CBCS SCHEME

USN | | | | | | | | | |

21CS51

Fifth Semester B.E. Degree Examination, Dec.2023/Jan.2024
Automata Theory and Compiler Design

Time: 3 hrs. Max. Marks: 100

Note: Answer any FIVE full questions, choosing ONE full question from each module.

Module-1

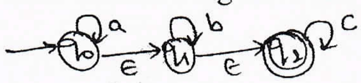
1 a. Define the following terms :
i) String ii) Language iii) Alphabet iv) Length of string **(04 Marks)**

b. Explain the various phases of compiler with neat diagram. **(08 Marks)**

c. Define DFA and design a DFA to accept the following language:
i) To accept strings having even number of a's and odd number of b's.
ii) To accept strings of a's and b's not having the substring aab. **(08 Marks)**

OR

2 a. Design the equivalent DFA to the following $\in$-NFA.



(05 Marks)

b. Minimize the following DFA by identifying distinguishable and non-distinguishable states.

	δ	0	1
→	A	B	F
	B	G	C
*	C	A	C
	D	C	G
	E	H	F
	F	C	G
	G	G	H
	H	G	C

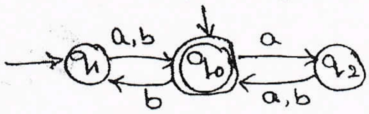
(10 Marks)

c. With neat diagram explain the components of language processing system in detail. **(05 Marks)**

Module-2

3 a. Define Regular Expressions. Write a regular expressions for the following :
i) $L = \{a^n b^m \mid n+m \text{ is even}\}$
ii) The set of all strings whose 3^{rd} symbol from right end is 0
iii) $L = \{a^{2n} b^{2m} \mid n \geq 0, m \geq 0\}$ **(10 Marks)**

b. Convert the following automata to a regular expression.



(04 Marks)

c. Explain the concept of input buffering in the Lexical Analysis along with sentinels. **(06 Marks)**

OR

4 a. State and prove Pumping Lemma for regular languages and also prove the language $L = \{a^n b^n \mid n \geq 0\}$ is not a regular. **(10 Marks)**

b. Construct ∈-NFA for the following regular expression
(0 + 11) 0*1 **(04 Marks)**

c. Define Token, Lexeme and Pattern with example. **(06 Marks)**

Module-3

5 a. Define CFG. Write a CFG to the following languages.
i) All strings over {a, b} that are even and odd Palindromes.
ii) $L = \{a^n \mid n \geq 0\}$ **(10 Marks)**

b. Define ambiguity. Consider the grammar $E \to E + E \mid E * E \mid (E) \mid id$
Construct the leftmost and rightmost derivation, parse tree for the string id + id * id.
Also show that the grammar is ambiguous. **(10 Marks)**

OR

6 a. Consider the CFG given below with the production set, compute the following for the same.
(i) First( ) and Follow( ) set (ii) Predictive Parsing table
Grammar is,
$E \to TE'$
$E' \to +TE' \mid E$
$T \to FT'$
$T' \to *FT' \mid E$
$F \to (E) \mid id$ **(14 Marks)**

b. Write an algorithm to eliminate lift recursion from a grammar. Also eliminate lift recursion from the grammar
$S \to Aa \mid b$
$A \to Ac \mid Sd \mid \in$ **(06 Marks)**

Module-4

7 a. Define PDA. Design PDA for the language $L = \{WCW^R \mid W \in (a, b)\}$ and also show the Instaneous Description (ID) for the input aabCbaa. **(10 Marks)**

b. Construct LR(0) automata for the grammar given below.
$S \to L = R \mid R$
$L \to *R \mid id$
$R \to L$ **(10 Marks)**

OR

8 a. Define shift reduce Parser and Handle. Also list and explain the different actions operations available in Bottom up parser. **(10 Marks)**

b. Construct the LR(1) automata for the given grammar.
$S \to AA$
$A \to aA \mid b$ **(10 Marks)**

Module-5

9 a. Design a Turing machine to accept the language $L = \{0^n 1^n 2^n \mid n \geq 1\}$ **(10 Marks)**

b. Write a short note on the following :
(i) Post correspondence problem (ii) Design issues in code generation **(10 Marks)**

OR

10 a. Translate the arithmetic expression $a = b * -c + b * -c$ into
(i) Three address code (ii) Quadruple (iii) Triple **(10 Marks)**

b. Write a short note on :
(i) Decidable language (ii) Halting problems in Turing machines. **(10 Marks)**

* * * * *

KLS, VDIT, HALIYAL.

