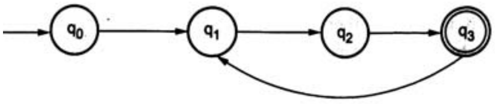
S.E. Sem. IV [INFT]

Automata Theory

Time: 3 Hrs.] Prelim Paper Solution **[Marks : 80**

Q.1(a) Design a FA to check whether a given decimal number is divisible by three **05**

Ans.:



The unary number is made up of ones.

The number 3 can be written unary for as 111, number 5 can be written as 11111 and so on. The unary number which is divisible by 3 can be 111 or 111111 or 111111111 and so on. The transaction table is as follows :

Input	1
$\rightarrow q_0$	q_1
q_1	q_2
q_2	q_3
$\textcircled{q_3}$	q_1

Consider a number 111111 which is equal to 6 i.e. divisible by 3. So after complete scan of this number we reach to final state q_3.

Start 111111

State q_0

1 q_1 11111

11 q_2 1111

111 q_3 111

1111 q_1 11

11111 q_2 1

111111 q_3 $\rightarrow$ Now we are in final state.

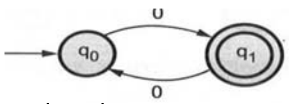
Q.1(b) Differentiate between NFA and DFA with example. **05**

Ans.:

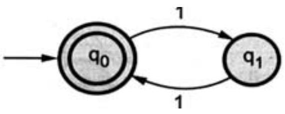
	DFA	NFA
1.	Every input string leads to the unique state of finite automata.	For the same input there can be more than one next states.
2.	Conversion of regular expression to DFA is complex.	Regular expression can be easily converted to NFA using Thompson's construction.
3.	The DFA requires more memory for storing the state information.	The NFA requires more computations to match RE with input.
4.	The DFA it is not possible to move to next state without reading any symbol.	In NFA we can move to next state without reading any symbol.

Q.1(c) Design a DFA to accept a set of all strings which began and end with different letters. **05**

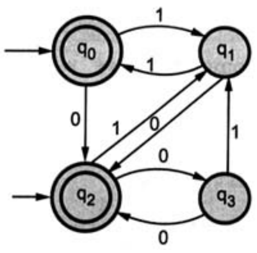
Ans.: Step 1 : W that has zero with old length



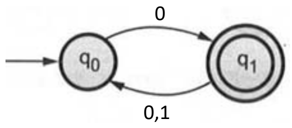
Step 2 : W that has 1 with even length.,



Final DFA



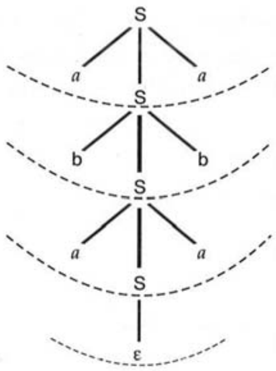
The odd position means 1, 3, 5, ……
This position is occupied by 1.



Q.1(d) Derive a string abaaba for CFG given by G **05**

where P = {$S \rightarrow aSa$
$S \rightarrow bsb$
$S \rightarrow a/b/\varepsilon$}

Ans.:



We have chosen $\rightarrow$ aSa

We have chosen $\rightarrow$ aSb

We have chosen $\rightarrow$ aSa

$S \rightarrow \varepsilon$ is chosen

Q.2(a) Convert the given CFG to CNF consider G(V, T, P, S) **10**
where v = {S, A, B} T = {a, b} P consists of

$S \rightarrow aB$	
$S \rightarrow bA$	**$A \rightarrow baa$**
$A \rightarrow a$	**$B \rightarrow b$**
$A \rightarrow as$	**$B \rightarrow as$**
	$B \rightarrow aBB$

Ans.: Let us start with first rule.

