# TOC akash previous year questions

Information Technology (Guru Gobind Singh Indraprastha University)

Scan to open on Dotnotes

# Syllabus

## THEORY OF COMPUTATION
## (ETCS-206)

**Instruction to Paper Setters:** **Maximum Marks : 75**

1. *Question No. 1 should be compulsory and cover the entire syllabus. This question should have objective or short answer type questions. It should be of 25 marks.*
2. *Apart from Question No. 1, rest of the paper shall consist of four units as per the syllabus. Every unit should have two questions. However, student may be asked to attempt only 1 question from each unit. Each question should be 12.5 marks.*

### UNIT-I

Overview: Alphabets, Strings & Languages, Chomsky Classification of Languages, Finite Automata, Deterministic finite Automata (DFA) & Nondeterministic finite Automata (NDFA), Equivalence of NDFA and DFA, Minimization of Finite Automata, Moore and Mealy machine and their equivalence, Regular expression and Kleen's Theorem(with proof), Closure properties of Regular Languages, Pumping Lemma for regular Languages(with proof).

**[T1, T2] [No. of hrs. 11]**

### UNIT-II

Context free grammar, Derivation trees, Ambiguity in grammar and its removal, Simplification of Context Free grammar, Normal forms for CFGs: Chomsky Normal Form & Greibach Normal Form, Pumping Lemma for Context Free languages, Closure properties of CFL(proof required), Push Down Automata (PDA), Deterministic PDA, Non Deterministic PDA, Equivalence of PDA and CFG, Overview of LEX and YACC.

**[T1, T2] [No. of hrs. 11]**

### UNIT-III

Turing machines, Turing Church's Thesis, Variants and equivalence of Turing Machine, Recursive and recursively enumerable languages, Halting problem, Undecidability, Examples of Undecidable problem. **[T1, T2] [No. of hrs. 11]**

### UNIT-IV

Introduction to Complexity classes, Computability and Intractability, time complexity, P, NP, Co-NP, Proof of Cook's Theorem, Space Complexity, SPACE, PSPACE, Proof of Savitch's Theorem, L, NL, Co-NL complexity classes.

**[T1, T2] [No. of hrs. 11]**

# FIRST TERM EXAMINATION [FEBRUARY-2015]
# FOURTH SEMESTER [B. TECH]
# THEORY OF COMPUTATION [ETCS-206]

**Time: 1 Hour** **MM : 30**

**Note:** *Attempt any five question. All questions carry equal marks.*

**Q.1.(a) What is the difference between the strings and the words of Language? Explain with example.**

**Ans. Strings:** A string is any combination of the letters of an alphabet.

**For example:** Alphabet $\Sigma = \{a, b\}$

Here $a$, $b$ are the letters of this alphabet.

As you can see make a lot of string from these letters $a$ and $b$.

a, b, aa, ab...

**Words:** The words of a Language are the strings that are always made according to certain rules used to define that language.

**For example:** Words of this Language would have only those strings that have only add no. of b's and no a's. Some example words of our defined Language are

*b, bbb, bbbbb* ..... and so on.

**Note:** We can say that all the words are strings but all the strings may not be the words of a Language.

**Q.1. (b) Differentiate between kleen closure and positive closure using suitable example.**

**Ans.**

| | Kleen Closure | Positive Closure |
|---|---|---|
| (*i*) | It is denoted by $\Sigma^*$ | (*i*) It is denoted by $\Sigma^+$. |
| (*ii*) | The set of strings, include the empty string over an alphabet $\Sigma$. | (*ii*) The set of string, exclude The empty string over on alphabet $\Sigma$. |
| (*iii*) | **For exapmple:-** $\Sigma = \{0\}$ $\{0\}^* = \{\varepsilon, 0, 00, 000...\}$ | (*iii*) **For Example:** $\Sigma = \{0\}$ $\{0\}^+ = \{0, 00, 000, ...\}$ |
| (*iv*) | $\Sigma^* = \Sigma^+ U \{0\}$ | (*iv*) $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ |

**Q.2. (a) Define Regular Language. What is the difference between (a, b) and (a +b).**

**Ans. Regular Languge:** The set of regular Languages over an alphabet is defined recursively as below. Any Language belonging to this set is a regular-Language over.

**Basis Clause:** $\phi$, $\{\}$ and $\{a\}$ for any symbol a belongs to ($\varepsilon$) are regular Languages.

**Inductive Clause:** If $L_r$ and $L_s$ are regular Language, then $L_rL_s$, $L_r + L_s$ and $L_r^*$ are regular Language.

**Defference between (a,b) and (a+b):** In regular expressions, $(a,b)$ and $(a + b)$ are same. There is no difference between them.

**Q.2. (b) The reverese of a string is defined by recursive rules $a^R = a$. For all $a\varepsilon$ $\Sigma$ and $w\varepsilon$ $\Sigma$, there is**

$$(wa)^R = aw^R$$

**Using this prove that $(uv)^R = v^R u^R$.**

**Ans.** One solution is recursively define a reversing operation on regular expression, and apply that operation on the regular expression for A. In particular, given a regular expression $R$, reverse $(R)$ is:

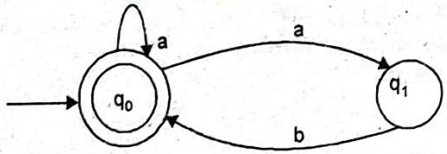
- $a$ for some $a \in \Sigma$.
- $\varepsilon$ if $R = \varepsilon$,
- $\phi$ if $R = \phi$,
- (reverse $(R_1)$ $U$ reverse $(R_2)$, if $R = R_1 U R_2$.
- (reverse $(R_2)$ o reverse $(R_1)$, if $R = R_1 o R_2$ or
- (reverse $(R_1)^*$), if $R = (R_1{}^*)$

Another solution is to start with a DFA M for A, and build a NFA M' for $A^R$ as follows: reverse all the arrows of M and designate the start state for M as the only accept state q'acc for M'. Add a new start state $q_o'$ for M', and from $q_o'$, add ε-transitions to each state of M' corresponding to accept states of M.

It is easy to verfy that for any $\omega \in \Sigma^*$, these is path following w from the state start to an accept state in M iff there is a path following $w^R$ from $q'_0$ to $q'_{acc}$ in M'. If follows that $\omega \in A$ iff $w^R \in A^R$.

**Q.3. (*a*) Draw the NFA that accept the Language $a^* + (ab)^*$.**

**Ans.**




In case of NFA we know that there is more transitions possible for the same state. So, when we design given Language than comes in mind same point.

**Q.3. (*b*) Design a FA that accepts all binary strings where o's and 1's alternate.**

**Ans.** First of all we have to design regular expression for o's and 1's alternate.

There are two cases for given condition:-

(*i*) For odd number of string for o's and 1's alternate accepted by FA.

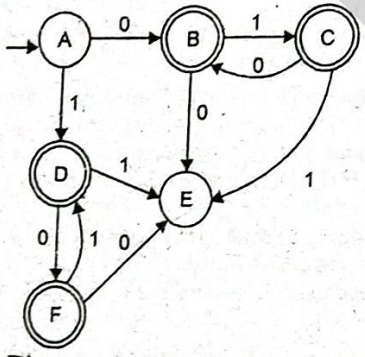
$$(01)^*0 + (10)^* 1$$

(*ii*) For even number of string for o's and 1's alternate accepted by FA.

$$(01)^+ + (10)^+$$

So add two cases then resulting regular expression is :-

$$(01)^+ + (10)^+ + (01)^* 0 + (10)^* 1$$




**Transition Diagram for Binary 0's and 1's alternate**

In this case A is initial state i.e starting state. B, C, D and F all these all final states or Acceptable state that can be accepted only alternate binary o's and 1's either even or odd.



At the end Last E state is known as **Dead** state that is not connected to final state.

**Acceptable string 0101:**

$\delta$ (A, 0101)
— $\delta$ (B, 0101)
— $\delta$ (C, 01)
— $\delta$ (B, 1)
— C [Final State]

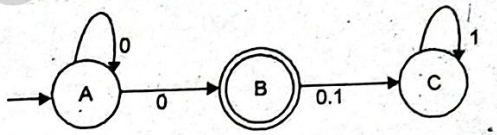
So, 0101 is acceptable string for designing automata.

**Non-acceptable string 011:**

$\delta$ (A, 011)
— $\delta$ (B, 11)
— $\delta$ (C, 1)
— E [Non-Final State]

So, 011 is non-acceptable string because terminate by Non-Final state.

**Q.4. (*a*) Convert the following NFA (Figure Given belows into an equivalent DFA. In the figure, A is start state and B is Final state.**




**Ans.** State Table for given Figure

| State/Σ | 0 | 1 |
|---|---|---|
| → A | A,B | – |
| B | B | B |
| C | – | C |

The Deterministic automaton $M_1$ equivalent to given NFA, M is defined as follows:

$$M_1 = (2^Q, \{0, 1\}, \delta, [A], F')$$

where

$$F = \{[B], [A,B], [B, C], [A, B, C]\}$$

we start the construction by considering [A] first. We get [A,B]. Then we construct δ for for [A,B]. [B] is a new state appearing under the input columns. After constructing δ for [B], we don not get any new states and so we terminate the construction of δ. The state table is given by Table.

**State Table for $M_1$ for above figure**

| *State* / Σ | 0 | 1 |
|---|---|---|
| → [A] | [A,B] | – |
| [A,B] | [A,B] | [B] |
| [B] | [B] | [B] |

**Q.4. (b) What is the significance of Melay and Moore Machine in computing?**

**Ans. Significance of Melay and Moore Machine:-**

(1) Melay Machine model may be more flexible than the moore Machine, it is the needs of the system being analyzed or designed that determines which of the two is most suitable.

(2) Note that a given state machine may be comprised of both Melay and Moore models, if such a design meets the functional requirements of the system.

(3) The point here is that the correct state logic required for efficent operation is what's important, the resulting machine archetype (Melay or Moore) should be only a secondary obersevation.

**Q.5. (a) How pumping Lemma can be applied to prove that certain sets are not regular?**

**Ans.** Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton with $n$ states. Let $L$ be the regular set accepted by M. Let $w \in L$ and $|w| \geq m$. If $m \geq n$, then there exists $x$, y, z such that $w = xyz, y \neq \wedge$ and $xy^i z \in L$ for each $i \geq 0$.

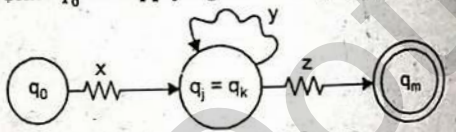
**Proof:** Let

$$w = a_1 a_2 \ldots\ldots a_m,\ m \geq n$$
$$\delta(q_0, a_1 a_2 \ldots a_m) = q_i \text{ for } i = 1, 2, \ldots\ldots m;$$
$$Q_1 = \{q_0, q_1, q_2, \ldots\ldots, q_m\}$$

That is, $Q_1$ is the sequence of states in the path with path value $w = a_1 a_2 \ldots\ldots a_m$. As there are only n district states, at least two states in $Q_1$ must coincide. Among the various pairs of repeated states, we take the first. Let us take them as $q_i$ and $q_k$ $(q_j = q_k)$. Then $j$ and $k$ satisfy the condition $0 \leq j < k \leq n$.

The string $w$ can be decomposed into three substrings $a_1 a_2 \ldots a_j$, $a_{j+1} \ldots\ldots a_k$ and $a_{k+1} \ldots\ldots a_m$. Let $x, y, z$ denote these strings $a_1 a_2 \ldots a_{i+1} \ldots a_k$, $a_{k+1} \ldots a_m$, respectively. As $k \leq n$, $|xy| \leq n$ and $w = xyz$. The path with the path value $w$ in the transition diagram of $M$ is shown is Fig below.

The automaton $M$ starts from the inital state $q_0$. On applying the string $x$, it reaches $qj\,(= qk)$. On applying the string $y$, it comes back to $qj\,(= qk)$. On so after application of $y^i$ for each $i \geq 0$, the automotor is in the same state if. On applying $z$, it reaches $q_m$, a final state. Hence, $x, y^i z \in L$. As every state in $Q_1$ is obtained by applying an input symbol, $y \neq \wedge$.

