-

USN | | | | | | | | | | |

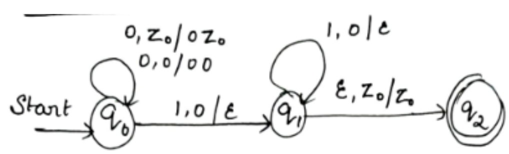
Internal Assessment Test 3 Answer Key – March 2024

Sub:	**AUTOMATA THEORY AND COMPILER DESIGN**			Sub Code:	**21CS51**	Branch:	**AIDS**
Date:	**16/03/2024**	Duration: **90 Mins**	Max Marks: **50**	Sem	**V**		**OBE**

		Answer any FIVE Questions	MARKS	CO	RBT
1	a	(see below)	[6]	CO3	L3

Construct PDA for

a. L={0^n1^n | n>=0}



$$\delta(q_0, 0, Z_0) = (q_0, 0Z_0)$$

$$\delta(q_0, 0, 0) = (q_0, 00)$$

(input, top; push, previous)

$$\delta(q_0, 1, 0) = (q_1, \varepsilon)$$

$$\delta(q_1, 1, 0) = (q_1, \varepsilon)$$

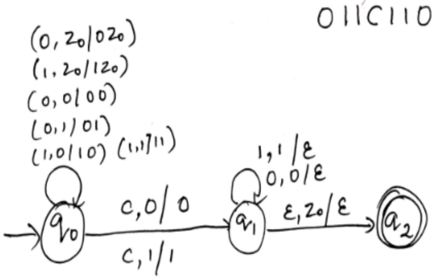
$$\delta(q_1, \varepsilon, Z_0) = (q_2, Z_0)$$

P=({q0,q1,q2}, {0,1},{0,z0},delta, q0,z0,q2} **–3marks**

b. L={wcw^R | w is in $(0+1)^+$}

$L = \{wcw^R \mid w \text{ is in } (0+1)^+\}$ determini

011C110



–3 marks

	b	**List the issues in the design of a code generator.** 1. Input to the code generator 2. Target program 3. Memory management 4. Instruction selection 5. Register allocation 6. Evaluation order	[4]	CO2	L1
2	a	Parse the string using Shift Reduce Parsing and write the viable prefixes. E->E+T E->T T->T*F T->F F->(E) F->id w= id*id+id	[6]	CO4	L3

Stack	Input	Action
$	id*id+id$	shift
$id	*id+id$	Reduce by F->id
$F	*id+id$	Reduce by T->F
$T	*id+id$	shift
$T*	id+id$	shift
$T*id	+id$	Reduce by F->id
$T*F	+id$	Reduce by T->T*F
$T	+id$	shift
$T+	id$	shift
$T+id	$	Reduce

		by F->id
$T+F	$	Reduce by T->F
$T+T	$	Reduce by E->T
$E+T	$	Reduce by E->E+T
$E	$	Accept

—-----------5 marks

Viable prefixes: id, F, T, T*, T*id, T*F, T, T+, T+id, T+F, T+T, E+T, E

—----1Mark

	b	Explain Turing Machine and the Halting Problem. (see below)	[4]	CO5	L2

Explain Turing Machine and the Halting Problem.

We may visualize a Turing machine as in Fig. 8.8. The machine consists of a *finite control*, which can be in any of a finite set of states. There is a *tape* divided into squares or *cells*; each cell can hold any one of a finite number of symbols.

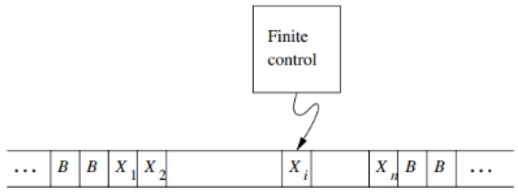


Figure 8.8: A Turing machine

A single tape Turing machine has a single infinite tape, which is divided into cells.

The tape symbols are present in these cells.

A finite control is present, which controls the working of Turing machines based on the given input.

The Finite control has a Read/write head, which points to a cell in tape.

A Turing machine can move both left and right from one cell to another

that used for finite automata or PDA's. We describe a TM by the 7 tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

whose components have the following meanings:

Q: The finite set of *states* of the finite control.

Σ: The finite set of *input symbols*.

Γ: The complete set of *tape symbols*; Σ is always a subset of Γ.

δ: The *transition function*. The arguments of $\delta(q, X)$ are a state q and a tape symbol X. The value of $\delta(q, X)$, if it is defined, is a triple (p, Y, D), where:

1. p is the next state, in Q.
2. Y is the symbol, in Γ, written in the cell being scanned, replacing whatever symbol was there.
3. D is a *direction*, either L or R, standing for "left" or "right," respectively, and telling us the direction in which the head moves.

q_0: The *start state*, a member of Q, in which the finite control is found initially.

B: The *blank* symbol. This symbol is in Γ but not in Σ; i.e., it is not an input symbol. The blank appears initially in all but the finite number of initial cells that hold input symbols.