$R_1 \rightarrow a$
$S \rightarrow R_1B$ is in CNF for $S \rightarrow aB$
$R_2 \rightarrow b$
$S \rightarrow R_2A$ is in CNF for $S \rightarrow bA$
$A \rightarrow a$
$A \rightarrow aS$ can be written as
$A \rightarrow R_1S$
Now, $A \rightarrow bAA$ can be written as
$A \rightarrow R_2AA$ replace it by R_3
i.e. $R_3 \rightarrow AA$

then $A \to R_2R_3$

$B \to a$

$B \to bS$ can be written as

$B \to R_2S$ is in CNF

Now, $B \to aBB$

i.e. $B \to R_1BB$ $\because R_1 \to a$

Let $R_4 \to BB$

then $B \to R_1R_4$

Finally we can write,

$S \to R_1B$

$S \to R_2A$

$A \to R_1S$

$A \to R_2R_3$

$B \to b$

$B \to R_2S$

$B \to R_1R_4$

$R_1 \to a$

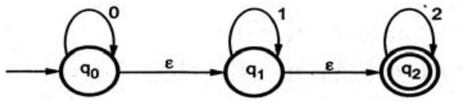
$R_2 \to b$

$R_3 \to AA$

$R_4 \to BB$

is a Chomsky's normal form.

Q.2(b) Convert the given NFA into its equivalent DFA : 10



$$\begin{tikzcd} \arrow[r] & q_0 \arrow[loop above, "0"] \arrow[r, "\varepsilon"] & q_1 \arrow[loop above, "1"] \arrow[r, "\varepsilon"] & q_2 \arrow[loop above, "2"] \end{tikzcd}$$

Ans.: Let us obtain ε – closure of each state

ε–closer $(q_0) = \{q_0, q_1, q_2\}$

ε–closer $(q_1) = \{q_1, q_2\}$

ε–closer $(q_2) = \{q_2\}$

Now we will obtain δ' transaction. Let ε–closer $(q_0) = \{q_0, q_1, q_2\}$ call it as state A.

$\delta'(A,–0) = \varepsilon$–closure$\{\delta((q_0, q_1, q_2))\}$

$= \varepsilon$–closure$\{\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)\}$

$= \varepsilon$–closure$\{(q_0\}$

$= \{q_0, q_1, q_2\}$ i.e. state A

$\delta'(A,–1) = \varepsilon$–closure$\{\delta((q_0, q_1, q_2, 1)\}$

$= \varepsilon$–closure$\{\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)\}$

$= \varepsilon$–closure$\{q_1\}$

$= \{q_1, q_2\}$ Call it as state B

$\delta'(A,–2) = \varepsilon$–closure$\{\delta((q_0, q_1, q_2, 2)\}$

$= \varepsilon$–closure$\{\delta(q_0, 2) \cup \delta(q_1, 2) \cup \delta(q_2, 2)\}$

$= \varepsilon$–closure$\{q_2\}$

$= \{q_2\}$ Call it as state C

Thus we have obtained

$\delta'(A,0) = A \qquad \delta'(A,1) = B \qquad \delta'(A,2) = C$

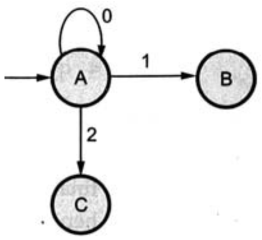
i.e. Now we will find transitions on states B and C for Each input.

Hence $\delta'(B,0) = \varepsilon\text{-closure}\{\delta(q_1,q_2),0\}$

$= \varepsilon\text{-closure}\{\delta(q_1,0) \cup \delta(q_2,0)\}$

$= \varepsilon\text{-closure}\{\ \phi\ \}$

$= \phi$



$\delta'(B,1) = \varepsilon\text{-closure}\{\delta(q_1,q_2),1\}$

$= \varepsilon\text{-closure}\{\delta(q_1,1) \cup \delta(q_2,1)\}$

$= \varepsilon\text{-closure}\ \{q_1\}$

$= \{q_1, q_2\}$ i.e. state B itself.