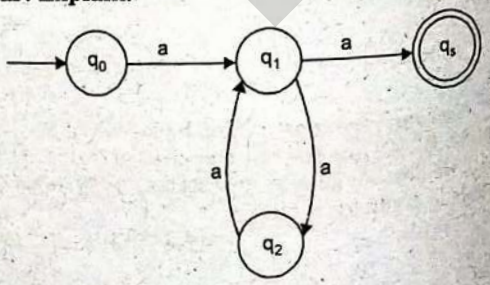



**String accepted by M**

**Q.5. (b) Is $L = \{a^{2n} / n \geq 1\}$ regular? Explain.**

**Ans.** We can write $a^{2n}$ as $a\,(a^2)^i\,a$, where $i \geq 0$.

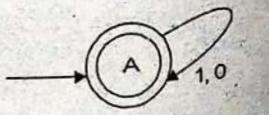
Now $\{(a^2)^i / i \geq 0\}$ is simply $\{a^2\}^*$. So $L$ is represented by the regular expression $a\,(p)^*\,a$, where $p$ represents $\{a^2\}$. The corresponding finite automation is shown in below figure.




Finite Automation of $L = \{a^{2n} / n \geq 1\}$

**Q.6. (a) Determine the regular expression for the DFA given below using Arden's Theorem.**

**Ans.** There is only one initial state. Also, there are no $\wedge$- moves. The equation is






$$A = A(1 + 0) + \wedge$$

By applying Theorem $[R = Q + RP$ then $R = QP^*]$ to the A-equation, we get

$$A = \wedge (1+0)^*$$

Therefore,

$$A = (1 + 0)^*$$
$$[\because \wedge R^* = R^* \wedge = R^*]$$

So, the regular expression for given DFA is $(1+0)^*$.

**Q.6. (b) Design a grammer generating the Language**

$$L = \{a^n b^m / m, n \geq 1\}$$

**Ans:** There are three cases i.e.

(i) $m = n$

(ii) $m < n$

(iii) $m > n$

Hemce the production rules will be

$$\left.\begin{aligned} S &\rightarrow A / B / C \\ A &\rightarrow aAb / ab \\ B &\rightarrow aab / aaBb / aBbb \\ C &\rightarrow acb / accb / aacb \end{aligned}\right\} \text{ These three cases covered in all the grammars.}$$

**Q.7. (a) Define the following terms with example:**

**(a) Chomsky Classification:**

**Ans. Chomsky Hierarchy of Grammars:** We can exhibit the relationship between grammars by the Chomsky Hierarchy. Noum Chomsky, a founder of formal Language theory, provided on initial classification into four languages types:

(i) Type-0 (Unrestricted Grammer)

(ii) Type-1 (Context Sensitive Grammar)

(iii) Type-2 (Context Free Grammar)

(iv) Type-3 (Regular Grammar)

The modified Chowsky Hierarchy of grammars is given by:

**Unrestricted (Type-0) Grammars:** A set of productions of following form

$$\alpha \rightarrow \beta$$

where $\alpha$ and $\beta$ are arbitary string of grammar symbols with $a \neq \in$. These grammars are known as type–0, phase - structure or unrestricted grammars.

**Context Sensitive Grammar:** $P$ is the set of rules called productions defined as

$$\alpha \rightarrow \beta$$

where $\beta$ is at least as long as $\alpha$ that is clearly

$$|\alpha| \leq |\beta|$$

The term "Context-Sensitive" comes from a normal form for these grammars, where each production is of form $\alpha_1, A\,\alpha_2 \rightarrow \alpha_1\,\beta\,\alpha_2$, with $\beta \neq \in$. Replacement of variable A by string $\beta$ is permitted in the "context" of $\alpha_1$ and $\alpha_2$.


**Context Free Grammars:** G is context-free and all production in P have the form:

$$\alpha \rightarrow \beta$$

where $\alpha \in V$ and $\beta \in (V \cup N \cup T)$

It means L.H.S. should contain only one variable.

**Regular Grammars:** If all production of a CFG are of the form $A \rightarrow \omega B$ or $A \rightarrow \omega$, where A and B are variables and $\omega \in NT$, then we say that grammar is right Linear.

**Q.7. (*b*) Arden's Theorem.**

**Ans. Theorem:** Let $P$ and $Q$ be two regular expressions over $\Sigma$. If $P$ does not contain ^, then the following equation in $R$, namely

$$R = Q + RP \qquad ...(1)$$

has a unique solution (*i.e.* one and only one solution given by $R = QP^*$).

**Proof:** $Q + (QP^*)P = Q(^ + P^*P) = QP^*$

Hence Eq (1) is satisfied when $R = QP^*$. This means $R = QP^*$ is a solution of Eq (1).

To prove uniqueness, consider Eq (1). Here, replacing $R$ by $Q+RP$ on the R.H.S., we get the equation

$$\begin{aligned} Q + RP &= Q + (Q + RP)P \\ &= Q + QP + RPP \\ &= Q + QP + RP^2 \\ &= Q + QP + QP^2 + ....+ QP^i + RP^{i+1} \\ &= Q(^ + P + P^2 + .....+P^i) + RP^{i+1} \end{aligned}$$

From Eq (1)

$$R = Q(^ + P + P^2 + ....._p^i) + RP^{i+1} \text{ for } i \geq 0 \qquad ..(2)$$

We now show that any solution of Eq...(1) is equivalent to $QP^*$. Suppose $R$ satisfies Eq...(1) it satisfies Eq-2. Let $w$ be a string of Length $i$ in the set $R$. The $w$ belongs to the set $Q(^ + p + p^2 + .....+p^i) + RP^{i+1}$. As $P$ does not contain ^. $RP^{i+1}$ has no string of length less than $i + 1$ and so $w$ is not in the set $RP^{i+1}$. This means that $w$ belongs to the set $Q(^ + P + P^2 + ....+ P^i)$ and hence to $QP^*$.

Consider a string $w$ in the set $QP^*$. Then $w$ is in the set $QP^K$ for some $K \geq 0$, and hence in $Q(^ + P + P^2 + .....+ P^k)$. So $w$ is on the R.H.S. of Eq...(2) Therefore, $w$ is in $R$. Thus $R$ and $QP^*$ represent the same set. This proves the uniquness of the solution of Eq-(2).

# SECOND TERM EXAMINATION [APRIL-2015]
# FOURTH SEMESTER [B. TECH]
# THEORY OF COMPUTATION [ETCS-206]

**Time: 1 Hour** **MM : 30**

**Note:** *Q.No. 1 is compulsory. Attempt any Two more questions. All questions carry equal marks.*

**Q.1. (*a*) What is context Free Grammar? How it is useful to generate context free Language using Pushdown Automata?**

**Ans. Context Free Grammer:** $G$ is context-Free if every production is of the form $A \rightarrow \alpha$, where $A \in V_N$ and $\alpha \in (V_N \cup \Sigma)^*$.

**For example:**

$$P = \{ S \rightarrow S + S \\ S \rightarrow S * S \\ S \rightarrow 4 \}$$

we are using following conventions:–

1. The capital letters are used to denote the Non-terminals.
2. The lower case letters are used to denote the terminals.

**How it is useful to generate CFL using PDA:** For converting given CFG to PDA, by this method the neccessary condition is that the first symbol on R.H.S production must be a terminal symbol. The rule that can be used to obtain PDA from CFG is

**Rule 1:** For non-terminal symbols, add following rule

$$\delta(q, \in, A) = (q, \alpha)$$

where the production rule is $A \rightarrow \alpha$.

**Rule 2:** For each terminal symbols, add following rule.

$$\delta(q, a, a) = (q, \varepsilon) \text{ for every terminal symbol}$$

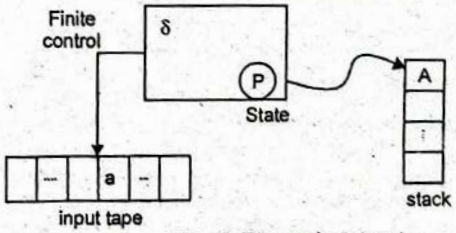
**Q.1. (*b*) Differentiate between finite Automator and pushdown Automata.**

**Ans.**

| Finite Automata | Push Down Automata |
|---|---|
| 1. A DFA is a "state" machine. | 1. A PDA is a "stack" machine |
| 2. DFA and NFAs cannot do that stuff, but any DFA or NFA can also be represented as a push down automata. | 2. A PDA can actually store information in a stack as it process it. It can then choose what to do next by looking at the top of the stack. |

**Q.1. (*c*) Explain the operations on stack of Pushdown Automata.**

**Ans.** Pushdown automata differ from finite state machines in two ways:




**A diagram of Push Down Automata**

1. They can use the top of the stack to decide which transtion to take.
2. They can manipulate the stack as part of performing a transition.

There are mainly two opertions performed on pushdown automata.

**(i) Push:** The manipulation can be to push a particular symbol to the top of the stack.

**(ii) Pop:** The manipulation can be to pop off the top of the stack.

**Q.1. (d) Give the formal defination of Turing Machine.** **(2)**

**Ans.** A turing machine $M$ is a 7- tuple, namely $(Q, \Sigma, \tau, \delta, q_o, b, F)$, where

(i) $Q$ is a finite nonempty set of states.

(ii) $\tau$ is a finite nonempty set of tape symbols

(iii) $b \in \tau$ is the blank.

(iv) $\Sigma$ is a nonempty set of I/P symbols and is a subset of $\tau$ and $b \notin \Sigma$.

(v) $\delta$ is the transition function mapping $(L', x)$ in to $(q', y, D)$ where $D$ denotes the direction of movement of $R/w$ head; $D = L$ or $R$ according as the movement is to the left or right.

(vi) $q_o \in Q$ is the initial state

(vii) $F \subseteq Q$ is the set of final states.

**Q.2. (a) Explain the Language accepted by Pushdown Automata using Final state and empty state with suitable example.**

**Ans. PDA Using Final State:-** Let $A = (Q, \Sigma, \tau, \delta, q_o, z_o, F)$ be a pda. The set accepted by PDA by final state is defined by

$$T(A) = \{w \in \Sigma^* / (q_o, w, z_o) \vdash^* (q_f \wedge, \alpha)$$

$$q_f \in F \text{ and } \alpha \in \Gamma^*\}$$

**Example:** Construct a pda A accepting $L = \{wcw^T / w\varepsilon(a,b)^*\}$ by final state.

**Solution:**

Let $wcw^T \varepsilon L$. write $w = a_1 a_2 .... a_n$, where each $ai$ is either $a$ or $b$. Then, we have

$$\begin{aligned}(q_o, a_1, a_2 .... a_n\, cw^T, z_0) &\xrightarrow{*} (q_o, cw^T, a_n a_{n-1} ..... a_1 z_o)\\ &\xrightarrow{*} (q_1, a_n a_{n-1} ..... a_n a_{n-1} ..... a, z_o)\\ &\xrightarrow{*} (q_1, \wedge, z_0)\\ &\xrightarrow{*} (q_f, \wedge, z_0)\end{aligned}$$

Therefore, $wcw^T \in T(A)$, *i.e.* $L \subseteq T(A)$.

**PDA using Null state:** Let $A = (Q, \Sigma, \tau, \delta, q_0, z_0, F)$ be a pda. The set N (A) accepted by null store is define by

$$N(A) = \{w \in \Sigma^* / (q_o, w, zo) \xrightarrow{*} (q, \wedge, \wedge) \text{ for some } q \in Q\}$$

**Example:**

$$N(A) = \{wcw^T / w\varepsilon(a,b)^*\}$$

**Solution:** From the construction of A, we see that the rules given below cannot erase $z_0$. we make a provision for erasing $z_0$ from PDS by Rule 13. By $wcw^T \varepsilon T(A)$ if and only if the pda reaches the ID $(q_f, \wedge, z_o)$. By rule 13, PDS can be emptied by ^-moves if and only if the pda reaches the ID $(q_f \wedge, z_o)$.