DEPT. OF. CSE.

QP SOLUTION DEC-2023-24.

Sub :- AUTOMATA THEORY & COMPILER DESIGN [21CS51].

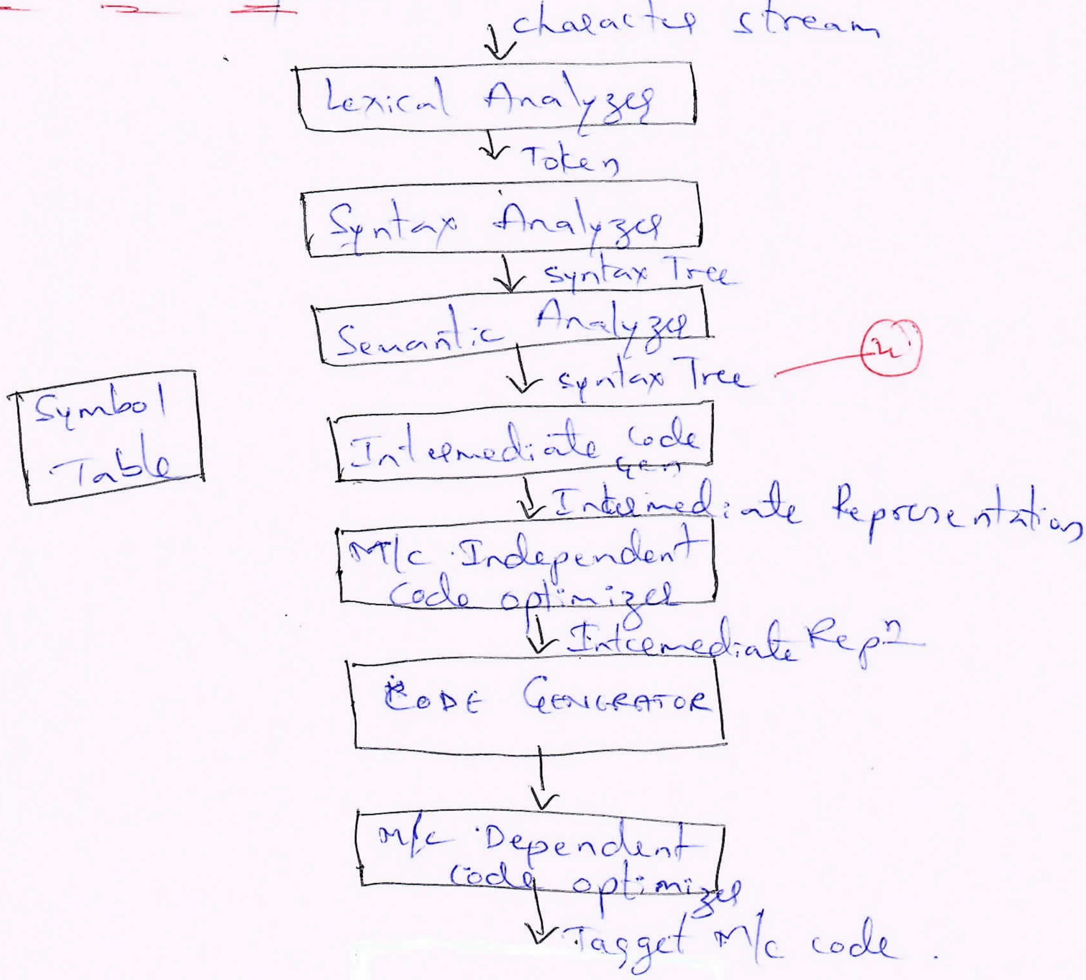
1. a) String :- It is a finite sequence of symbol. choosen from some alphabet. [4M]

Language :- Set of strings of all of which are choosen.

Alphabet :- Its finite non-empty set of symbols.

Length of Strings :- The no. of characters in a given string x & its is denoted by |x|.

b) Phases of Compiler. [8M]



* There are 2 parts of mapping :-
→ ANALYSIS , → SYNTHESIS.