$\delta'(B,2) = \varepsilon\text{-closure}\{\delta(q_1,q_2),2\}$

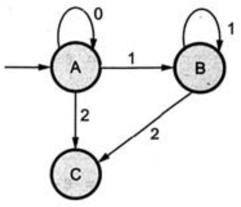
$= \varepsilon\text{-closure}\{\delta(q_1,2) \cup \delta(q_2,2)\}$

$= \varepsilon\text{-closure}\ \{q_2\}$

$= \{q_2\}$ i.e. state C.

Hence $\delta'(B,0) = \phi \qquad \delta'(B,1) = B \qquad \delta'(B,2) = C$

The partial transition diagram will be



Now we will obtain transitions for C :

$\delta'(C,0) = \varepsilon\text{-closure}\{\delta(q_2,0)\}$

$= \varepsilon\text{-closure}\ \{\phi\}$

$= \{\phi\}$

$\delta'(C,1) = \varepsilon\text{-closure}\{\delta(q_2,1)\}$

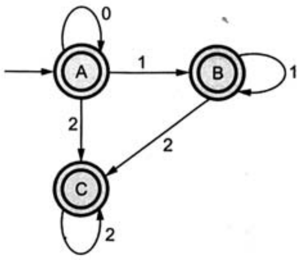
$= \varepsilon\text{-closure}\ \{\phi\}$

$= \{\phi\}$

$\delta'(C,2) = \varepsilon\text{-closure}\{\delta(q_2,2)\}$

$= \{q_2\}$

Hence the DFA is



As A = $\{q_0, q_1, q_2\}$ in which final State lies A is final state in B =$\{q_1, q_2\}$ the state q_2 lies hence B is also final state in C = $\{q_2\}$, the state q_2 lies hence C is also a final state.

Q.3(a) Design a Moore machine and mealy machine for a binary input sequence such that if it has a substring 101 the machine outputs A if input has a substring 110 the machine outputs B otherwise it outputs C. **10**

Ans.: For designing such a machine we need to take care of two conditions and those are checking 101 and checking 110. If we get 101 the output will be A. If we recognize 110 the output will be B. For other strings the output will be C. We can make a partial design of it as shown in Figure 1.

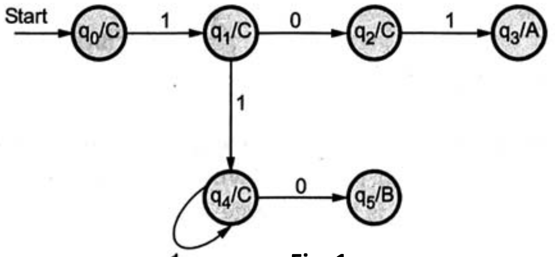
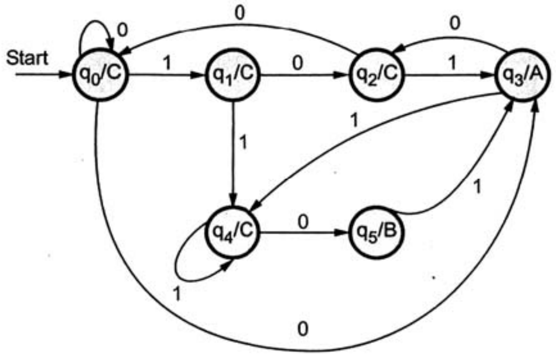
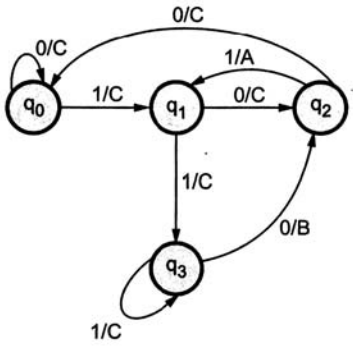


Fig. 1

Now we will insert the possibilities of 1's and 0's for each state. Then the Moore machine becomes.