Hence $N(A) = \{wcw^T / w \in (a,b)^*\}$

Rules for $L = \{wcw^T / w \in (a,b)^*\}$

**Rule 1:** $\delta(q_o, a, zo) = \{(q_o, azo)\}$

**Rule 2:** $\delta(q_o, b, zo) = \{(q_o, bzo)\}$

**Rule 3:** $\delta(q_o, a, a) = \{(q_o, aa)\}$

**Rule 4 :** $\delta(q_o, b, a) = \{(q_o, ba)\}$

**Rule 5:** $\delta(q_o, a, b) = \{(q_o, ab)\}$

**Rule 6:** $\delta(q_o, b, b) = \{(q_o, bb)\}$

**Rule 7:** $\delta(q_o, c, a) = \{(q_1, a)\}$

**Rule 8:** $\delta(q_o, c, b) = \{(q_1, b)\}$

**Rule 9:** $\delta(q_o, c, zo) = \{(q_1, zo)\}$

**Rule 10:** $\delta(q_1, a, a) = \{(q_1, \wedge)\}$

**Rule 11:** $\delta(q_1, b, b) = \{(q_1, \wedge)\}$

**Rule 12:** $\delta(q_1, \wedge, zo) = \{(q_\rho, zo)\}$

**Rule 13:** $\delta(q_\rho, \wedge, zo) = \{(q_f \wedge)\}$

**Q.2. (b) Draw a pushdown Automata for the CFG given below**

$$S \rightarrow a\,S\,b \;\; ; \;\; S \rightarrow a/b/\varepsilon$$

**Ans:** This is a Language from given CFG of equal number of a's and equal number of b's or Total number of a's one more than total number of b's in input string or total number of b's one more than total number of a's in input string. and number of a's are followed by number of b's. The instantaneous description can be given as–

$$\begin{aligned}\delta(q_o, a, zo) &= (qo, az_o)\\ \delta(q_o, a, a) &= (q_o, aa)\\ \delta(q_o, b, a) &= (q_1, \varepsilon)\\ \delta(q_1, b, a) &= (q_1, \varepsilon)\\ \delta(q_1, b, a) &= (q_1, \varepsilon)\\ \delta(q_1, \varepsilon, b) &= (q_\rho \varepsilon),\\ \delta(q_1, a, zo) &= (q_f \varepsilon),\\ \delta(q_1, \varepsilon, zo) &= (q_\rho \varepsilon)\end{aligned}$$

where $q_o$ is a start state and $qf$ is accept state.

We will simulate this PDA for the following string:

$$\begin{aligned}(q_o, aab, zo) &\vdash (q_o, ab, azo)\\ &\vdash (q_o, b, aazo)\\ &\vdash (q_1, \varepsilon, az_o)\\ &\vdash (q_f, \varepsilon) \text{ Accept state.}\end{aligned}$$

**Q.3. (a) Explain the representation of Turing Machine by Instantaneous Description using diagram.**

**Ans.** A snapshot of Turing machine is shown in below figure. Obtain the instantaneous descrition.

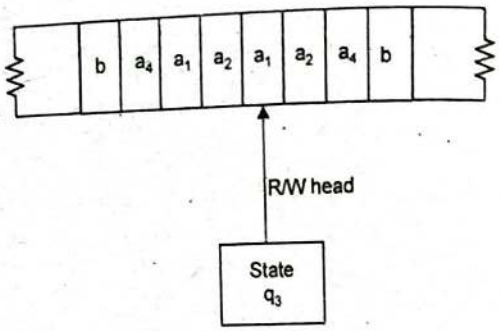
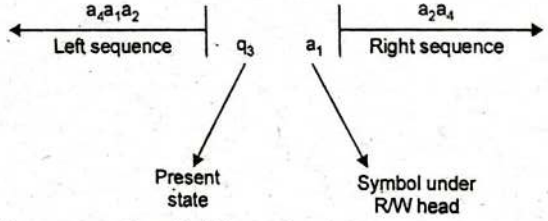



**Fig. Snapshot of Turing Machine**

The present symbol under the R/W head is $a_1$. The present state is $q_3$. So $a_1$ is written to the right of $q_3$. The nonblank symbols to the left of a, from the string $a_4\, a_1\, a_2$, which is written to the left of $q_3$. The sequence of nonblank symbols to the right of $a_1$ is $a_2\, a_4$. Thus the ID is an given in Figure.




**Notes:** (*i*) For constructing the ID, we simply insert the current state in the input string to the left of the symbol under the R/W head.

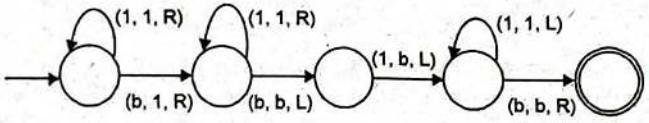
(*ii*) We observe that the blank symbol may occur as par of the ceft or right substring.

**Q.3. (*b*) Consider the turing machine given by the following table:**

| Present State | Input-type 1 | symbols *b* |
|---|---|---|
| $\rightarrow q_o$ | $1\,R\,q_0$ | $1\,Rq_1$ |
| $q_1$ | $1\,R\,q_1$ | $b\,L\,q_2$ |
| $q_2$ | $b\,Lq_3$ | – |
| $q_3$ | $1\,Lq_3$ | $b\,R\,q_4$ |
| $q_4$ | – | – |

**Draw the transition diagram of the Turing machine, where tape symbols are {1,b} and L, R are R/w head moves.**

**Ans.**

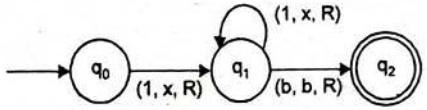



Transition diagram for given TM



**Q.4. (*a*) Design a Turing machine that accepts the Language denoted by the regular expression 11*.**

**Ans.** Give regular expression 11* means atleart one 1's should be present only that expression accepted by Turing machine. So the machine will be:




Transition diagram for given TM

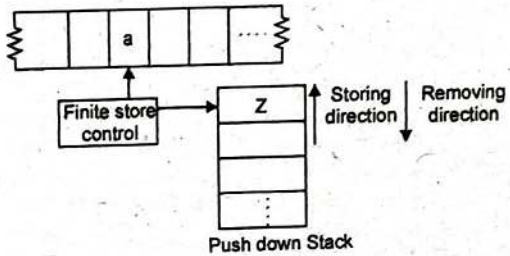
| Present State | Input-tape 1 | symbols b |
|---|---|---|
| $\rightarrow qo$ | $x\,r\,q_1$ | – |
| $q_1$ | $xR\,q_1$ | $bR\,q_2$ |
| $q_2$ | – | – |

**Transition Table for 11***

**Q.4. (*b*) Write short notes on the following:**

**Q.(*i*) Pushdown stack**

**Ans.**




**Model of push down Automata**

**Pushdown stack:** It has a stack called the push down stack (abbreviated PDS). It is a read-write pushdown store as we add elements to PDS or remove elements from PDS.

**Q.(*ii*) Universal Turing Machine**

**Ans. Universal Turing Machine:** The universal Language Lu is a set of binary strings which can be modeled by a turing machine. The universal Language can be represented by a pair (M,w) where M is a TM string in (O+1)* such that *w* belongs to *L* (M). Thus we can say that any binary string belongs to a universal language. From this the concept of universal Turing machine came up. The universal Language can be represented by Lu = L (*U*) where *U* is a universal turing machine. In fact *U* is a binary string represents various codes of many turing machines Thus the UTM is a TM which accepts many turing machines. The TM U is a multitape TM which can be shown below.

- The UTM accepts the TM M.
- The transitions of M are stored initially on first tape along with the string w.
- On the second tape the simulated tape of M is placed. here the tape symbol $x_i$ of M will be represented by $o^i$ and tape symbols are separated by single 1's.

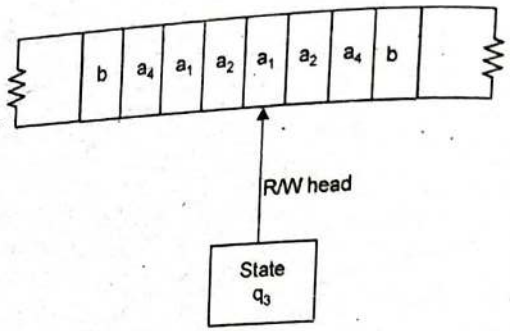
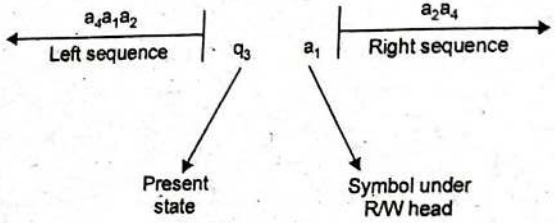



**Fig. Snapshot of Turing Machine**

The present symbol under the R/W head is $a_1$. The present state is $q_3$. So $a_1$ is written to the right of $q_3$. The nonblank symbols to the left of a, from the string $a_4\, a_1\, a_2$, which is written to the left of $q_3$. The sequence of nonblank symbols to the right of $a_1$ is $a_2\, a_4$. Thus the ID is an given in Figure.




**Notes:** (*i*) For constructing the ID, we simply insert the current state in the input string to the left of the symbol under the R/W head.

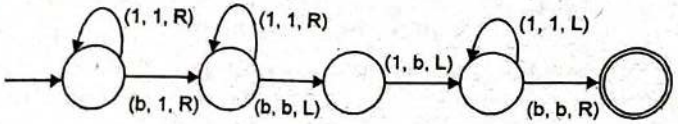
(*ii*) We observe that the blank symbol may occur as par of the ceft or right substring.

**Q.3. (*b*) Consider the turing machine given by the following table:**

| Present State | Input-type 1 | symbols *b* |
|---|---|---|
| $\rightarrow q_o$ | $1\,R\,q_0$ | $1\,Rq_1$ |
| $q_1$ | $1\,R\,q_1$ | $b\,L\,q_2$ |
| $q_2$ | $b\,Lq_3$ | – |
| $q_3$ | $1\,Lq_3$ | $b\,R\,q_4$ |
| $q_4$ | – | – |

**Draw the transition diagram of the Turing machine, where tape symbols are {1,b} and L, R are R/w head moves.**

**Ans.**

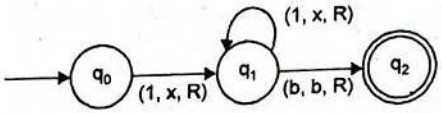



Transition diagram for given TM



**Q.4. (*a*) Design a Turing machine that accepts the Language denoted by the regular expression 11*.**

**Ans.** Give regular expression 11* means atleart one 1's should be present only that expression accepted by Turing machine. So the machine will be:




Transition diagram for given TM

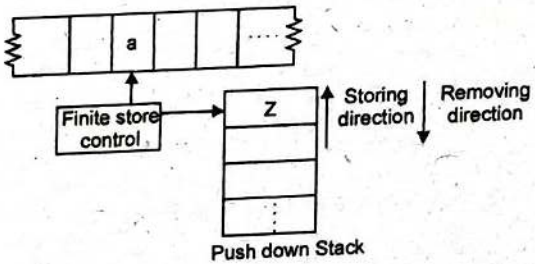
| Present State | Input-tape 1 | symbols b |
|---|---|---|
| $\rightarrow qo$ | $x\,r\,q_1$ | – |
| $q_1$ | $xR\,q_1$ | $bR\,q_2$ |
| $q_2$ | – | – |

**Transition Table for 11***

**Q.4. (*b*) Write short notes on the following:**

**Q.(*i*) Pushdown stack**

**Ans.**



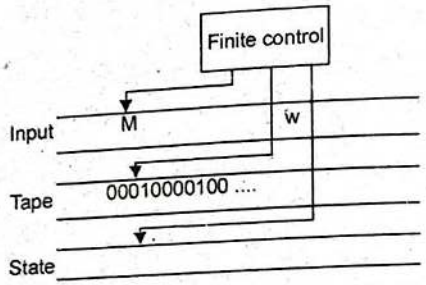

**Model of push down Automata**

**Pushdown stack:** It has a stack called the push down stack (abbreviated PDS). It is a read-write pushdown store as we add elements to PDS or remove elements from PDS.

**Q.(*ii*) Universal Turing Machine**

**Ans. Universal Turing Machine:** The universal Language Lu is a set of binary strings which can be modeled by a turing machine. The universal Language can be represented by a pair (M,w) where M is a TM string in (O+1)* such that $w$ belongs to $L$ (M). Thus we can say that any binary string belongs to a universal language. From this the concept of universal Turing machine came up. The universal Language can be represented by Lu = L ($U$) where $U$ is a universal turing machine. In fact $U$ is a binary string represents various codes of many turing machines Thus the UTM is a TM which accepts many turing machines. The TM U is a multitape TM which can be shown below.

- The UTM accepts the TM M.
- The transitions of M are stored initially on first tape along with the string w.
- On the second tape the simulated tape of M is placed. here the tape symbol $x_i$ of M will be represented by $o^i$ and tape symbols are separated by single 1's.




• On the third tape various states of machine M are stored. The state $q_i$ *is represented by* i *o's.*

***Q. (iii)* Halting Problem by Turing Machine:**

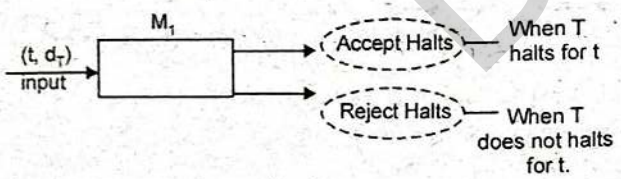
**Ans. Halting Problem:** To state halting problem we will consider the given configuration of a turing machine.