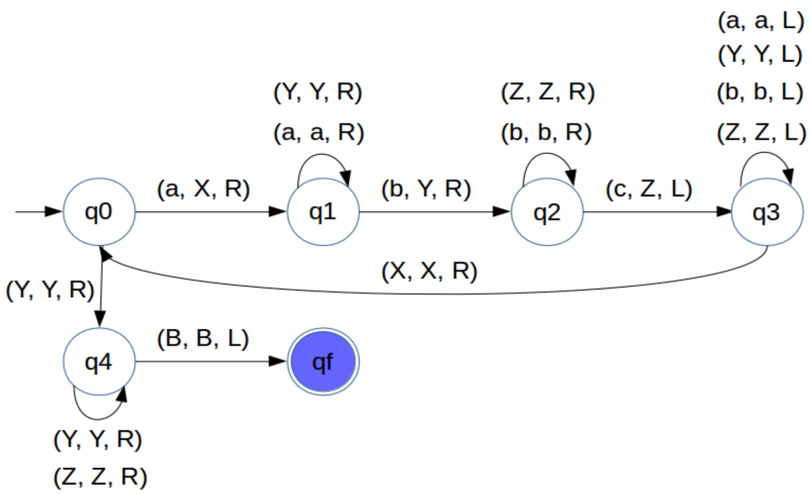
F: The set of *final* or *accepting* states, a subset of Q.

There is another notion of "acceptance" that is commonly used for Turing machines: acceptance by halting. We say a TM *halts* if it enters a state q, scanning a tape symbol X, and there is no move in this situation; i.e., $\delta(q, X)$ is undefined.

Turing machine-2 marks

Halting problem-2 marks

3	a	Construct a Turing Machine to accept L= $\{a^n b^n c^n \mid n>=1\}$. Draw the state diagram, transition table and definition.	[6]	CO5	L4



—-4 Marks

states	a	b	c	X	Y	Z	B
q0	(q1,X,R)				(q4,Y,R)		
q1	(q1,a,R) ~~(q1,q,R)~~	(q2,Y,R)			(q1,Y,R)		
q2		(q2,b,R)	(q3,Z,R)			(q2,Z,R)	
q3	(q3,a,L)	(q3,b,L)		(q0,X,R)	(q3,Y,L)		(a,q,L

M=({q0,q1,q2,q3,q4,qf},{a,b,c},{a,b,c,B},delta, q0, B, qf}

—->2 marks

	b	Prove that every language accepted by a multitape TM is acceptable by some standard TM.	[4]	CO5	L5

Theorem: Every language accepted by a multitape TM is recursively enumerable.

(or)

every Multitape TM has an equivalent Single Tape Turing Machine.

(or)

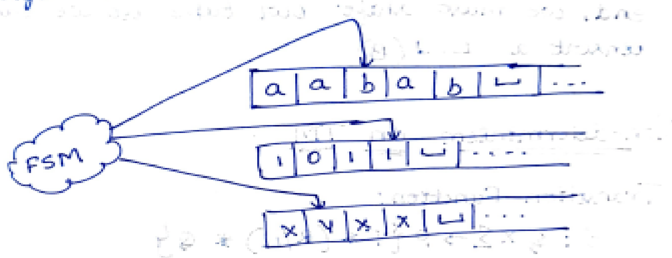
every language accepted by a Multitape TM is acceptable by some standard TM.

Proof: Given a Multitape TM, show how to build a single tape TM.

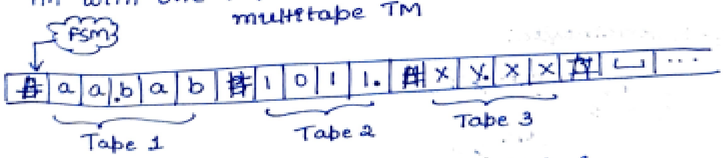
1. Need to store all tapes on a single tape

Show data representation

- Each tape has a tape head
  Show how to store that info
- Need to transform a move in the multitape TM into one or moves in the single tape TM.



TM with one tape to simulate the above multitape TM

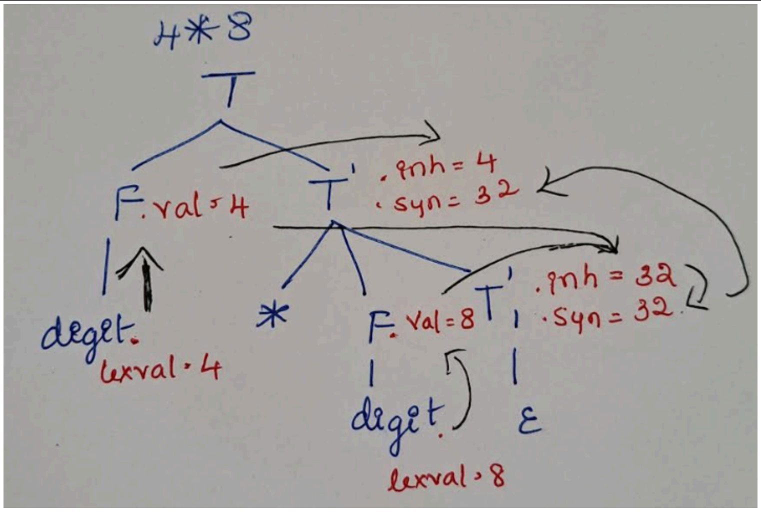


- Add dots to show where head "K" is
- To simulate a transition from state Q, we must scan our Tape to see which symbols are under the K tape Heads.