* The different phases of Compilers are :-

→ LEXICAL ANALYSIS :- It reads the stream of charac making up source program & groups the characters into meaningful sequences called lexemes.

→ SYNTAX ANALYSIS :- Parser uses the first component of tokens produced by lexical analyzer to create a tree like intermediate representation that depicts the grammatical structure of token stream.

→ SEMANTIC ANALYSIS :- It uses the syntax tree & information in the symbol table to check source program for semantic consistency with the definiti

→ An important part of semantic analysis is type checking, where compiler checks that each operator has matching operands.

→ Intermediate CODE GENERATION :- In a process of translating a source program into target code compiler may construct one or more intermediate representations which have variety of forms.

* Syntax tree are a form of intermediate representation.

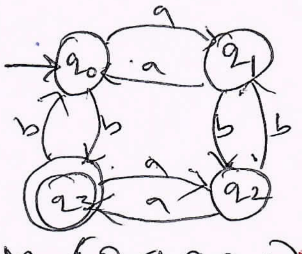
→ CODE OPTIMIZATION :-
There are simple optization which improve running time of target program.

→ CODE GENERATION :- There are an i/p to intermediate rep^n. of src prog & maps to target language.

c) **DFA** – M(Q, Σ, δ, q_0, F). Q - set of states, Σ - alphabet, δ - Transition function, q_0 - initial state & F - Final state. (2)

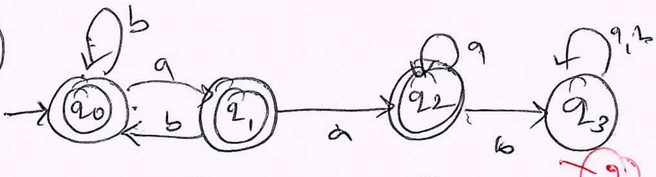
[2+6M]

i)



M = (Q, Σ, δ, q_0, F) — (2) (1)

ii)

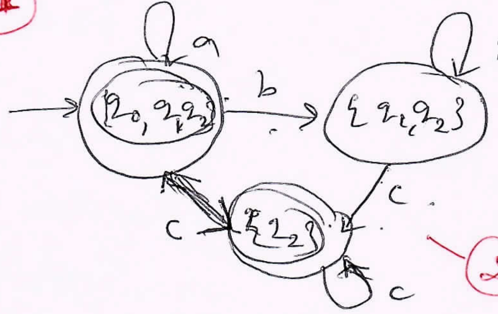


M = (Q, Σ, δ, q_0, F) — (2) (1)

2. a) ε-closure(q_0) = {q_0, q_1, q_2}. — (1)

ε-closure(q_1) = {q_1, q_2}

ε-closure(q_2) = {q_2}.



— (2)

[5M]

	a	b	c
→ {q_0, q_1, q_2}	q_0, q_1, q_2	q_1, q_2	q_2
* q_1 q_2	∅	q_1, q_2	q_2
* q_2.	∅	∅.	q_2

— (2)

b) **Minimization of DFA :-**

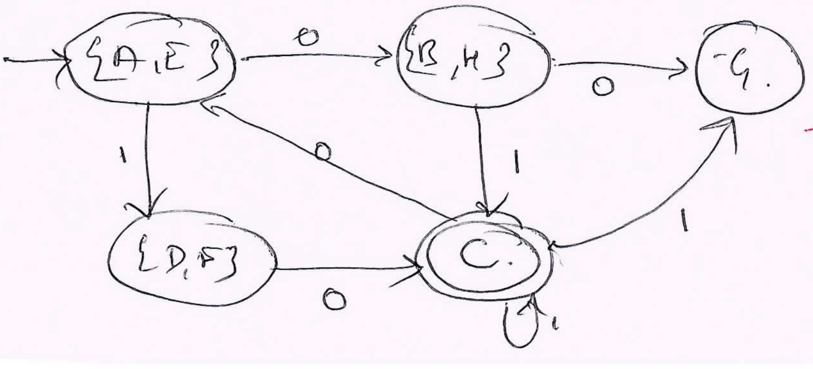
	A	B	C	D	E	F	G
B	✓						
C	✓	✓					
D	✓	✓	✓				
E		✓	✓	✓			
F	✓	✓	✓		✓		
G	✓	✓	✓	✓	✓	✓	
H	✓		✓	✓	✓	✓	✓