The output of TM can be:

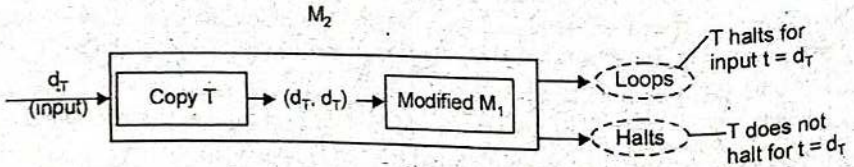
• **Halt:** The machine starting at this configuration will halt after a finite number of states.

• **No halt:** The machine starting at this configuration never reaches a halt state, no matter how long it runs.

Now the question arises based on these two observations: Given any functional matrix, input data tape and initial configuration, then is it possible to determine whether the process will ever halt? This is called halting problem. That means we are asking for a procedure which enable us to solve the halting problem for every pair (machine, tape). The answer is "no". That is the halting problem is unsolvable. Now we will prove why it is unsolvable. Let,, there existis a TM, M, which decides whether or not any computation by a TM T will ever halt when a description $d_T$ of T and tape t of T is given [That means the input to machine M, will be (machine, tape) pair]. Then for every input (t, $d_T$) to M, if T halt for input t, M, also halts which is called accept halt. Similarly if T does not halt for input t then the M, will halt which is called reject halt. This is shown in given figure.




It first copies $d_T$ and duplicates $d_T$ on its tape and then this duplicated tape information is given as input to machine $M_1$,. But machine $M_1$, is a modified M/C with the modification that whenever $M_1$ is supposed to reach an accept halt, $M_2$ loops forever.




# END TERM EXAMINATION [MAY-2015]
# FOURTH SEMESTER [B. TECH]
# THEORY OF COMPUTATION [ETCS-206]

**Time. 3 Hours** **MM:75**

*Note: Attempt any five questions including Q.No.1 which is compulsory.*

**Q.1. Differentiate between** **(5 × 5 = 25)**

**Q.1. (*a*) Deterministic PDA and Non-deterministic PDA.**

**Ans. NDPA:**

(*i*) NDPA is standard PDA used in automata theory.

(*ii*) Every PDA is NPDA unless otherwise specified.

**DPDA:**

(*i*) The standard PDA in the practical situation is DPDA.

(*ii*) The PDA is deterministic in the scnse, that at most one move is possible from any ID.

**Q.1. (*b*) Context free grammer and context sensitive grammer.**

**Ans. Context Sensitive Grammar:** *P* is the set of rules called productions defined as

$$\alpha \rightarrow \beta$$

where β is at least as long as α that is clearly

$$|\alpha| \leq |\beta|$$

The term "Context-Sensitive" comes from a normal form for these grammars, where each production is of form $\alpha_1, A\,\alpha_2 \rightarrow \alpha_1\,\beta\,\alpha_2$, with $\beta \neq \in$. Replacement of variable A by string β is permitted in the "context" of $\alpha_1$ and $\alpha_2$.

**Context Free Grammars:** G is context-free and all production in P have the form:

$$\alpha \rightarrow \beta$$

where $\alpha \in V$ and $\beta \in (V \cup NT)^*$

It means L.H.S should contain only one variable.

**Q.1. (*c*) Push down Automata and Turing Machine.**

**Ans:**

| PDA | Turing machine |
|---|---|
| (*i*) PDA can only access stored data in a last-in first–out sequence. | (*i*) Turing machine can scan through memory arbitrarily. |
| (*ii*) A PDA cannot recognise a Language $L = \{a^n b^n c^n / n \geq 1\}$ | (*ii*) Turing machine can recognise a langage $L = \{a^n b^n c^n / n \geq 1\}$ |
| (*iii*) PDA has only one direction. | (*iii*) TM has two directions either it can move left of Right |
| (*iv*) PDA cannot able to perform write operation on input tape. | (*iv*) TM has able to perform write operation on input tape. |

**Q.1. (*d*) NP complete and NP hard Problems.**

**Ans. NP Complete:** It is a complexity class which representes the set of all problems x in NP for which it is possible to reduce any other NP probelm y to x in polynomial time.

**NP- Hard:** These are the problems that are at least as hard as the NP–complete

problems: Note that NP- hard problems do not have to be in NP, and they do not have to be desision problems.

**Q.1. (e) Syntatic Grammer and Semantic Grammer.**

**Ans.** Syntax is structure and semantics is meaning. In some arbitary simple Language the statement:

$$\text{int } i = \text{"hello"}$$

is syntactically correct, but not semantically correct since it has no meaning even though it correctly follows the structure of the Language.

In coding,

$$\text{"}i\text{ (int)} = 3\text{"}$$

is semantically correct, even though its syntactically correct. There;s really no meaning in this distiriction either. generally a statement has to be syntactically valid before it even has a chance of being semantically valid.

**Q.2. (a) Describe the equivalence of PDA and CFG. Explain your answer using an example.** **(6.25)**

**Ans.** Define PDA A as follows:

$$A = (\{q\}, \{0, 1\}, \{S, B, 0, 1\}, \delta, q, s, \phi)$$

$\delta$ is defined by the following rules:

$R_1$ : $\delta(q, \wedge, S) = \{(q, 0\ BB)\}$

$R_2$ : $\delta(q, \wedge, B) = \{(q, 0S), (q, 0S), (q, 0)\}$

$R_3$ : $\delta(q_0, 0, 0) = \{(q, \wedge)\}$

$R_4$ : $\delta(q_1, 1, 1) = \{(q, \wedge)\}$

$(q_1, 010^4, S)$

$\vdash (q, 010^4, 0BB)$ by Rule $R_1$

$\vdash (q, 10^4, BB)$ by Rule $R_3$

$\vdash (q_1\ 10^4, ISB)$ by Rule $R_2$

$\vdash (q, 0^4, SB)$ by Rule $R_4$

$\vdash (q, 0^4, 0BBB)$ bby Rule $R_1$

$\vdash (q, 0^3, BBB)$ by Rule $R_3$

$\overset{*}{\vdash} (q, 0^3, 000)$ by Rule $R_2$

$\overset{*}{\vdash} (q, \wedge, \wedge)$ by Rule $R_3$

Thus $010^4 \leq N(A)$

**Q.2. (b) Discuss the "Closure properties of CFL" with proof.** **(6.25)**

**Ans:** These properties are as below

(*i*) The CFL are closed under union.

(*ii*) The CFL are closed under concatenation.

(*iii*) The CFL are closed under kleen closure.

(*iv*) The CFL are not closed under intersection.

(*v*) The CFL are not closed under complement.

**CFLs are closed under Union:** We will be consider two language $L_1$ and $L_2$ which are CFL. We can give these Languages using CFG $G_1$ and $G_2$ such that $G_1 \in L_1$ and $G_2$ . $\epsilon L_2$. The $G_1$ can be given as $G_1 = \{V_1, \Sigma, P, S,\}$ where $P_1$ can be given as

$$P_1 = \{S_1 \to A_1, S_1, A_1, /B_1, S_1\ B_1/\epsilon$$
$$A_1 \to a$$
$$B_1 \to b$$



Here $V_1 = \{S_1, A_1, B_1,\}$ and $S_1$ is a start symbol. similarly, we can write $G_2 = \{V_2, \Sigma, P_2, S_2\}$

$$N_2 = \{S_2, A_2, B_2\} \text{ and } S_2 \text{ is a start symbol}$$

$P_2$ can be given as:

$$P_2 = \{S_2 \to aA_2A_2/\ b\ B_2B_2\}$$
$$A_2 \to b$$
$$B_2 \to a\}$$

Now $L = L_1\ UL_2$ gives $G \in L$. This $G$ can be written as

$$G = \{V, \Sigma, P, S\}$$
$$V = \{S_1, A_1, B_1, S_2, A_2, B_2\}$$
$$P = \{P_1\ U\ P_2\}$$

$S$ is a start symbol

$$P = \{S \to S_1/S_2\}$$
$$S_1 \to A_1, S_1, A_1, /B_1, S_1, B_1, /\epsilon$$
$$A_1 \to a$$
$$B_1 \to b$$
$$S_2 \to a\ A_2A_2/\ b\ B_2B_2$$
$$A_2 \to b$$
$$B_2 \to a$$

Thus grammer $G$ is a CFG which produces Languages $L$ which is CFL.

Similarly, we can prove concantenation and kleen thorem.

**CFL is not closed under intersection:**

Let,

$$L_1 = \{0^n\ 1^n\ 2^i / n \geq 1, i \geq 1\}$$
$$L_2 = \{0^i\ 1^n\ 2^n / n \geq 1, i \geq 1\}$$

The grammar for $L_1$ is

$$S \to AB$$
$$A \to 0\ A\ 1 / 01$$
$$B \to 1B2\ /2$$

Similarly $L_2$ can be represented by grammer.

$$S \to AB$$
$$A \to OA/O$$
$$B \to 1\ B2/\ 12$$

Now if we try to obtain

$L = L_1 \cap L_2$ then we get sometimes CFL and sometimes non- context free languages. thus we can say that CFLS are not closed under intersection.

Similarly, we can prove CFL not closed under complementation.

**Q.3. (a) Discuss and Prove the Kleen's theorem.** **(6.25)**

**Ans: Part 1:** To each regular expression three corresponds a NFA.

**Strategy:** The corresponding NFA is the one constructed by Thompson's.

**Algorithm:** Proof that it is equivalent is by induction over regular expressions.

**Kleen's Theorem, part 2:** To each NFA there corresponds a regular expression.

**Strategy:** Proof by induction over the states of the NFA.

**Proof:** (Kleen's Theorem, part 2)

**Best Case:** (J = 1) we must be able to get from $p$ to $q$ without passing through any states; therefore either $p = q$ or we go directly in one transition from $p$ to $q$. The only strings that can be recognised in this case are $\epsilon$ and single characters from $\Sigma$. The rules $R_1$ and $R_2$ give us a regular expression corresponding to these situations.

**Inductive Step: ($j = k + 1$)**

The induction hypothesis is that for some k such that $1 \le k \le N$, the Language $L(p, q, k)$ has a corresponding regular expression. Now we must prove that, based on this assumption, $L(p, q, k + 1)$ has a corresponding regular expression. Suppose the machine $L(p, q, k + 1)$ consumes some string; we must find a regular expression corresponding to $x$. Now suppose also that it passes through state $k$ on its way (if it doesn't, then $x$ is in $L(p, q, k)$, which has a corresponding regular expression by the induction hypothesis). Since the machine can "loop" on $K$ arbitrarily many times, we can split $x$ up into three substrings:

($a$) Which moves the m/c from state $p$ to state $k$.

($b$) Which causes the m/c to "Loop" on state $k$.

($c$) Which moves the m/c from state $k$ to state $q$.

We note that while any of the above strings may cause us to start or finish at state k, none of them actually cause us to move through $k$. Thus we have:

$$a \in L(p, k, k)$$
$$b \in L(k, k, k)$$
$$c \in L(k, q, k)$$

By the induction hypothesis each of these have corresponding regular expressions, say A, B and C, respectively, and thus the regular expression for $x$ is $A\,(B^*)\,C$.

Since our choice of $x$ was arbitrary, we have proved the theorem.

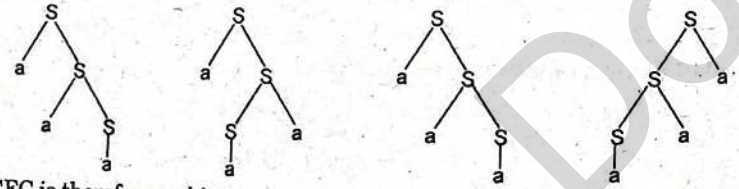
**Q.3. ($b$) Define Ambiguity in Grammer and discuss the procedure to remove the ambiguity from the grammer.** **(6.25)**

**Ans. Ambiguous Grammar:** A CFG is called ambiguous if for at least one word in the language that it generates, there are two possible derivations of the word that correspond to different syntax trees.

**For Example:**

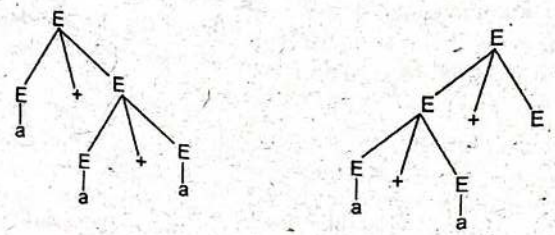
$S \rightarrow aS/Sa/a$

In this case, the word $a^3$ can be generated by four different tree's.



