are two states in a DFA, we can combine these two states into {X, Y} if they are not distinguishable. Two states are distinguishable, if there is at least one string S, such that one of δ (X,	[5]	CO1	L3

	S) and δ (Y, S) is accepting and another is not accepting. Hence, a DFA is minimal if and only if all the states are distinguishable. **Step 1** **(q0,q2)** **(q1,q3,q4,q5)** **Step 2** **(q0,q2)** **(q3,q4,q5)** **(q1)** **Step 3** **(q0,q2)** **(q4)** **(q1)** **(q3,q5)**  {q0,q2} a q1 a b b {q3,q5} a q4 a b b			
(b)	Design a Ɛ-NFA to accept Unsigned numbers. For example: 1234, 23.56, 23.5E12, 25.E-2. Write the regular definition for it. Ans:  start 12 digit 13 digit . 14 digit 15 digit E 16 + or - 17 digit 18 other 19 * E digit other 20 * other 21 *	[5]	CO3	L3
3 (a)	With a neat diagram explain the functions of different phases of a compiler. Show the output of each phase for the source code **Result=b*5+c/2-d**	[6]	CO2	L3

<table>
<tr>
<td></td>
<td>

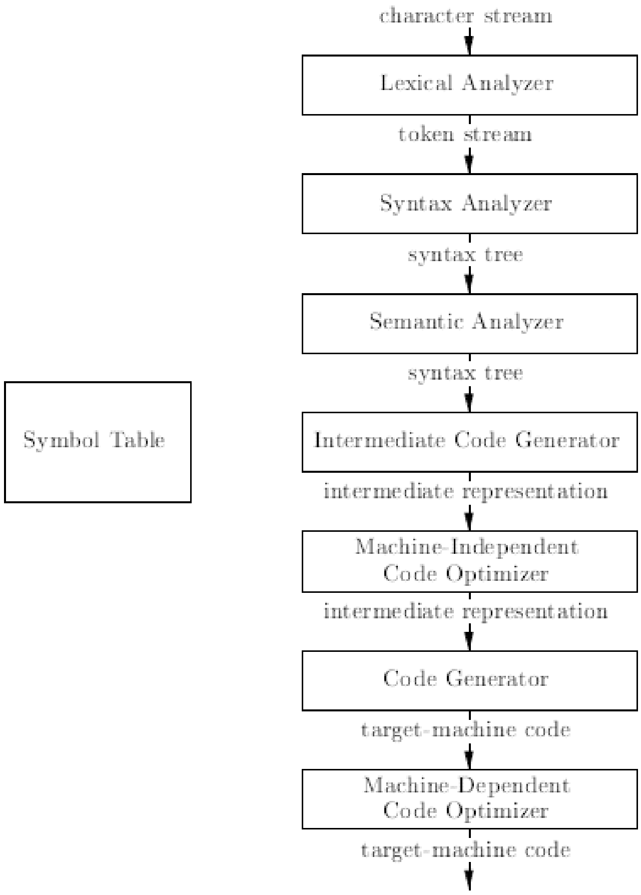


Figure 1.6: Phases of a compiler

<u>Lexical Analyzer:</u> The 1st Phase of a compiler is called lexical analysis or scanning. The lexical analyzer reads the stream of characters making up the source program and groups the characters into meaningful sequences called lexemes.

<u>Syntax Analyzer:</u> The parser uses the 1st components of the tokens produced by the lexical analyzer to create a tree-like intermediate representation that depicts the grammatical structure of the token stream.

<u>Semantic Analyzer:</u> The semantic analyzer uses the syntax tree and the information in the symbol table to check the source program for semantic consistency with the language definition.

Intermediate Code generation: We consider an intermediate form called three-address code, which consists of a sequence of assembly-like instructions with three operands per instruction.

<u>Optimizer:</u> The machine-independent code-optimization phase attempts to improve the intermediate code so that better target code will result. Usually better means faster, but other objectives may be desired, such as shorter code, or target code that consumes less power.