Now the Mealy machine can be



Q.3(b) Convert the following grammar to CNF form : **10**

S → ABA

A → aA/bA/E

B → bB/aA/E

Ans.: In Chomsky's normal form the ε production is not allowed. So first we will eliminate ε productions.

A → ε and B → ε

If put ε instead of A and B

We can get

S → ABA | AB | BA | A | B

A → aA | Ba | a | b

B → bB | aA | a | b

This $S \to A$ and $S \to B$ is unit production getting introduced in the grammar after removal of ε production. So we will remove unit production also.

$S \to ABA \mid AB \mid BA \mid AA \mid aA \mid bA \mid a \mid b \mid bB \mid aA$
$A \to aA \mid bA \mid a \mid b$
$B \to bB \mid aA \mid a \mid b$

Now let us convert to CNF

1) $S \to ABA \quad \Rightarrow \quad R1 \to BA$
$\therefore \quad S \to AR1$
2) $S \to AB \mid BA \mid AA$ are already in CNF
3) $S \to aA \quad \Rightarrow \quad R2 \to a$
$\therefore \quad S \to R2A$
4) $S \to bA \quad \Rightarrow \quad R3 \to b$
$\therefore \quad S \to R3A$
5) $S \to a \mid b$ is already in CNF
6) $S \to bB \quad \Rightarrow \quad S \to R3B$
7) $S \to aA \quad \Rightarrow \quad S \to R2A$
8) $A \to aA \quad \Rightarrow \quad A \to R2A$
9) $A \to bA \quad \Rightarrow \quad A \to R3A$
10) $A \to a \mid b$ is already in CNF
11) $B \to R3B$
12) $B \to R2A$
13) $B \to a \mid b$ is already in CNF

To summarize,

$S \to AR1 \mid AB \mid BA \mid AA \mid R2A \mid R3A \mid a \mid b$
$A \to R2A \mid R3A \mid a \mid b$
$B \to R3B \mid R2A \mid a \mid b$

Q.4(a) Construct a PDA for the language **10**
$\alpha = \{a^n b^m a^n \mid m, n >= 1\}$

Ans.: The PDA accepting $\{ a^n b^m a^n \mid m, n \geq 1\}$ is given by,

$P = (\{q_0, q_1\}, \{a, b\}, \{a, Z_0\}\delta, q_0, Z_0, \phi)$

Where δ is given as

$$\left.\begin{aligned} &1.\quad \delta(q_0, a, Z_0) = \{(q_0, aZ_0)\} \\ &2.\quad \delta(q_0, a, a) = \{(q_0, aa)\} \end{aligned}\right\} \text{Push a's onto stack.}$$

$$\left.\begin{aligned} &3.\quad \delta(q_0, b, a) = \{(q_1, a)\} \\ &4.\quad \delta(q_0, b, a) = \{(q_0, a)\} \end{aligned}\right\} \text{Read b's and stack will remain unchanged.}$$

$$\left.\begin{aligned} &5.\quad \delta(q_1, a, a) = \{(q_1, \varepsilon)\} \\ &6.\quad \delta(q_1, \varepsilon, Z_0) = \{(q_0, \varepsilon)\} \end{aligned}\right\} \text{Remaining a's are erased. Stack symbol } Z_0 \text{ is erased.}$$

Thus, $(q_0, a^n b^m a^n, Z_0) \overset{*}{\vdash} (q_1, \varepsilon, Z_0) = (q_1, \varepsilon, \varepsilon)$ is PDA.

Q.4(b) Design a PDA to accept a string of balanced parenthesis the parenthesis can contain (,), {, }. **10**

Ans.: **Step 1:** Initially stack will be empty and q_0 be the start state. The empty stack contains Z_0 as initial symbol.