The CFG is therefore ambiguous.

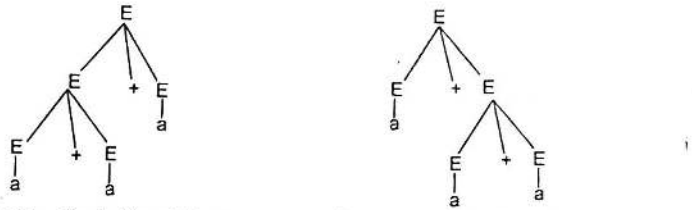
If a CFG is ambiguous, it is often possible and usually desirable to find an equivalent unambiguous CFG. Although same CFGS are "inherently ambiguous" in the sense that they cannot be produced expect by ambiguous grammars. Ambiguity is usually the property of grammar rather than the language. Let us consider the grammar for algebric expressions and for the string $a + a + a$ the two parse tree are possible.





The first cause of ambiguity in the grammar is that the precedence of the operator + and * is not respected. Both passing trees are valid. We need to force only the structure to be legal in an ambiguous grammar in which the * operator has higher proceedence than + because a + a * a is equivalent to $a + (a * a)$ and a * a is equivalent to (a * a) + a.

Again, the parse trees for a + a + a are.



The cause of ambiguity in the string a + a + a is that the associativity it of + operator is not respected. Since, we assume the + operator is left associative the string a + a + a is equivalent to (a + a) + a. Thus again we need to force only the structure in figure one to be legal in an ambiguous grammar.

**Q.4. ($a$) Give regular expression for the following:** **(6.25)**

**($i$) $L_1$ : Set of all strings of 0 and 1 ending in 00.**

**($ii$) $L_2$ : Set of all strings of 0 and 1 beginnings with o and ending with 1.**

**Ans.** ($i$) The RE has to be formed in which at the end, there should be 00. That means

$$\text{R.E} = \text{(Any combination of o's and 1's)00}$$

*i.e.* $$R.E. = (0 + 1)^* 00$$

Thus the valid string are 100, 0100, 1000, 10100..... strings ending with 00.

($ii$) The first symbol in RE should be 1 and the last symbol should be 0.

So, $$R = 1\,(0 + 1)^* 0$$

Note that the condition is strictly followed by keeping starting and ending symbols correctly. In between them there can be any combination of o and 1 including a null string.

**Q.4. ($b$) State the pumping Lemma for CFL. Provide an example to understand the Lemma concept.** **(6.25)**

**Ans.** The pumping lemma for regular sets states that every sufficiently long string in a regular set contains a short substring that can be pumped. In other words, if a long string is given and if we push or pump any no. of substrings in any number of times then we always get a regular set.

**Lemma:** let $L$ be any context Free Language, then there is a constant $n$, which depends only upon $L$, such that there exist a string $w \in L$ and $|w| \ge n$ where $w = pq\,rst$ such that

($i$) $|qs| \ge 1$

($ii$) $|prs| \le n$ and

($iii$) for all $i \ge 0$ $pq^i\,rs^i$ t is in $L$.

**Proof:** This pumping Lemma states that if there is a language L which is without unit productions and without a null production and there exist $w$ ehere $w \in L$. The string $w$ can be derived by a CFG G.

The $G$ be a grammer which is in CNF. The grammer $G$ generates Language $L$. For the string $w$, we can obtain a parse tree which derives the string $w$. Then if the Length of the path to w is less than equal to $i$ then the length of the word $w$ is less than or equal to $2^{i-1}$. We can prove this by induction step.

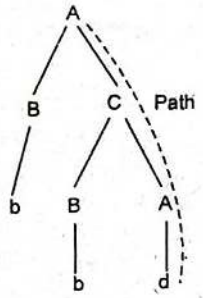
**Basis:** If $i = 1$

Let $G$ contains the rule $S \to a$ where Length of the derived string is 1 *i.e.* $i = 1$. Now according to the rule the word length should be $\leq 2^{i-1}$ *i.e* $2^0 = 1$. Observe that we have a word '$a$' which is of Length 1. Also observe that the grammer $G$ is in CNF. The Language is regular since $|w| = |pq\ rst| = 1$.

**Induction Step:** Let $w$ be a string which is derived by grammer $G$. Let $K$ be a variable such that $n = 2^k$, $|w| \geq n$ then $|w| > 2^{k-1}$ while deriving $w$ string we may get some non-terminals of CFG-G can be repeated for any number of times and will give the string w. If we pump the substrings to $w$ such that the path length of this newly formed string $w'$ ($w$ + pumped string = $w'$) is $i$ and the word length of $w'$ is $\leq 2^{i-1}$ then the grammer $G$ deriving $w'$ is called regular grammer. The necessary condition is that grammer is in chomsky's Normal Form.

Let us consider a grammer

$$G = (\{A, B, C\}, \{a\}, \{A \to BC, B \to BA, C \to BA, A \to a, B \to a, X\}, A)$$




Thus $A \overset{*}{\Rightarrow} bba = w$

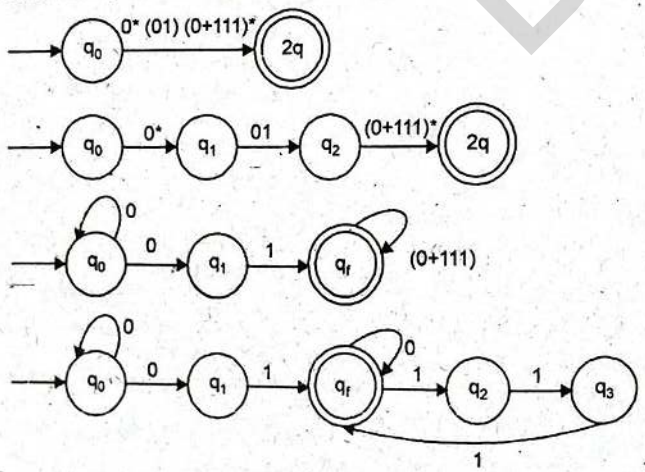
*i.e.* Path Length $i = 3$

$|w| \leq 2^{i-1}$ *i.e.* $3 \leq 2^2$

If we pump a substring into $w$ which satisfies the condition as $i \leq |w| \leq 2^{i-1} \leq n$ the grammer producing string $w$ is a regular grammer.

**Q.5. (*a*) Construct a minimal DFA that can be derived from the following regular expression o* (01) (0/111)*.** **(6)**

**Ans.** First we will construct the transition diagram for given regular expression.






Now we have got NFA without $\epsilon$. Now we will convere it to required DFA for that, we that, we will first write a transition table for this NFA.

| State \ Input | 0 | 1 |
|---|---|---|
| $\to q_0$ | $\{q_0, q\}$ | — |
| $q_1$ | — | $q_f$ |
| $q_f$ | $q_f$ | $q_2$ |
| $q_2$ | — | $q_3$ |
| $q_3$ | — | $q_3$ |

**The equivalent DFA will be**

| State \ Input | 0 | 1 |
|---|---|---|
| $\to [q_0]$ | $[q_0, q_1]$ | $[\Phi]$ |
| $[q_0, q_1]$ | $[q_0, q_1]$ | $[q_f]$ |
| $(q_f)$ | $[q_f]$ | $[q_2]$ |
| $[q_2]$ | $[\Phi]$ | $[q_3]$ |
| $[q_3]$ | $[\Phi]$ | $[q_f]$ |
| $[\Phi]$ | $[\Phi]$ | $[\Phi]$ |

**Minimize DFA:-**

| State \ Input | 0 | 1 |
|---|---|---|
| $\to [q_0, q_1]$ | $[q_0, q_1]$ | $[q_f]$ |
| $(q_f)$ | $[q_f]$ | $[q_2]$ |
| $[q_2]$ | $[\Phi]$ | $[q_3]$ |
| $[q_3]$ | $[\Phi]$ | $[q_f]$ |
| $[\Phi]$ | $[\Phi]$ | $[\Phi]$ |

**Q.5. (*b*) State and prove cook's theorem.** **(6.5)**

**Ans. Defination:** The satisfiability problem (SAT) is the problem: Given a boolean expression, is it satisfiable?

**Theorem:** SAT is NP-Complete.

**Proof: PART I:** SAT ε NP.

If the encoded expression E is of Length n, then the number of variables is [n/2]. Hence, for guessing a truth assignment t we can use multitape TM for E. The time taken by a multitape NTM M is 0(n). The M evaluates the value of E for a truth assignment t.

This is done in $O(n^2)$ time. An equivalent single tape TM takes $O(n^4)$ time. Once an accepting truth assignment is found, M accepts E and M and halts. Thus we have found a polynomials time NTM for SAT. Hence SAT $\varepsilon$ NP.

**PART II:** POLYNOMIAL-TIME REDUCTION OF ANY L IN NP TO SAT.

**1. Construction of NTM for L:** Let L be any language in NP. Then there exists a single-tape NTM M and a polynomial p(n) such that the time taken by M for an input of length n is at most p(n) along any branch. If M accepts an input w and $|w| = n$, then there exists a sequence of moves of M such that

- $\alpha_0$ is the initial ID of M with input w.
- $\alpha_0 \vdash \alpha_1 \vdash \ldots \vdash \alpha_k$, $k \le P(n)$
- $\alpha_k$ is an ID with an accepting state.

**2. Representation of sequence of moves of M:** As the maximum number of steps on w is p(n) we need not bother about the contents beyond p(n) cells. We can write $\alpha_i$ as a sequence of $p(n) + 1$ symbols (one symbol for the state and the remaining symbols for the tape symbols. So $a_i = x_{i_0} x_{i_1} \ldots x_{ip(n)}$. If $\alpha_m$ is an accepting ID in the course of processing $w$ then we write $\alpha_0 \vdash \ldots \vdash \alpha a \vdash \alpha_m = \alpha_{i, p(n)}$.

**3. Representation of IDs in terms of Boolean Variables:** We define a boolean variable $y_{ijA}$ corresponding to $(i, j)$th entry in the $i$th ID. The variable $y_{ijA}$ represents the proposition that $x_{ij} = A$, where $A$ is a state or tape symbol and $0 \le i, j \le p(n)$.

**4. Polynomial Reduction of M to SAT:** In order to check that the reduction of M to SAT is correct, we have to ensure the correctness of

($a$) the initial ID.

($b$) the accepting ID and

($c$) the intermediate moves between successive IDs.

**($i$) Simulation of initial ID:** $x_{00}$ must start with the initial state $q_0$ of $M$ followed by the symbols of $w = a_1 a_2 \ldots a_n$ of length $n$ and ending with b's. The corresponding boolean expression S is defined as:

$$S = y_{00q_0} \wedge y_{01a_1} \wedge \ldots \wedge y_{0n\,a_n} \wedge y_{0n\,a_{n+1}} \ldots \wedge y_{0,p(n),\#}$$

**($ii$) Simulation of accepting ID:** $\alpha_{p(n)}$ is the accepting ID. If $p_1, p_2, \ldots p_k$ are the accepting states of $M$, then $\alpha_{p(n)}$ contains one of $p_i$'s $1 \le i \le k$ in any place j.

It is given by

$$F = F_0 \vee \ldots \vee F_{p(n)}$$

where $$F_j = y_{p(n),j,p_1} \vee y_{p(n),j,p2} \vee \ldots \vee y_{p(n),n,pk}$$

**($iii$) Simulation of Intermediate Moves:** We have to simulate valid moves $\alpha_i \vdash \alpha_{i+1}$, $i = 0, 1, 2 \ldots p(n)$. corresponding to each move, we have to define a boolean variable $N_i$. Hence the entire sequence of IDs leading to acceptance of w is.

$$N = N_0 \wedge N_1 \wedge \ldots \wedge N_{p(n)-1}$$

**Formulation of $B_{ij}$:** When the state of $\alpha_i$ is none of $x_{i,j-1}$, $x_{ij}$, $x_{i,j+1}$, then the transition corresponding to $\alpha_{i+1}$ will not affect $x_{i,j+1}$. In this case $x_{i+1,j} = x_{ij}$.

- **Formulation of $A_{ij}$:** This step corresponds to the correctness of the $2 \times 3$ array

| | | |
|---|---|---|
| $x_{i,j-1}$ | $x_{ij}$ | $x_{i,j+1}$ |
| $x_{i+1,j-1}$ | $x_{i+1,j}$ | $x_{i+1,j+1}$ |



**Defination of $N_i$ and N:** We define $N_i$ and N by

$$N_i = (A_{io} \vee B_{io}) \wedge (A_{i1} \vee B_{i1}) \wedge \ldots \wedge (A_{i,\,p(n)} \vee Bi,\, p(n))$$

$$N = N_0 \wedge N_1 \wedge N_2 \wedge \ldots \wedge N_{p(n)-1}$$

**5. Completion of Proof:** Let $E_{m,w} = S \wedge N \wedge F$

We have seen that the time taken to write S and F are $0(p(n))$ and the time taken for N is $0\,(P^2(n))$. Hence the time taken to write $E_{M,w}$ its $(p^2(n))$.