(5)

δ	a	b.
→ {A, E}	{B, H}	{D, F}
{B, A}	{G}	{C}
{D, F}	{C}	{G}
{G}	{G}	{A, E}
* {C}	{A, E}	{C}

[10M]

DFA :-

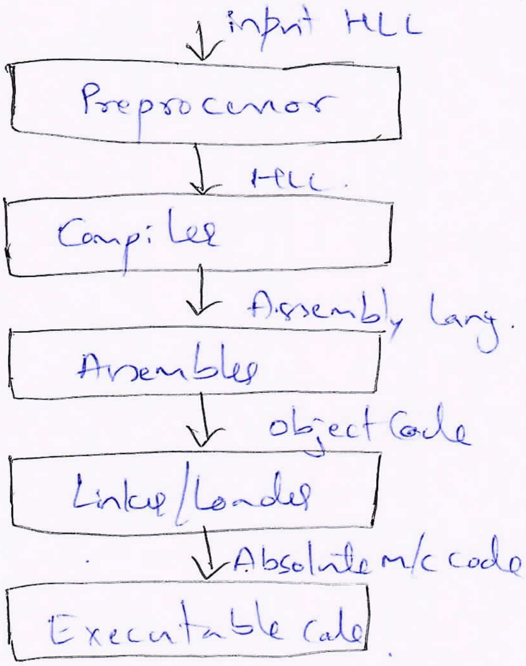


— (5).

Write $Q, \Sigma, \delta, q_0, F$.

of $M = (Q, \Sigma, \delta, q_0, F)$.

c) Language Processing System :- [5M]



(3)

(2)

* preprocessor handles all header files & evaluates a macro.
* Compiler which takes the src prog & produces a target code, Linker is a system s/w which links to the library functions.
* Loader keeps the linked program in main memory.

3. a) Regular Expression :- It is notation to specify regular Language. (1)

(i) $L = \{a^n b^m \mid n+m \text{ is even}\}$ [10M]

$$RE = (aa)^* bb^* + a(aa)^* b(bb)^*$$ (3)

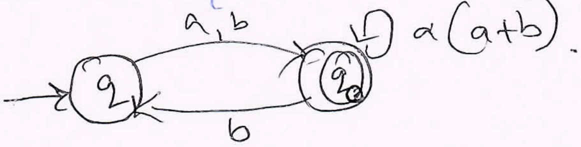
(ii) The set of all strings whose 3 symbol from right end is 0.

$$RE = (0+1)^* 0 (0+1) \cdot (0+1)$$ (3)

iii) $L = \{a^{2n} b^{2m} \mid n \geq 0 ; m \geq 0\}$.

$RE = (aa)^* (bb)^*$. — (3)

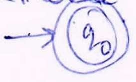
b) Conversion of DFA to RE by state Elimination method :-

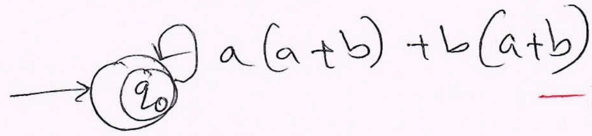


— 2 [4M]

Eliminate q_2.

Eliminate q_1

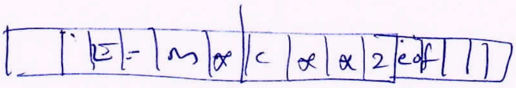




—2

$\therefore RE = a(a+b) + b(a+b)$.

c). I/p Buffering :-



The i/p buffering of lexical analyser is implemented with buffer pairs & sentinels.

Each buffer is of same size n & N is usually size of disk block. The 2 pointers.

⇒ pointer lexemeBegin marks the beginning of current lexeme

→ forward scans ahead until a pattern match is found.