Step 2: When some opening parenthesis is read we will push it onto the stack. Then the state changes to q_1. This transaction can be given as,

$\delta(q_0, (, Z_0) = (q_1, (, Z_0)$
$\delta(q_0, \{, Z_0) = (q_1, \{, Z_0)$

Step 3: $\delta(q_1, (, () = (q_1, (()$
$\delta(q_1, (, \{) = (q_1, (\{)$
$\delta(q_1, \{, () = (q_1, \{ ()$
$\delta(q_1, \{, \{) = (q_1, \{ \{)$

Step 4: Now when we read) parenthesis then pop '(' from the stack. Same is the case for { bracket.

$\delta((q_1,), () = (q_1, \varepsilon)$
$\delta((q_1, \}, \{) = (q_1, \varepsilon)$

Step 5: When we read all the symbols, we will read ε. At that time check the stack contents. If stack is having Z_0 Then that means stack is empty.

$\delta(q_1, \varepsilon, Z_0) = (q_0 ,Z_0)$

The PDA successfully halts here by accepting valid string. The final state is q_0.
Simulate ({ } { }), Z_0)

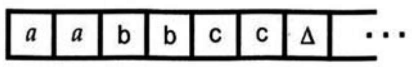
$\vdash (q_1, \{ \}\{ \}, Z_0)$
$\vdash (q_1, \ \}\{ \}), \{(Z_0)$
$\vdash (q_1, \{ \}), Z_0)$
$\vdash (q_1, \}), \{(Z_0)$
$\vdash (q_1,), (Z_0)$
$\vdash (q_1, \varepsilon, Z_0)$
$\vdash (q_0, \varepsilon)$

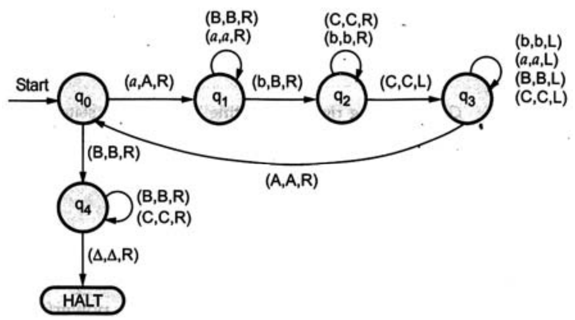
Accept state.

Q.5(a) Construct a TM for L = {a^n b^n c^n | n ≥ 1}. **10**

Ans.: This problem cannot be solved even by PDA. We have solved this type of problem by two stack PDA (we have solved a^n b^n a^n) but turing machine solves this type of problem even! Let us assume that the input is

The simulation for a a b b c c can be shown as below :