Also M accepts w if and only if $E_{M,w}$ is satisfiable Hence the determinsitic multiple TM $M_1$ can convert W to a boolean expression $E_{M,W}$ in $0(p^2(n))$ time. An equivalent single tape TM taken $0\,(p^4(n))$ time. This proves the Part II, of the cook's theorem, thus completing the proof of this theorem.

**Q.6. ($a$) Construct the PDA accepting the Language $\{(ab)^n / n \ge 1\}$ by empty stack.** **(6)**

**Ans.** $L = \{(ab)^n / n \ge 1\}$

This is a Language in which alternate's a's and b's. The instaneous description can be gives as:-

$$\delta(q_o\, a, z_o) = (q_1, a\, z_o)$$
$$\delta(q_1, b, a) = (q_2, \in)$$
$$\delta(q_2, a, z_o) = (q_3, az_o)$$
$$\delta(q_3, b, a) = (q_2, \in)$$
$$\delta(q_2, \in, z_o) = (q_f \in)$$

where $q_o$ is a start state and $q_f$ is accept state. We will simulate this PDA for following string–

$$(q_o, \text{abab}, z_o) \vdash (q_1, bab, az_o)$$
$$\vdash (q_2, ab, z_o)$$
$$\vdash (q_3, b, az_o)$$
$$\vdash (q_2, \in, z_o)$$
$$\vdash (q_f, \in)$$

ACCEPT State

**Q.6. ($b$) Explain Melay and Moore M/C in brief.** **(6.5)**

**Ans.** ($i$) A special case of FA is Moore machine in which the output depends on the state of the machine.

($ii$) An automaton in which the output depends on the transition and input is called Melay machine.

**Example:** Construct a Melay Machine equivalent to the Moore machine given by the following table:

| Present State | Next State | | Output |
|---|---|---|---|
| | $a = 0$ | $a = 1$ | |
| $q_0$ | $q_3$ | $q_1$ | 0 |
| $q_1$ | $q_1$ | $q_2$ | 1 |
| $q_2$ | $q_2$ | $q_3$ | 0 |
| $q_3$ | $q_3$ | $q_0$ | 0 |

**Solution:** We must follow the reverse procedure of converting a Melay machine into a Moore Machine. In the case of Moore Machine, for every input symbol we form the pair

consisting of the next state and the corresponding output and reconstruct the table for the Melay machine. For example, the states $q_3$ and $q_1$ in the next state column should be associated with outputs 0 and 1, respectively. The transition table for the Melay machine is given by:

| Present State | Next State | | | |
|---|---|---|---|---|
| | $a = 0$ | | $a = 1$ | |
| | State | Output | State | Output |
| $q_0$ | $q_3$ | 0 | $q_1$ | 1 |
| $q_1$ | $q_1$ | 1 | $q_2$ | 0 |
| $q_2$ | $q_2$ | 0 | $q_3$ | 0 |
| $q_3$ | $q_3$ | 0 | $q_0$ | 0 |

**Q.7. (a) Prove that the halting problem is undecidable.** **(6)**

**Ans.** Refer to Q.No. (4) (b) (iii) Second Term Examination, April 2015.

**Q.7. (b) State and prove savitch Theorem.** **(6.5)**

**Ans. Savitch's Theorem:** Savitch theorem gives a relationship between determinstic and non-deterministic space complexity. It states that for any function.

$$f \in \Omega (\log (n),$$
$$\text{NSPACE}(f(n)) \subseteq \text{DSPACE}((f(n))^2)$$

In other way, if a non-deterministic Turing machine can solve a probelm using $f(n)$ space, an ordinary deterministic Turing machine can solve the same problem in the square of that space bound.

**Proof:** There is a proof of the theorem that is constructive; is demonstrates an algorithm for STCON, the problem of determining whether there is a path between two vertices in a directed graph, which runs in $0(\log n)^2)$.

Space for $n$ vertices. S TCON can be solved from this problem by setting $k$ to $n$. To test for a k edge path from, one way test whether each vertex $u$ may be the midpoint of the path by recursively searching for path of half the length from $s$ to $u$ and $u$ to $t$.

**Code:**

```
def k - edge - path (s, t, k):
if                    k = =0
return                s = =t
if                    k = =1
return                s = =t or (s, t)
in edges
for u in vertices
if k- edge- path (s, 4, floor (k/2)) and k- edge-path return true return false.
                      (u, t, ceil) (k/2)):
return true
return false.
```

**Q.8. Write short note on any two:** **(6.25 × 2 = 12.5)**

**Q.8. (a) Co-NL complexity classes.**

**Ans.** In computational complexity theory, NL-complete is a compexity class containing the language that are complete for NL, the class of decision problems that can be solved by non-determinstic Turing machine using a logarithmic amount of memory



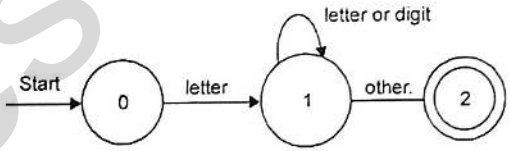
space. The NL-complete language are the most "difficult" or "expressive" probelms in NL. If a method exist for solving any one of the NL- complete problems in logarithmic memory space, then NL = L

NL machine of the decision problem that can be solved by a non deterministic Turing machine with a read only input tape and a separate read-write tape whose size is limited to be proportional to the logarithm of the input length.

**Q.8. (b) Write short note on LEX and VACC**

**Ans. LEX:** Lex is a program that generates lexical analyser. It is mostly used with Yacc parser generator. It reads the input stream and output source code implementing the Lexical analyzer in the c-programming language. Lex will read pattern (regular expression (then produces) c code for a lexical analyser that scan for identifiers.

Refular expression are translated by lex to a computer program that mimics an FSA. A simple pattern is letter (letter/digit)* This pattern matches string of characters that begins with a single letter followed by zero or more letter or digits.




**YAAC:** YAAC is a tool which will produce a partial for a given grammar. YACC is a program designed to compile a LALR (1) grammer and to produce the source code of a syntactic analyzer of the language produced by this grammar. Input is a grammer (rules) and actions to take upon recognizing a rule and output is a C program and optimally a header file of token.

**Input:** A CFG and a translation sechme – file. y

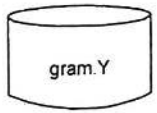
**Output:** A parser file. tab. c (bison) or y. tab. c (yacc).

(1) An output file file. Output containing the parsing table.

(2) A file file. tab. h containing declaration.

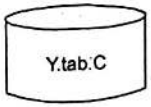
(3) The parser called yy parse ().

(4) Parser expects to use a function called yylex () to get token.

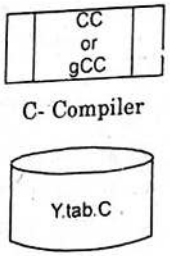



file containing desired grammer in YACC format


YACC program




C source program created by YACC




executable program that will parse grammar given in gram.y.

**Q.8. (c) Write short notes on Normal Froms for Context Free grammars?**

**Ans.** In a CFG, the R.H.S. of a production can be any string of variables and terminals. When the productions in $G$ satsify certain restrictions, then $G$ is said to be in a 'normal form'. Among several 'normal forms' we study two of them in this section — the chomsky Normal Form (CNF) and the Greibach Normal Form.

**Chomsky Normal Form:** In the CNF, we have restrictions on the length of R.H.S. and the nature of symbols in the R.H.S. of productions.

→ A CFG is in CNF if every production is of the form $A \to a$ or $A \to BC$ and $S \to \wedge$ is $G$ if $\wedge \in L(G)$. When $\wedge$ is in $L(G)$, we assume that S does not appear on the R.H.S. of any production. For example, consider $G$ whose productions are $S \to AB/\wedge, A \to a, B \to b$. Then $G$ is in CNF.

**Greibach Normal Form:** GNF is another normal form quite useful in some proofs and constructions. A CFG generating the set accepted by a push down automata is in GNF.

→ A CFG is in GNF if every production is of the form $A \to a\alpha$, where $\alpha \in V_N^*$ and $a \in \Sigma$ ($\alpha$ may be $\wedge$), and $S \to \wedge$ is $G$ if $\wedge \in L(G)$. When $\wedge \in L(h)$ we assume that $S$ does not appear on the R.H.S of any production.

For example, $G$ given by $S \to a\,AB/\wedge, A \to bc, B \to b, C \to c$ is in GNF.

# FIRST TERM EXAMINATION [FEB. 2016]
# FOURTH SEMESTER [B.TECH]
# THEORY OF COMPUTATION [ETCS–206]

**Time : 1.5 hrs.** **M.M. : 30**

**Note:** *Attempt any three questions including Q.No.1, which is compulsory.*

**Q.1. (a) Define the following terms with example:**

**(a) Input Alphabet** **(b) Automata**

**(c) Language** **(d) Grammar**

**Ans: (a) Input Alphabet**: A non-empty set is called alphabet,when it is used in string operation. It's members are then commonly known as symbols or letters.eg. Characters and Digits.

"Alphabets are defined as finite set of symbols"

It is a set of decimal numbers , $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

**(b) Automata**: An automata is defined as a system where energy, material, information are transformed, transmitted and used for performing some function without direct participation of man.

**OR**

An automata is a mathematical object that takes a word as input and decides either to accept it or reject it. Since all computational problems are reducible into the accept/reject question onwords, automata theory plays a crucial role in computational theory.

**For e.g.** Automatic machine tool, Automatic packaging machine, Automatic photo printing machine.

**(c) Language**: A language is a collection of sentences of finite length all constructed from a finite alphabet of symbols.

***OR***

L is a said to be a language over alphabet $\Sigma$, only if $L \subseteq \Sigma^*$ this is because $\Sigma^*$ is the set of all strings (of all possible length including 0) over the given alphabet $\Sigma$.

For e.g. set of binary numbers whose value is prime: {10,11,101,111,1011}

**(d) Grammars**: A grammar G can be formally written as a 4-tuple (N, T, S, P) where

- N or VN is a set of Non-terminal symbols
- T or $\Sigma$ is a set of Terminal symbols
- S is the Start symbol, $S \in N$
- P is Production rules for Terminals and Non-terminals

For e.g. –

({S, A, B}, {a, b}, S, {S → AB, A → a, B → b})

Here, S, A, and B are Non-terminal symbols;

a and b are Terminal symbols.

S is the Start symbol, $S \in N$.Productions, P : S → AB, A → a, B → b

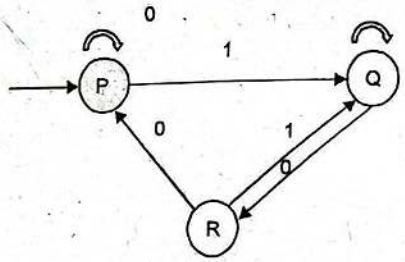
**Q.2.(a)Differentiate between Deterministic Finite Automata and Non-Deterministic Finite Automata.**

**Ans.**

| Deterministic Finite Automata | Non-Deterministic Finite Automata |
|---|---|
| 1. Transition function ($\delta : Q \times S \to Q$) | Transition function $\Delta : Q \times \Sigma \to P(Q)$. |
| 2. DFA cannot use empty string transition | NFA can use empty string transition |
| 3. DFA requires more space | NFA requires less space. |

| | |
|---|---|
| 4. Backtracking is allowed in DFA | In NFA it may or may not be allowed |
| 5. DFA will reject the string if it end at other than accepting state. | If all of the branches of NFA dies or rejects the string, we can say that NFA reject the string. |

**Q.2. (*b*) Construct a regular expression from the given FA(where P is the final state) by using Arden's Theorem.**




**Ans:** We can directly apply the Arden's method, since graph doesn't contain any move & there is only one initial state.