Algorithm to lookahead code with sentinels :-

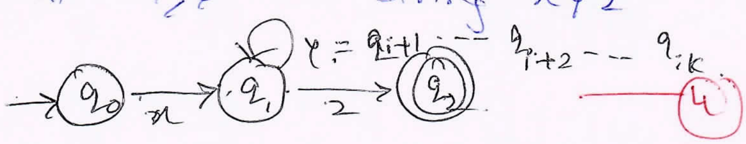
```
switch (*forward++)
{
  case eof :
      if (forward is at end of first buffer)
      { reload sec buffer;
        forward = beginning of sec buffer.
      }
    else if (forward is at end of sec buffer)
    {
      reload first buffer.
      forward = beginning of first buffer;
    }
    else { terminate lexical analysis; }
  break;
      cases for other characters.
}
```

4-a) <u>Pumping Lemma</u>:-

Let L be regular language, it can be generated by DFA. Then there exist a constant n (no. of states) such that for every string w in L |w| ≥ n, we can break w into 3 strings. w = xyz such that. → (2) [10M]

1. $y \neq \epsilon$ or $|y| > 0$.

2. $|xy| \leq n$ 3. For all $k \geq 0$ the string xy^kz is also L



(4)

$L = \{a^n b^n \mid n \geq 0\}$. (4)

Assume L is regular. Let n be a constant.

$w = a^n b^n$ split $w = xyz$ such that $|y| > 0$ $|xy| \leq n$

for all $k \geq 0$ string xy^kz be in L

Assume n = 2

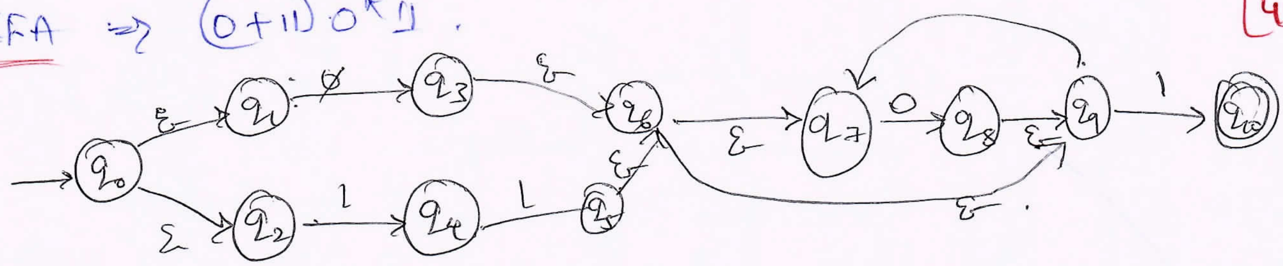
w = aabb

xy = aa then y = a & x = a & z = bb. for k = 2

⟹ xy^kz ⟹ aa^2bb ⟹ aaabb.

∴ no. of a's not equal to b's. ∴ L is not regular.

b) <u>ε-NFA</u> ⟹ (0+11)0*1. [4M]



c). <u>Token</u> is a pair consisting of token name & an optional attribute value. Σ eg:- if, id, ele. [6M]

Pattern:- is a discription of the form that lexeme of a token may take

eg:- if → i & f   else → e, l, s, e

lexeme :- is a sequence of char in the src prog that matches the pattern for token.

eg:- if matched keyword.

5, a) CFG :- $G = (V, T, P, S)$ where V is set of variables, P - set of productions, T - set of terminals, S - set of symbols. — (2)

CFG → i) Even palindrome

$P = S \rightarrow aSa \mid bSb \mid \epsilon$

$CFG = \{ \{S\}, \{a, b\}, P, S \}$ — (6)

ii) Odd palindrome

$P = S \rightarrow aSa \mid bSb \mid a \mid b$

$CFG = \{ \{S\}, \{a, b\}, P, S \}$

[10M]

iii) $L = \{ a^n ; n \geq 0 \}$. — (2)

$P = S \rightarrow aS \mid \epsilon$ $\quad$ $CFG = \{ \{S\}, \{a\}, P, S \}$.

b) Grammar is ambiguous if there exists 2 or more parse tree for the given string. — (2)

$E \rightarrow E+E \mid E*E \mid (E) \mid id$.

LMD :- E

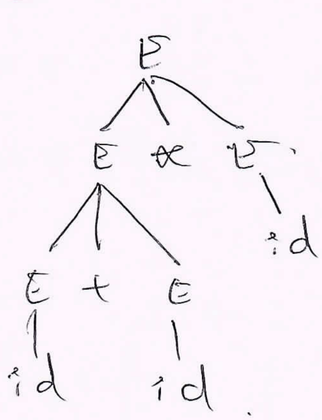
$\Rightarrow E*E$ — (2)

$\Rightarrow E+E*E$

$\Rightarrow id+E*E$

$\Rightarrow id+id*E$

$\Rightarrow id+id*id$.