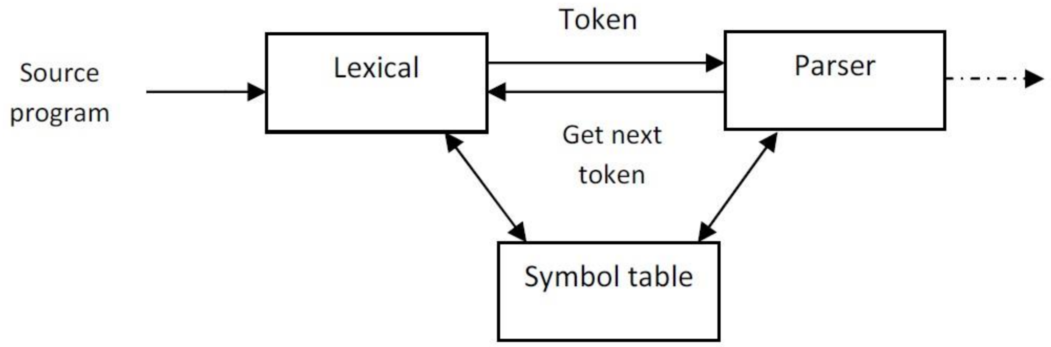
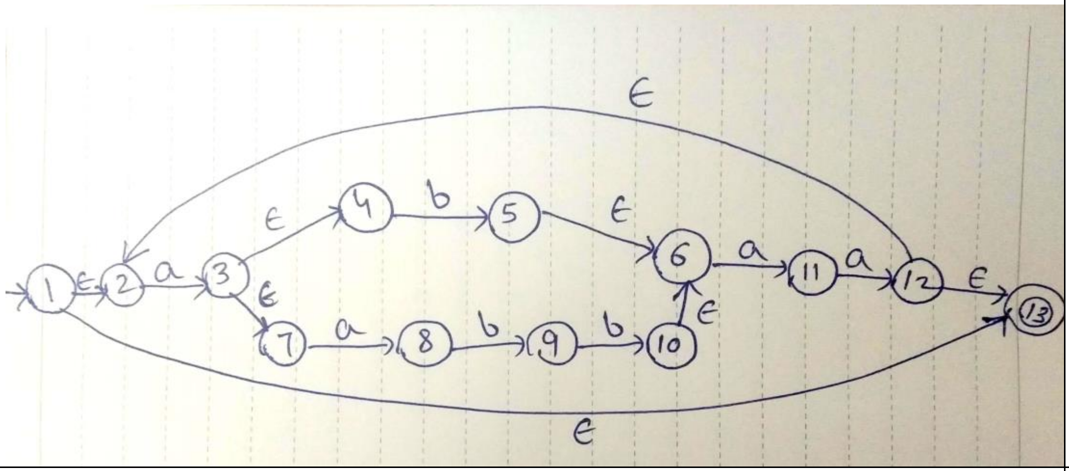
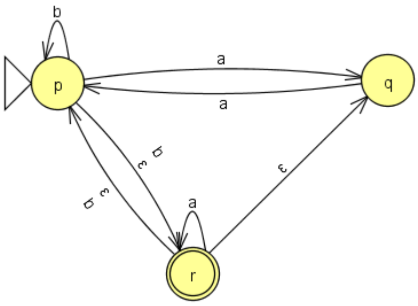
<u>Code Generator:</u> The code generator takes as input an intermediate representation of the source program and maps it into the target language. If the target language is machine code, registers or memory locations are selected for each of the variables used by the program. Then, the intermediate instructions are translated into sequences of machine instructions that perform the same task.

</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td>

What is the role of lexical analyzer? Identify lexemes and tokens in the following sequence of statements: int a=15; int b= a+10; printf (“b = %d”, b);

Ans:

</td>
<td>[4]</td>
<td>CO2</td>
<td>L2</td>
</tr>
</table>

	The lexical analyzer is responsible for breaking these syntaxes into a series of tokens, by removing whitespace in the source code.  Token Source program Lexical Parser Get next token Symbol table i) A **token** is a pair a token name and an optional token value ex: keyword, identifier.-if else and num1,num2 A **pattern** is a description of the form that the lexemes of a token may take Ex: identifier: ([a-z]|[A-Z]) ([a-z]|[A-Z]|[0-9])* A **lexeme** is a sequence of characters in the source program that matches the pattern for a token. ex: printf(“total = %d\n”, score); both **printf and score** are lexemes matching the pattern for token **id,** and **"Total = %d\n”** is a lexeme matching **literal**.			
4(a)	Construct an equivalent Ɛ-NFA using Kleene’s theorem for the regular expression: **(a (b ∪ abb) aa) *.** **Ans:**  1 2 3 4 5 6 7 8 9 10 11 12 13 a b ϵ	[5]	CO3	L3
(b)	Convert the following Ɛ-NFA to DFA using subset construction method. Find the ECLOSE of each state.  	[5]	CO1	L3

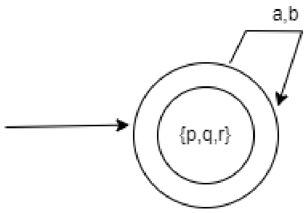
Ans:

ECLOSE(p)= {p,r,q}

ECLOSE(q)= {q}

ECLOSE(r)= {p,r,q}

State/Input	**a**	**b**
->*{p,q,r}	{p,q,r}	{p,q,r}



5 (a) Consider the following grammar. Generate the LMD, RMD, and draw the parse tree for the string w= **badbabaadb** [8] CO3 L3