Theorem 8.9: Every language accepted by a multitape TM is recursively enumerable.

PROOF: The proof is suggested by Fig. 8.17. Suppose language L is accepted by a k-tape TM M. We simulate M with a one-tape TM N whose tape we think of as having $2k$ tracks. Half these tracks hold the tapes of M, and the other half of the tracks each hold only a single marker that indicates where the head for the corresponding tape of M is currently located. Figure 8.17 assumes $k = 2$. The second and fourth tracks hold the contents of the first and second tapes of M, track 1 holds the position of the head of tape 1, and track 3 holds the position of the second tape head.

<table>
<tr>
<td>4</td>
<td>a</td>
<td>
Give the Difference between L-attributed and S-attributed.
For the given SDD, give annotated parse tree for the following expression with dependency graph for L-Attributed
<table>
<tr><th>Productions</th><th>Semantic Rules</th></tr>
<tr><td>T->FT’</td><td>T’.inh=F.val
T.val=T’.syn</td></tr>
<tr><td>T’->*FT1’</td><td>T’.inh=F.val*T’.inh
T’.syn=T’1.syn</td></tr>
<tr><td>T’->ε</td><td>T’.syn=T’.inh</td></tr>
<tr><td>F->digit</td><td>F.val=digit.lexval</td></tr>
</table>
Expression: 4*8
<table>
<tr><th>S-Attributed SDT</th><th>L-Attributed SDT</th></tr>
<tr><td>1) uses only ~~signif~~ synthesized attribute
A->BC</td><td>1) uses both synthesized & inherited attribute. Inherited attribute can inherit values from either parent or left siblings only.
A->BCD {B.V=A.V, C.V=B.V, D.V=B.V}
(C.V=D.V) ×.</td></tr>
<tr><td>2) Semantic actions are placed at right end of production
A->BC {...}</td><td>3) Semantic actions are placed anywhere on RHS.
A -> {...} BC
| B {...} C
| BC {...}</td></tr>
<tr><td>3) Attributes are evaluated during Bottom up Parsing.</td><td>3) By traversing parse tree depth first, left to right.</td></tr>
</table>
—4 Marks
</td>
<td>[8]</td>
<td>CO4</td>
<td>L1, L3</td>
</tr>
</table>

		 —--4 marks			
	b	List the programming techniques used for TM. 1. Storage in a state 2. Multiple Tracks 3. Subroutines	[2]	CO5	L1

5		Parse the grammar and the string using CLR Parsing S->CC C->cC|d w= cdcd	[10]	CO4	L3

1) S→CC
C→cC
C→d.

Augmented Grammar
S'→S
S→CC
C→cC
C→d

LR(1) items (closure)

I_0 : S'→.S , $ → lookahead symbol
S→.CC, $
C→.cC, c/d
C→.d, c/d

goto (I_0, c)
I_3: C→c.C, c/d
C→.cC, c/d
C→.d, c/d

goto (I_0, d)
I_4 → C→d., c/d

goto (I_0, S)
I_1: S'→S., $

goto (I_0, C)
I_2: S→C.C, $
C→.cC, $
C→.d, $

goto (I_2, C)
I_5 → S→CC., $

goto (I_2, c)
I_6: C→c.C, $
C→.cC, $
C→.d, $

goto (I_2, d)
I_7: C→d., $

goto (I_3, C)
I_8: C→cC., c/d

goto (I_6, C)
I_9: C→cC., $

State	Action			goto	
	c	d	$	S	C
0	S3	S4		1	2
1			accept		
2	S6	S7			5
3	S3	S4			8
4	r3	r3			
5			r1		
6	S6	S7			9
7			r3		
8	r2	r2			
9			r2		

Stack	Input	Action
0	cdcd$	shift
0c3	dcd$	shift
0c3d4	cd$	reduce C→d.
0c3C8	cd$	reduce C→cC
0C2	cd$	shift
0C2c6	d$	shift
0C2c6d7	$	reduce C→d.
0C2c6C9	$	reduce C→cC
0C2	$	reject

6		For the given input find a. DAG/Syntax tree b. Value number Method c. Quadruple d. Triple e. Indirect triple	[10]	CO4	L3

Expression: a+b+(a+b)

1	id	a	
2	id	b	
3	+	1	2
4	+	3	3

Dag and Number value—------4 marks
Quadruple—--2 marks
Triple—--2 marks
Indirect triple —-2 marks

t1=a+b
t2=t1+t1

Quadruple

	op	arg1	arg2	result
0	+	a	b	t1
1	+	t1	t1	t2

Triples

	op	arg1	arg2
0	+	a	b
1	+	0	0

Indirect Triples

(0) 11
(1)12

	op	arg1	arg2
11	+	a	b
12	+	(11)	(11)

CCI SIGNATURE

HOD SIGNATURE