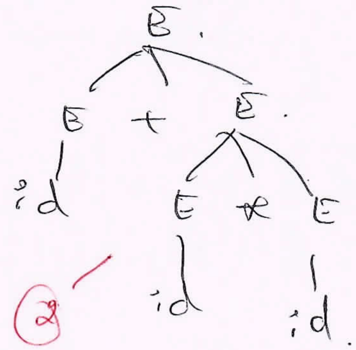
— (2)

RMD :- E — (2)

$\Rightarrow E+E \Rightarrow E+E*E$

$\Rightarrow E+E*id \Rightarrow E+id*id$ [10M]

$\Rightarrow id+id*id$



(2)

6.a) E → TE'

E' → +TE' / ε

T → FT'

T' → *FT' / ε

F → (E) id.

	FIRST	FOLLOW
E	{id, (}	{$,)}
E'	+, ε.	{$,)}
T	{id, (}	{+, $,)}
T'	{*, ε}	{+, $,),}
F	{id, (}	{+, *, $,)}

⑧ [14M]

Parsing Table.

⑥

	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ε	E'→ε
T	T→FT'			T→FT'		
T'		T'→ε	T'→*FT'.		T'→ε	T'→ε
F	F→id.			F→(E)		

b). Left Recursion Algorithm :-

Arrange non-terminal in some order $A_1, A_2, A_3 \ldots A_n$.

for (each i from 1 to n)

{ for (each j from 1 to i-1) ③ [6M]

{ replace each production of form $A_i \to A_j \gamma$

by the . $A_i \to \delta_1 \gamma / \delta_2 \gamma / \ldots \delta_n \gamma$ where

$A_j \to \delta_1 / \delta_2 / \ldots / \delta_k$ are all current A_j prod

}

eliminate immediate left recursion among A_i prod.

}.

* S → Aa / b

A → Ac / Sd / ε.

LR ⇒

S → Aa / b

A → bdA' / A' ③

A' → cA' / adA' / ε.

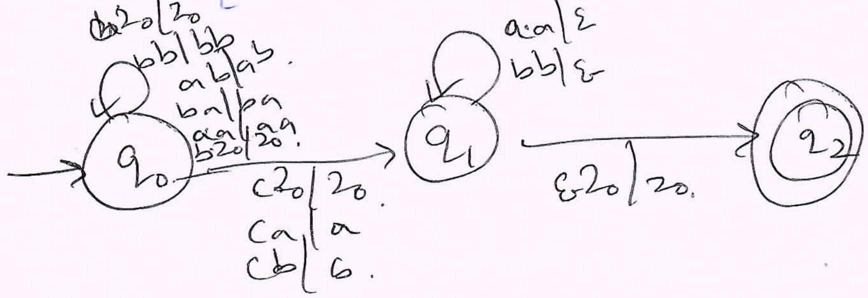
7. a) **PDA** is defined as $M = \{Q, \Sigma, \Gamma, Z_0, q_0, \delta, F\}$

Q is set of states. Σ - set of i/p alphabets [10M]

Γ - stack alphabets $Z_0 \to$ content of stack

q_0 - start state, F - ~~final~~ state δ - transition Fun — (2)

PDA for $L = \{wCw^R ; w \in (a, b)\}$



— (6)