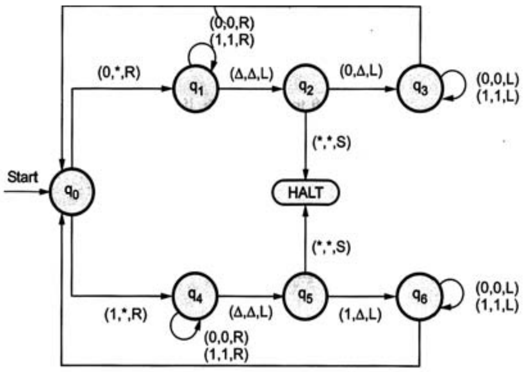


$\underset{\uparrow}{a}\ a\ b\ b\ c\ c\ \Delta$	Convert a to A, move right transition from q_0 to q_1
$A\ \underset{\uparrow}{a}\ b\ b\ c\ c\ \Delta$	Move right upto c
$A\ a\ \underset{\uparrow}{b}\ b\ c\ c\ \Delta$	Convert b to B, move right transition from q_1 to q_2
$A\ a\ B\ b\ \underset{\uparrow}{c}\ c\ \Delta$	Move right upto c
$A\ a\ B\ b\ \underset{\uparrow}{c}\ c\ \Delta$	Convert c to C, move right transition from q_2 to q_3
$A\ a\ B\ b\ C\ \underset{\uparrow}{c}\ \Delta$	Move left upto c
$A\ a\ B\ b\ \underset{\uparrow}{C}\ c\ \Delta$	Move left till A, skipping a, b, B, C, see state q_3 then move one step right
$A\ \underset{\uparrow}{a}\ B\ b\ C\ c\ \Delta$	Convert a to A, move right
$A\ a\ \underset{\uparrow}{B}\ b\ C\ c\ \Delta$	Move right
$A\ A\ B\ \underset{\uparrow}{b}\ C\ c\ \Delta$	Convert b to B and move right, transition from q_1 to q_2
$A\ A\ B\ B\ \underset{\uparrow}{C}\ c\ \Delta$	Move right
$A\ A\ B\ B\ C\ \underset{\uparrow}{c}\ \Delta$	Convert c to C, move left till A, refer q_3 state
$A\ \underset{\uparrow}{A}\ B\ B\ C\ C\ \Delta$	Move right, keep B as it is refer transition from q_0 to q_4
$A\ A\ \underset{\uparrow}{B}\ B\ C\ C\ \Delta$	Go on moving right by ignoring B, C refer q_4 state
$A\ A\ B\ B\ C\ C\ \underset{\uparrow}{\Delta}$	Since Δ is read TM will reach to HALT state
$A\ A\ B\ B\ C\ C\ \underset{\uparrow}{\Delta}$	

Thus TM can very effectively recognize $a^n\ b^n\ c^n$ and more powerful than PDA.

Q.5(b) Construct a TM for checking palindrome of string of odd palindrome for $\sum = (0, 1)$ 10

Ans.: The TM will be very much similar to previous one but with a slight difference.



For example : If the string is

```
0 0 1 0 0 Δ     Move right by making 0 to *
↑
* 0 1 0 0 Δ     Move right upto Δ
  ↑
* 0 1 0 0 Δ     Move left, replace 0 by Δ
          ↑
* 0 1 0 Δ Δ     Move left
        ↑
* 0 1 0 Δ       Move left upto*
      ↑
* 0 1 0 Δ       Move right convert to* and then move right
↑
* * 1 0 Δ       Move right upto Δ
    ↑
* * 1 0 Δ       Move left and replace 0 by Δ
        ↑
* * 1 Δ Δ       Move left till *
      ↑
* * 1 Δ Δ       Move right and convert 1 to *
  ↑
* * * Δ Δ       Move right
    ↑
* * * Δ Δ       Move left
      ↑
* * * Δ Δ       Since it is *, goto HALT state
    ↑
```

Q.6 Write short notes on : 20

Q.6(a) Write short note on applications and limitations of finite automata. 05

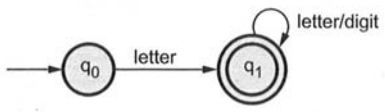
Ans.: The regular expressions can be modelled by finite automata. As we have solved many examples and experienced that regular expressions are effective representation of languages. The smaller unit of regular expression can express the given language over certain input set.

There are following applications of regular expressions and finite automata :

- **Text editors :** Text editors are some programs which are used for processing the text. For example UNIX text editor uses the regular expression can express for substituting the strings, such as
  S/bbb*/b/
  gives that substitute a single blank for the first string of two or more blanks found in a given line.
  In the UNIX text editors any regular expression is converted to an NFA with a transitions, this NFA is then simulated directly.

- **Lexical analyzers :** Complier uses this program of lexical analyzer in the process of compilation. The task of lexican analyzer is to scan the input program and separate out the 'tokens'.
  For example : Identifier is a category of token in the source language and it can be identified by regular expression as
  (letter) (letter + digit)*
  If anything in the source language matches with this regular expression then it is recognized as identifier. The letter is nothing bit a set {A, B, …Z, a, b, …z}* and digit is {0, 1, ... 9} *. The regular expression is effective way for identifying token from a language.