Equations for P, Q, R can be written as:

P = € + P.0 + R.0----------(1)

R = Q.0 ----------(2)

Q = Q.1 + R.1 + P.1---------- (3)

Put value of R from eq. 2 to eq. 3

Q = Q.1 + Q.0.1 + P.1

Q = Q(1 + 0.1) + P.1

Apply the theorem result ( Q + RP=QP*)

Q = P.1(1+0.1)*----------(4)

Put value of R from eq. 2 to eq. 1

P = € + P.0 + Q.0.0----------(5)

Put value of Q from eq. 4 to eq. 5

P = € + P.0 +0.0.P.1(1 +0.1)*

P = € + P ((0 + 0.0.1(1 + 0.1)*)

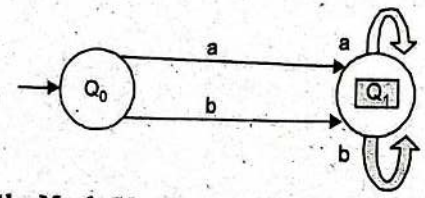
Apply the theorem result ( Q + RP=QP*)

P = € ((0 + 0.0.1(1 + 0.1)*)*

P = ((0 + 0.0.1(1 + 0.1)*)*

**Q.3.(*a*) Draw a DFA for the regular expression aa*/bb*.**

**Ans:**




**Q.3. (*b*) Explain the Mealy Machine and Moore Machine with example.**

**Ans:** Finite automata may have outputs corresponding to each transition. There are two types of finite state machines that generate output–

1. Mealy Machine
2. Moore Machine

**Mealy Machine:** A Mealy Machine is an FSM whose output depends on the present state as well as the present input.



It can be described by a 6 tuple (Q, Σ, O, δ, X, $q_0$) where–

**Q** is a finite set of states.

**Σ** is a finite set of symbols called the input alphabet.

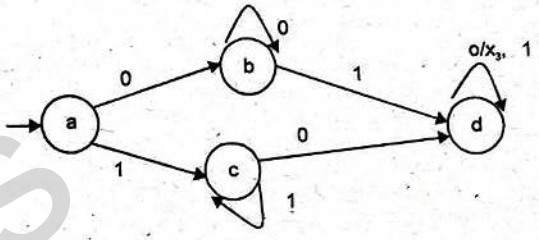
**O** is a finite set of symbols called the output alphabet.

**δ** is the input transition function where δ: Q × Σ → Q

**X** is the output transition function where X: Q → O

$q_0$ is the initial state from where any input is processed ($q_0 \in Q$).

The state diagram of a Mealy Machine is shown below –




**Moore Machine**

Moore machine is an FSM whose outputs depend on only the present state.

A Moore machine can be described by a 6 tuple (Q, Σ, O, δ, X, $q_0$) where –

**Q** is a finite set of states.

**Σ** is a finite set of symbols called the input alphabet.

**O** is a finite set of symbols called the output alphabet.

**δ** is the input transition function where δ: Q × Σ → Q

**X** is the output transition function where X: Q × Σ → O

$q_0$ is the initial state from where any input is processed ($q_0 \in Q$).

The state diagram of a Moore Machine is shown Fig.

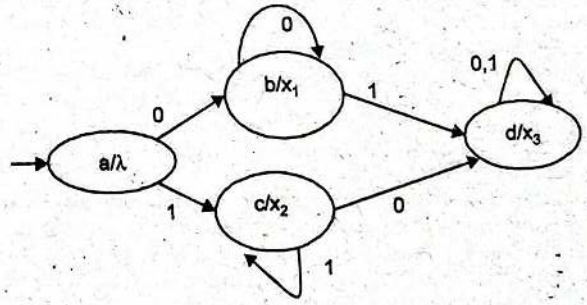



**Table format for both machines:**

**Moore Machine**

| Present state | Next state | | Output |
|---|---|---|---|
| | a = 0 | a = 1 | |
| → $q_0$ | $q_3$ | $q_3$ | 0 |
| $q_1$ | $q_1$ | $q_2$ | 1 |
| $q_2$ | $q_2$ | $q_3$ | 0 |
| $q_3$ | $q_3$ | $q_3$ | 0 |

**Mealy Machine**

| Present state | Next state | | | |
|---|---|---|---|---|
| | a = 0 | | a = 1 | |
| | state | output | state | output |
| →$q_0$ | $q_3$ | 0 | $q_1$ | 1 |
| $q_1$ | $q_1$ | 1 | $q_2$ | 0 |
| $q_2$ | $q_2$ | 0 | $q_3$ | 0 |
| $q_3$ | $q_3$ | 0 | $q_3$ | 0 |

**Q.4. (*a*) Define regular expression. Explain the four characteristics that are used to simplify the regular expression.**

**Ans:** As finite automata are used to *recognize* patterns of strings, regular expressions are used to *generate* patterns of strings. A regular expression is an algebraic formula whose value is a pattern consisting of a set of strings, called the language of the expression

***OR***

A regular expression is an expression that describes a whole set of strings, by following certain syntax rules. It is used by many Text editors and utilities to search a block of text for certain patterns.

For e.g . The set of strings over {0,1} that end in 3 consecutive 1's. (0 | 1)* 111

**Four characteristics that are used to simplify the regular expression:**

If r and s are regular expressions denoting the languages L(r) and L(s), then

1. Union : (r) | (s) is a regular expression denoting L(r) ∪ L(s).
2. Concatenation : (r)(s) is a regular expression denoting L(r)L(s).
3. Kleene closure : (r)* is a regular expression denoting (L(r))*(r) is a regular expression denoting L(r).
4. Positive Closure: It is denoted by $L^+$ represent the set of those strings that can be formed by taking any no. of strings from L, $L^+ = L^* - \epsilon$

**Q.4. (*b*) State the Pumping Lemma for regular grammar.**

**Ans:** Pumping Lemma is to be applied to show that certain languages are not regular. It should never be used to show a language is regular.

**Statement:** Let M = (Q, Σ, δ, $q_0$, F) be a finite automata with 'n'states. Let L be the regular set accepted by M. Let w ∈ L and | w | >= m.If m> = n, then there exist x, y, z such that w= xyz, y !=^ and $xy^iz$ belongs to L for each i>= 0.

**Proof:** Find a non-empty string ý'not too far from the beginning of w that can be pumped *i.e.* repeatingý'any no. of times, keeps the resulting string in the language.

Let w = $a_1, a_2, a_3$, -------- $a_m$ m> = n is string.

δ ($q_0, a_1, a_2, a_3$, ---------$a_i$) = $q_i$ for i = 1, 2 —— m

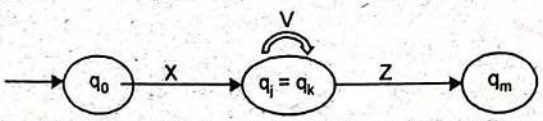
$Q_1$ = {$q_0 q_1 q_2$.....$q_m$ }

*i.e.* $Q_1$ = Sequence of states in the path with path value w = $a_1,a_2,a_3$,----------$a_m$ as there are only 'n'distinct states, at least two states in $Q_1$must coincide. So, among the various similar states, we take the first pair of states. Let us take them $q_j,q_k$

($q_j = q_k$). Then $j$ & $k$ satisfy the condition $0 <= j < k <= n$. The string 'w' can be decomposed in to 3 substrings.

$a_1, a_2, a_3$,--------$a_j$, $a_{j+1}$,---------$a_k$ ,$a_{k+1}$,--------- $a_m$. Let x, y, z denote the strings respectively.

As $k <= n$, | xy | <= $n$ & $w$ = xyz, so the transition diagram




Automata 'M' starts at the initial state $q_0$. Only applying 'x'it reaches $q_j$ ( = $q_k$),On applying ý it comes back to $q_j$ ( = $q_k$) so after application of $y^i$ for each $i$ > = 0,the automation is in the same state $q_j$, on applying z, it reaches $q_m$, a final state.

Hence $xy^iz$ belongs to $L$.

# END TERM EXAMINATION [MAY–2016]
# FOURTH SEMESTER [B.TECH]
# THEORY OF COMPUTATION [ETCS–206].

**Time : 3 hrs.** **M.M. : 75**

**Note:** *Attempt any five questions including Q.No.1, which is compulsory.*

**Q.1. (*a*) Differentiate between DFA and NFA.** **(5×5=25)**

**Ans.** Refer to Q.2(*a*) First Term Examination 2016.

**Q.1. (*b*)What is Ambiguity? How it is removed?**

**Ans**: If a context free grammar G has more than one derivation tree for string w∈ **L(G)**, it is called an **ambiguous grammar**. There exist multiple right-most or left-most derivations for some string generated from that grammar.

**Problem**

To check whether the grammar G with production rules –

$X \rightarrow X+X \mid X*X \mid X \mid$ a is ambiguous or not.

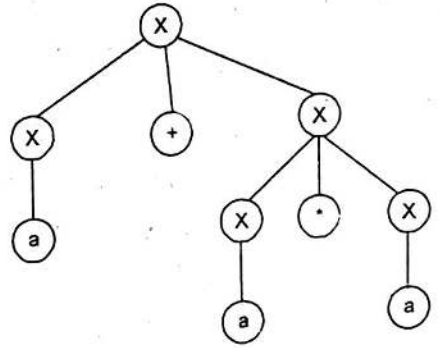
**Solution.**

Let's find out the derivation tree for the string "$a + a* a$". It has two leftmost derivations.

**Derivation 1 –**

$X \rightarrow X+X \rightarrow a+X \rightarrow a+X*X \rightarrow a+a*X \rightarrow a+a*a$

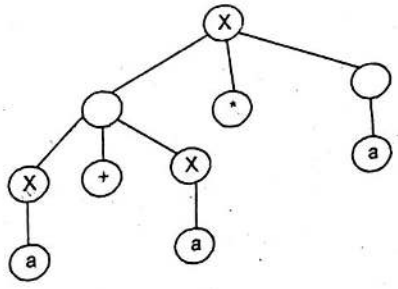
**Parse tree 1–**




**Derivation 2 –**

$X \rightarrow X*X \rightarrow X+X*X \rightarrow a+X*X \rightarrow a+a*X \rightarrow a+a*a$

**Parse tree 2–**



As there are two parse trees for a single string "$a + a*a$", the grammar $G$ is ambiguous.

**Removal of Ambiguity:**

1. Precedence of operators is not respected.

2. Sequence of identical operators can group either from left or from right .We would see two different parse tree for the above expression. Since addition is associative, it doesn't matter whether the group is from left or right, but to eliminate the ambiguity we must take one.

**Q.1. (c) Define Recursively Enumerable Language. What are its different properties?**

**Ans:** A recursively enumerable language is a formal language for which there exists a Turing machine (or other computable function) that will halt and accept when presented with any string in the language as input but may either halt and reject or loop forever with any string in the language not in the language. Contrast this to recursive languages, which require that the Turing machine halts in all cases. *e.g.* Halting Problem, Post correspondence problem.

**Different properties of Recursively Enumerable Language:**

1. If L, L 1 and L 2 are recursive languages, then so are L 1 ∩ L 2, L 1 L 2, L *, L 1) ∪ L 2 and L 1 – L 2.

2. If L, L 1 and L 2 are recursively enumerable languages, then so are L 1 ∪ L 2, L1 L 2, L * and L 1 ∩ L2.

3. If Σ is an alphabet, L ⊆ Σ * is a recursively enumerable language, and Σ* – L is recursively enumerable, then L is recursive.

4. If Σ is an alphabet, L ⊆ Σ * is a recursively enumerable language, and Σ * – L is recursively enumerable, then L is recursive.

**Q.1. (d) Differentiate between NP –Hard and Np-Complete Problem.**

**Ans:** A problem **H is NP-hard** if and only if there is an NP-complete problem L that is polynomial time Turing-reducible to H (i.e., L ≤ TH).

A problem is **NP-complete** if any problem in NP can be reduced to it in polynomial time AND it is also in NP (and thus solutions can be verified in polynomial time)

All NP-Complete problems are NP-Hard. However, not all NP-Hard problems are NP-Complete. For a language L to be NP-Complete, it must be the case that:

1. L is in NP 2. L is NP-Hard

In order for a language to be NP-Hard, it means that if you can solve it in polynomial-time, you can solve any problem in NP in polynomial time.

An example of a language L being NP-Hard but (most likely) not NP-Complete is the language that determines, given a 3-SAT boolean formula f and an integer k, are there k or more satisfying assignments. By letting k = 1, this is exactly the 3-SAT problem, so therefore, this qualifies as NP-Hard.

**Q.1. (e) Differentiate between Moore and Mealy machine?**

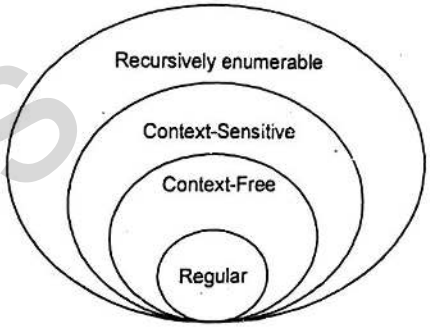
**Ans.** Refer to Q.3(*b*) of First Term Examination 2016.