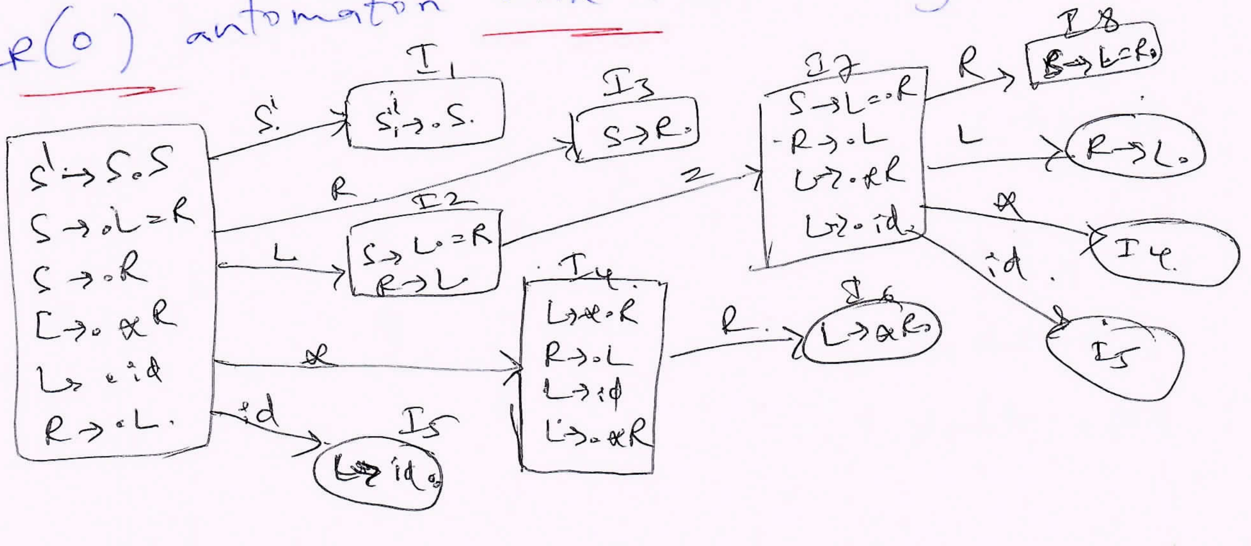
$(q_0, aabCbaa, Z_0) \vdash (q_0, abCbaa, aZ_0)$

$\vdash (q_0, bCbaa, aaZ_0) \vdash (q_0, Cbaa, baaZ_0)$

$\vdash (q_1, baa, baaZ_0) \vdash (q_1, aa, aaZ_0) \vdash (q_1, a, aZ_0)$

$\vdash (q_1, \varepsilon, Z_0) \vdash (q_2, \varepsilon, Z_0)$. — (2)

b) **LR(0)** automaton with the following sets. [10M]



8.a) Shift Reduce parser :- is a bottom up parsing method, where parse tree is constructed from bottom root. — (2)

[10M]

Handle :- is a substring of right hand side in a production rule in CFG. — (2)

Different actions in shift reduce parser are :-

* Shift :- which shift next i/p symbol into top of stack. — (6)
* Reduce : The right end of the string to be reduced must be at top of stack.
* Accept :- Announces ~~of~~ successful completion of parsing.
* Error :- Discovers a syntax ~~error~~ & call an error recovery routine.

b) LR(1) automata for the given grammar :- — (4)

S → AA

A → aA | b.

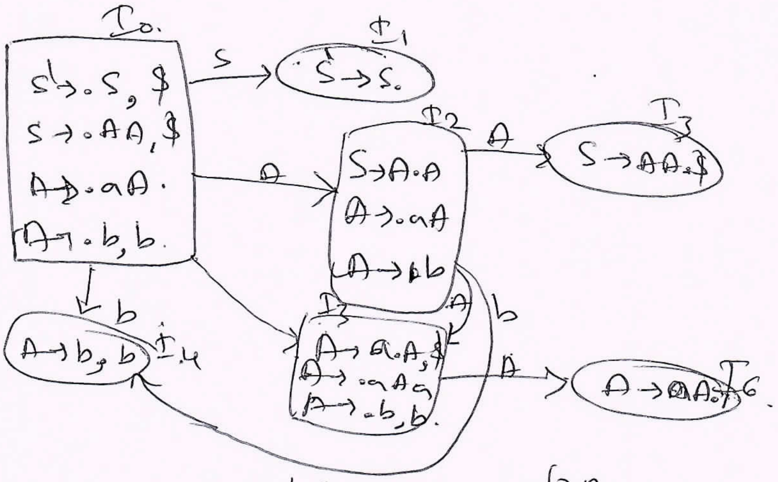
S' → .S, $.

S → .AA, $ — (2)

A → .aA | .b, a/b.