  We can represent the above regular expression using a finite automata as follows :



Limitations of FA :
The main limitation of FA is that it can not remember arbitrarily long input sequence because it does not certain any memory. Hence it can not solve following types of problems.
1. Checking well–formedness of parenthesis.
2. Checking the palindrome condition of given language.

Q.6(b) Write short note on comparison between Moore and Mealy machine. 05
Ans.: Moore and Mealy machines are two FAs with output. The Moore machine can be converted to Mealy and Mealy machine can be converted to Moore. Hence both the machines generate the same output string for corresponding input string. There are certain differences between both the machines – **length of Moore machine** is **one longer** than Mealy machine for given input. Secondly output of the input will be along the edges in case of Mealy machine but it should be associated with the state in case of Moore machine.

Q.6(c) Write short note on halting problem of TM. 05
Ans.: 1. For a given configuration of a TM, two cases arise:
   i) The machine starting at this configuration will halt after a finite number of steps.

ii) The machine starting at this configuration never halts no matter how long it runs.

2. Given any TM, the problem of determining whether it halts or not, is called as ‘halting problem’.

To solve the halting problem, we should have some mechanism such that for any given configuration of a TM, we should be able to determine whether the machine will ever halt or not. In reality, one cannot solve the halting problem. The halting problem is ‘unsolvable’. This means there exists no TM which can determine whether the given TM ‘T’ will ever halt or not.

Proof :
Let us prove that halting problem is unsolvable by contradiction.
Suppose that there exists a TM ‘A’ which decides whether or not any computation by a TM “T” will ever halt, given the description ‘d_T’ of “T” (FM and initial configuration) and the tape ‘t’ of “T”. Then for every input (t, d_T) to ‘A’ if “T” halts for the input‘t’, ‘A’ reaches an “accept halt”; if T does not halt for input ‘t’, then ‘A’ reaches a “reject halt”.

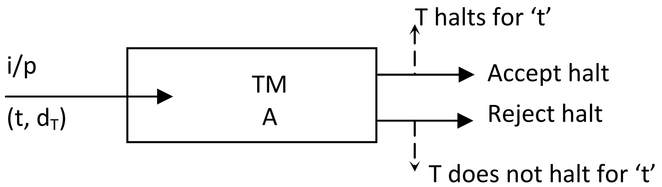


Fig. 1: TM A

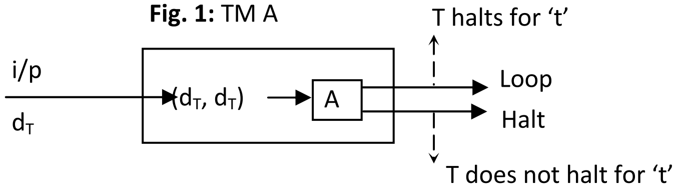


Fig. 2 : TM B

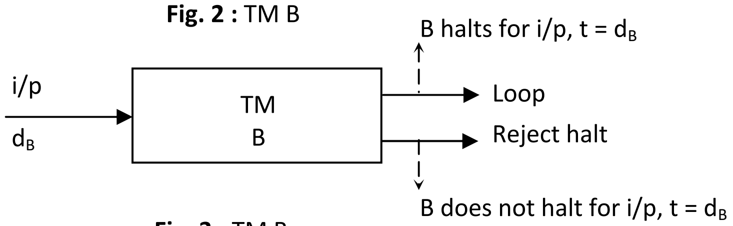

B halts for i/p, t = d_B
i/p
d_B
TM
B
Loop
Reject halt
B does not halt for i/p, t = d_B

Fig. 3 : TM B

3) We can now construct another TM ‘B’ figure (2) which takes ‘d_T’ as the input and proceed as follows:
First it copies the input ‘d_T’ and duplicates ‘d_T’ on its tape and then takes this duplicated information as input to ‘A’ with one modification : whenever ‘A’ is supposed to reach an “accept halt”, B will loop forever. Considering the original behavior of ‘A’ we find that ‘B’ acts as follows:
i) It loops if ‘T’ halts for input t = d_T and
ii) It halts if ‘T’ does not halt for the input t = d_T

Since, ‘B’ itself is a TM, let T = B as shown in figure 3.