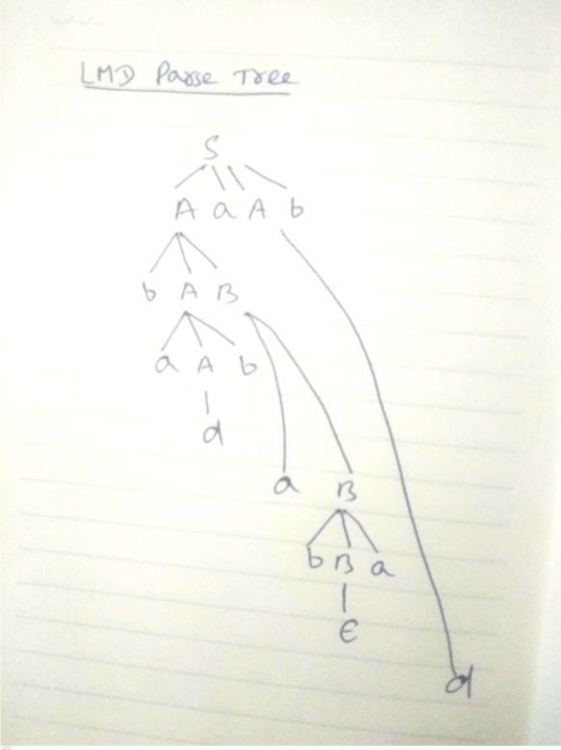
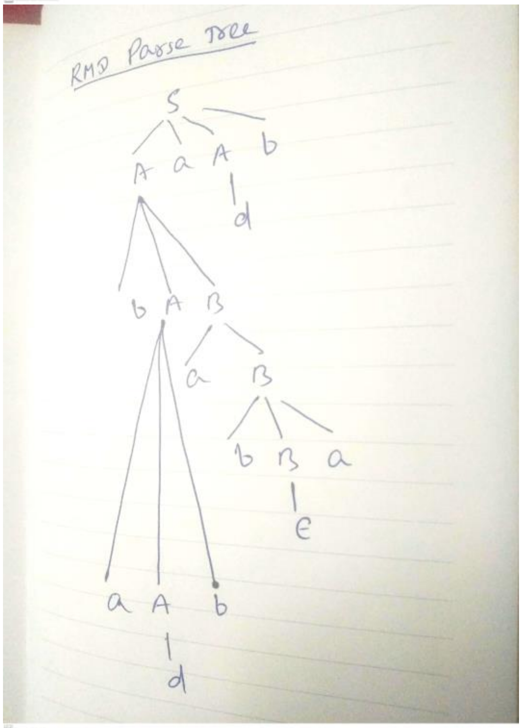
S→ AaAb | BbBa
A→ aAb | bAB | d
B → aB|bBa | Ɛ

LMD:

S=> AaAb
=>bABaAb **(**A→ bAB)
=> baAb **BaAb** **(**A→ aAb)
=> badb **BaAb** **(**A→ d)
=> badb **aBaAb** **(**B → aB)
=> badb **abBaaAb** **(**B → bBa)
=> badb **ab** Ɛ **aaAb** **(**B → Ɛ)
=> badb **abaadb** **(**A→ d)

RMD:

S=> AaAb
=>Aadb **(**A→ d)
=>**bABadb** **(**A→ bAB)
=>**bA**a**Badb** **(**B → aB)
=>**bA** abBa **adb** **(**B → bBa)
=>**bA** ab Ɛ a **adb** **(**B → Ɛ)
=>**b** aAb aba**adb** **(**A→ aAb)
=>**b** adb aba**adb** **(**A→ d)

	 			
(b)	Explain the following terms with an example. (i) Yield of a parse tree (ii) Sentential form **Yield of a parse tree :** Concatenating the leaves of a parse tree from the left produces a string of terminals. This string of terminals is called as **yield of a parse tree**. **Sentential form:** A sentential form is any string derivable from the start symbol. Thus, in the derivation of a + a * a , E + T * F and E + F * a and F + a * a are all sentential forms	[2]	CO3	L1
6 (a)	What is ambiguous grammar? Prove that the following grammar is ambiguous.	[6]	CO3	L3

	Write an equivalent unambiguous grammar for the given grammar. E→E+E |E-E |E*E|E/E |(E) |id **Ans:** A grammar is said to be ambiguous if there exists more than one leftmost derivation or more than one rightmost derivation or more than one parse tree for the given input string. If the grammar is not ambiguous, then it is called unambiguous. The input string is id+id-id **LMD1** E → E + E → id + E → id + E - E → id + id - E → id + id- id **LMD2** 1. E → E - E 2. → E + E - E 3. → id + E - E 4. → id + id - E 5. → id + id - id **Yes, it is ambiguous.** **Equivalent Unambiguous grammar:** **E-> E+T | E-T | T** **T-> T*F | T/F |F** **F-> (E) |id**			
(b)	Write CFG for the following language. (i) L={$a^ib^jc^k$ | j=i+k and i,k >=1} (ii) L={W | W is a palindrome and W ∈$\{a,b\}^*$ } **Ans:** (i) S->AB A->aAb | ab B-> bBc | bc (ii) S-> aSa | bSb | a | b | Ɛ	[4]	CO3	L3

CI CCI HOD