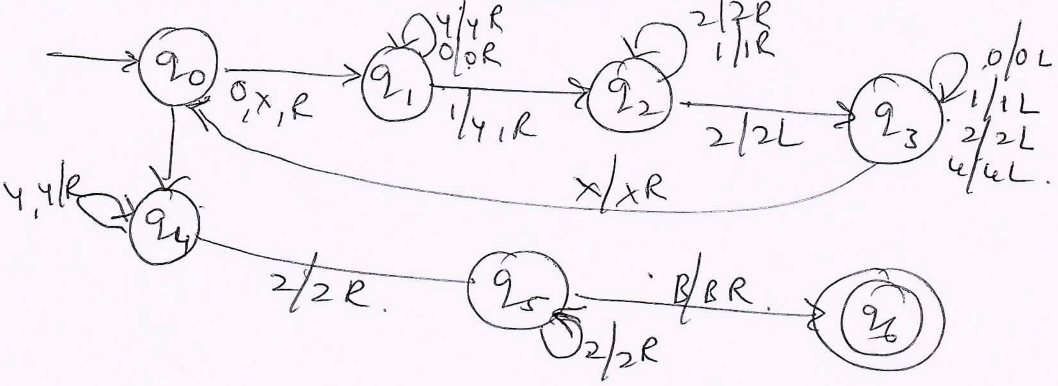
[10M]

state	Action a	Action b	Action $	Goto S	Goto A
I_0	S_2	S_3		I_1	I_2
I_1			Accept		
I_2	S_2	r_3			I_5
I_3	,				I_6
I_4		r_3			
I_5			r_1		
I_6	r_2				



(4)

9. a)



[10m

b) (i) Post correspondence problem :-

* Introduced by Emil Post in 1946, is undecidable decision problem
* PCP problem over an alphabet Σ is state as follows :- given the following 2 list M & N of non-empty string over Σ, $M = (x_1, x_2, --- x_n)$. $N = (y_1, y_2, --- y_n)$

We can say that there is position for some $i_1, i_2 - i_n$ where $1 \leq i \leq n$ the condition. $x_1, x_2 -- x_{i_k} = y_1, y_2 --- y_{i_k}$ satisfies.

[10m]

ii) Issues in code generation are :-

* I/p to code generator :-

There are many choices for IR include 3 address representation such as. Quadruples, triples & Indirect Triples.

*TARGET PROGRAM :-

The most common target m/c architecture are RISC & CISC, stack based architecture.

→ RISC m/c typically has many registers; 3-address instruction, simple addressing modes & simple instruction set architecture. [8m]

→ CISC has few registers, 2 address instruction, variety of addressing modes, several register classes, variable length instruction.

→ Stack based machine operations are done by pushing operations on operands at top of stack.

*INSTRUCTION SELECTION :-

There are 4 factors of mapping IR into target m/c

→ level of IR.

→ Nature of Instn set Architecture

→ Desired Quality of generated code.

*If IR is high level, translate each IR stmt into sequence of m/c instn.

→ The nature of instn set of target m/c has a set of strong effect on difficulty of instn selection.

eg:- x=y+z → LD R0, Y.
ADD R0, R0, Z.
ST x, R0

→ Quality of generated code is determined by its speed & size.

*REGISTER ALLOCATION :-

Registers are fastest computational unit on target m/c.

→ The use of registers is divided into 2 subproblems :

i) Register Allocation :- select set of variables which resides in register at each point of program.

10.a) Translate $a = b * -c + b * -c$

[10M]

i) Three address code

t_1 = uminus c

$t_2 = b * t_1$

t_3 = uminus c

$t_4 = b * t_3$

$t_5 = t_2 + t_4$

$a = t_5$

3

ii) Quadruple

4

	Op	Arg1	Arg2	res
(1)	+	x	y	t_1
(2)	+	y	2	t_2
(3)	*	t_1	t_2	t_3
(4)	+	t_1	2	t_4
(5)	+	t_3	t_4	t_5

3

iii) Triples ⇒

	op	arg1	arg2
(1)	+	x	y
(2)	+	y	2
(3)	*	(1)	(2)
(4)	+	(1)	2
(5)	+	(3)	(4)

b) Decidable Language :-

A language is decidable or recursive if its accepted by Turing machine & halts on every input string w.

5

* Every decidable language is turing machine acceptable.

* A decision problem P is decidable if the language L if all instances to P in decidable.

b) Halting problems in Turing machines :-

⇒ given a turing machine M & an i/p string x does M halt on input x?

* If a machine doesnot stop then we don't know for sure it its going to accept on the given input or not.

* * * * *.

[signature]
Prof. F. N. Nadaf

[signature]
1/8/24