Thus, replacing T by B we get that

i) B loop iff B halts for input d_B

ii) B halts iff B does not halt for input d_B

This is a contradiction. Hence we can conclude that machine 'A' which can decide whether any other TM will ever halt, does not exist. Therefore, the halting problem is unsolvable.

4) **Consequences of Halting Problem**

i) It cannot be decided whether a TM ever prints a given symbol ('a' $\subset$ I) of its alphabet. This is also unsolvable.

ii) Two TMs with the same alphabet cannot be checked for equivalence or inequivalence by an algorithm i.e. there is no effective general way to decide whether a given computational process will ever terminate or whether two given processes are equivalent. This is also another unsolvable problem.

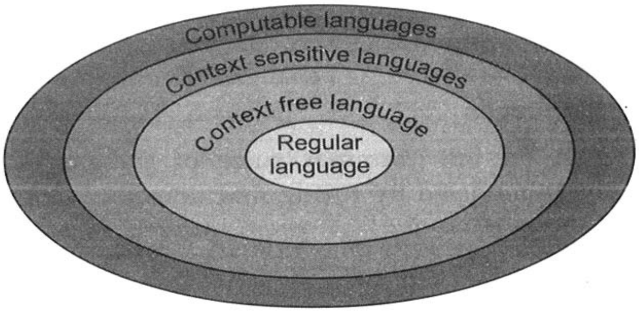
iii) Blank–tape theorem: There exists a TM which when started on a blank tape, can write its own description. This is of interest in constructing self-reproducing machine.

Q.6(d) Write short note on Chomsky Hierarchy of grammar. **05**

Ans.: The Chomsky's Hierarchy represents the class of languages that are accepted by different machine. The category of languages in Chomsky's Hierarchy is as given below :

Language class	Language	Grammar	Machine	One example
Type 3	Regular	Regular grammar	FSM i.e. NFA or DFA	a^*b^*
Type 2	Context free	Context free grammar	PDA	$a^n b^n$
Type 1	Decidable languages	Context sensitive grammar	Linear bounded automata	$a^n b^n c^n$
Type 0	Computable languages	Unrestricted grammar	Turing machine	n!

This is a hierarchy therefore every language of type 3 is also of type 2, 1 and 0. Similarly every language of type 2 is also of type 1 and 0 etc.



Type 3 - Regular languages
Regular languages are those languages which can be described using regular expressions. These languages can be modelled by NFA or DFA.

Type 2 - Context free languages
The context free languages are the languages which can be represented by Context Free Grammar (CFG). The production rule is of the form

$A \rightarrow \alpha$

where A is any single non-terminal and α is any combination of terminals and non-terminals.

A NFA or DFA cannot recognize strings of this language because these automata are not having "stack" to memorize. Instead of it the Push Down Automata can be used to represent these languages.

Type 1 - Context sensitive languages
The context sensitive grammars are used to represent context sensitive languages. The context sensitive grammar is follows the following rules –

1. The context sensitive grammar may have more than one symbol on the left hand side of their production rules.
2. The number of symbols on the left hand side must not exceed the number of symbols on the right hand side.
3. The rule of the form $A \rightarrow \varepsilon$ is not allowed unless A is a start symbol. It does not occur on the right hand side of any rule.

The automaton which recognizes context sensitive languages is called linear bounded automaton. While deriving using context sensitive grammar the sentential form must always increase in length every time a production rule is applied. Thus the size of a sentential form is bounded by a length of the sentence we are deriving.

Type 0 - Unrestricted languages
There- is no restriction on the grammar rules of these type of languages. These languages can be effectively modeled by turing machines.

❑ ❑ ❑ ❑ ❑