**Q.2. (a) Briefly Explain Chomsky Classification of languages with examples.**

**Ans:** According to Noam Chomsky, there are four types of grammars " Type 0, Type1, Type 2, and Type 3. The following table shows how they differ from each other "



| Grammar Type | Grammar Accepted | Language Accepted | Automaton |
|---|---|---|---|
| Type 0 | Unrestricted grammar | Recursively enumerable language | Turing Machine |
| Type 1 | Context-sensitive grammar | Context-sensitive language | Linear- bounded automaton |
| Type 2 | Context-free grammar | Context-free language | Pushdown automaton |
| Type 3 | Regular grammar | Regular language | Finite state automaton |




**Type - 3 Grammar**

**Type-3 grammars** generate regular languages. Type-3 grammars must have a single non-terminal on the left-hand side and a right-hand side consisting of a single terminal or single terminal followed by a single non-terminal.

The productions must be in the form $\mathbf{X \to a}$ or $\mathbf{X \to aY}$ ,where $\mathbf{X, Y \in N}$ (Non terminal) and $\mathbf{a \in T}$ (Terminal).The rule $\mathbf{S \to \varepsilon}$ is allowed if **S** does not appear on the right side of any rule. **Example**

$X \to \varepsilon$

$X \to a$

$X \to aY$

**Type - 2 Grammar**

**Type-2 grammars** generate context-free languages.

The productions must be in the form $\mathbf{A} \to \gamma$ where $\mathbf{A \in N}$ (Non terminal) and $\gamma \in (\mathbf{T \cup N})^*$ (String of terminals and non-terminals).These languages generated by these grammars are be recognized by a non-deterministic pushdown automaton.

**Example**

$S \to X\,a$

$X \to a$

$X \to aX$

$X \to abc$

$X \to \epsilon$

**Type - 1 Grammar**

**Type-1 grammars** generate context-sensitive languages. The productions must be in the form $\alpha A \beta \to \alpha \gamma \beta$ ,where $\mathbf{A \in N}$ (Non-terminal) and $\alpha, \beta, \gamma \in (\mathbf{T \cup N})^*$ (Strings of terminals and non-terminals)

The strings $\alpha$ and $\beta$ may be empty, but $\gamma$ must be non-empty.

The rule $\mathbf{S \to \epsilon}$ is allowed if **S** does not appear on the right side of any rule. The languages generated by these grammars are recognized by a linear bounded automaton.

**Example**

$AB \rightarrow AbBc$ $A \rightarrow bcA$ $B \rightarrow b$

**Type - 0 Grammar**

**Type-0 grammars** generate recursively enumerable languages. The productions have no restrictions. They are any phase structure grammar including all formal grammars.They generate the languages that are recognized by a Turing machine.

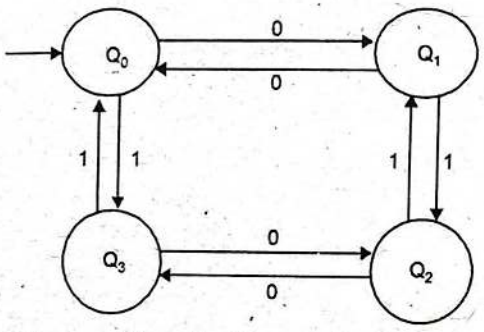
The productions can be in the form of $\alpha \rightarrow \beta$ where $\alpha$ is a string of terminals and non-terminals with at least one non-terminal and $\alpha$ cannot be null. $\beta$ is a string of terminals and non-terminals.

**Example**

$S \rightarrow ACaB$ $Bc \rightarrow acB$ $CB \rightarrow DB$ $aD \rightarrow Db$

**Q.2. (*b*) Draw a DFA for all strings over {0, 1} consisting of even no. of 0's and even no of 1's.**

**Ans.**




**Q.3. (*a*) State and prove Pumping Lemma for Regular Languages. Also prove that language L = {$a^n$ $b^n$ for n= 0, 1, 2, 3--------} is not regular.**

**Ans:** Pumping Lemma is to be applied to show that certain languages are not regular. It should never be used to show a language is regular.

**Statement:** Let M = (Q, Σ, δ $q_0$, F) be a finite automata with 'n'states. Let L be the regular set accepted by M. Let w L and | w | >= m. If m>= n,then there exist x, y, z such that $w = xyz$, $y$ ! = ^ and $xy^iz$ belongs to L for each $i >= 0$.

**Proof**: Find a non-empty string ý'not too far from the beginning of w that can be pumped i.e. repeating ý'any no of times, keeps the resulting string in the language.

Let $w = a_1, a_2, a_3, \text{----------} a_m$ $m >= n$ is a string.

$\delta(q_0 a_1, a_2, a_3, \text{---------} a_i) = q_i$ for i = 1, 2——m

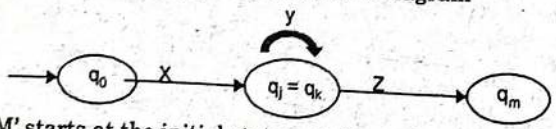
$Q_1 = \{q_0 q_1 q_2 \text{\_\_\_\_\_} q_m\}$

*i.e.* $Q_1$= Sequence of states in the path with path value $w = a_1, a_2, a_3, \text{———} a_m$ as there are only 'n'distinct states, at least two states in $Q_1$must coincide. So, among the various similar states, we take the first pair of states. Let us take them $q_j, q_k (q_j = q_k)$. Then j & k satisfy the condition $0 <= j < k <= n$.

The string 'w' can be decomposed in to 3 substrings.

$a_1, a_2, a_3, \text{———} a_j$, $a_{j+1}, \text{———} a_k$ , $a_{k+1}, \text{———} a_m$. Let $x, y, z$ denote the strings respectively.

As $k <= n$, $|xy| <= n$ & $w = xyz$, so the transition diagram




Automata 'M' starts at the initial state $q_0$.Only applying 'x' it reaches $q_j (=q_k)$, On applying ý it comes back to $q_j (=q_k)$ so after application of $y^i$ for each $i >= 0$,the automaton is in the same state $q_j$. on applying z, it reaches $q_m$, a final state.



Hence $xy^iz$ belongs to $L$.

**To prove that $L$ = {$a^n$ $b^n$ for n= 0, 1, 2, 3 ----------} the language is not regular**

According to pumping Lemma, there must be string x, xy, xz such that all the words of the form $xy^iz$ are in $L$.

**Case 1**: If middle part 'y' is made of entirely a's *i.e.* x a aaa----------z.

If we pump it as $xy^2z$, $xy^3z$ then no. of a's increase but in the given language no. of a's and b's are equal so it is not allowed.

**Case 2**: If middle part 'y' is made up of entirely b's *i.e* x b bbb--------z. for the above reason, it is not allowed.

**Case 3**: 'y' is made of some positive no. of a's and some positive number b's *i.e.* abn *i.e x* aaaabbbb——z thus xyyz would have 2 copies of the substring ab. Butenery word in L contains substring ab exactly once.

Therefore, $xyyz$ cannot be a word in $L$.

This proves that the pumping lemma cannot apply to L.Therefore,L is not regular.

**Q.3. (*b*) Find a Regular expression corresponding to each of the following subset {0, 1}:**

**(*i*). The language of all strings containing atleast two 0's**

**(*ii*). The language of all strings containing atmost two 0's**

**Ans**: (*i*) $r = (0+1)^*0\,(0+1)^*\,0\,(0+1)^*$

(*ii*) $r = 1^* + 1^*01^* + 1^*01^*01^*$

**Q.4. (*a*) Consider the CFG whose production are:**

**$S \rightarrow bB/aA$**

**$A \rightarrow b/bS/aAA$**

**$B \rightarrow a/aS/bBB$ for the string bbaababa. Find**

**(*i*) Left Most Derivation**

**(*ii*) Right Most Derivation**

**(*iii*) Parse Tree**

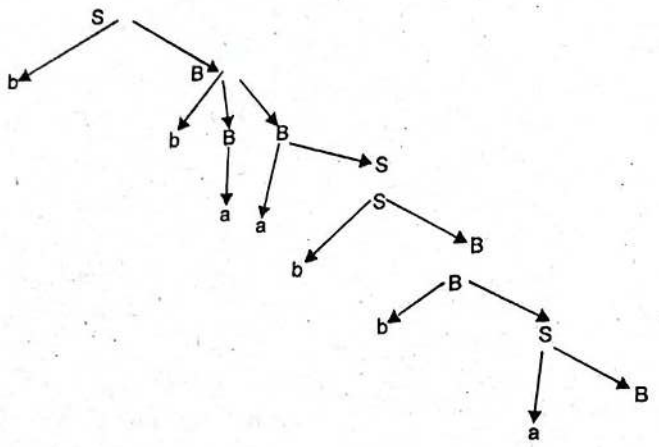
**Ans:** Left most derivation:

S = bB
= bbBB
= bbaB
= bbaaS
= bbaabB
= bbaabaS
= bbaababB
= bbaababa

**Right most derivation:**

S = bB
= bbBB
= bbBaS
= bbBabB
= bbBabaS
= bbBababB
= bbBababa
= bbaababa

**Parse Tree**




**Q.4. (b) What is PDA? Construct a PDA accepting the set of all strings over{a,b} with equal no. of a's and b's.**

**Ans:** A pushdown automaton is a way to implement a context-free grammar in a similar way we design DFA for a regular grammar. A DFA can remember a finite amount of information, but a PDA can remember an infinite amount of information.

Basically a pushdown automaton is –

**"Finite state machine" + "a stack"**

A PDA can be formally described as a 7-tuple (Q, ", S, ä, $q_0$, I, F)–

- **Q** is the finite number of states
- **Σ** is input alphabet
- **S** is stack symbols
- **δ** is the transition function – $Q \times (\Sigma \cup \{\varepsilon\}) \times S \times Q \times S^*$
- $q_0$ is the initial state ($q_0 \in Q$)
- **I** is the initial stack top symbol ($I \in S$)
- **F** is a set of accepting states ($F \in Q$).

We start by storing a symbol of the input string & continue storing until the other symbol occurs. If topmost symbol in stack is 'a' & the current input symbol is 'b','a' in stack is erased.

$M = (\{q\}, \{a, b, z_0\}, \delta, q, z_0, \phi)$

δ is defined by

$\delta(q, a, z_0) = \{(q, a, z_0)\}$

$\delta(q, b, z_0) = \{(q, b, z_0)\}$

$\delta(q, a, a) = \{(q, aa)\}$

$\delta(q, b, b) = \{(q, bb)\}$

$\delta(q, a, b) = \{(q, \in)\}$

$\delta(q, b, a) = \{(q, \in)\}$

$\delta(q, \in, z_0) = \{(q, \in)\}$

**Q.5. (a) What are the different closure properties of CFL? Explain with proof.**

**Ans. Closure Property of CFL**

Context-free languages are **closed** under –

- Union
- Concatenation
- Kleene Star operation



**Union**

Let $L_1$ and $L_2$ be two context free languages. Then $L_1 \cup L_2$ is also context free.

**Example:** Let $L_1 = \{ a^n b^n , n \geq 0\}$. Corresponding grammar $G_1$ will have P: S1 → aAb | ab

Let $L_2 = \{ c^m d^m , ne^n 0\}$. Corresponding grammar $G_2$ will have P: S2 → cBb | ε

Union of $L_1$ and $L_2$, $L = L_1 \cup L_2 = \{ a^n b^n \} \cup \{ c^m d^m \}$

The corresponding grammar G will have the additional production S → S1 | S2

**Concatenation**

If $L_1$ and $L_2$ are context free languages, then $L_1 L_2$ is also context free.

**Example:** Union of the languages $L_1$ and $L_2$, $L = L_1 L_2 = \{ a^n b^n c^m d^m \}$

The corresponding grammar G will have the additional production S → S1 S2

**KleeneStar:** If L is a context free language, then $L^*$ is also context free.

**Example:** Let $L = \{ a^n b^n , ne \geq 0\}$. Corresponding grammar G will have P: S → aAb | ε

Kleene Star $L_1 = \{ a^n b^n \}^*$

The corresponding grammar $G_1$ will have additional productions S1 → SS1 | ε

**Q.5. (b) State pumping lemma for CFL. Provide an example to understand.**

**Ans:** If **L** is a context-free language, there is a pumping length **p** such that any string $w \in L$ of length $\geq p$ can be written as $w = uvxyz$, where $vy \neq \varepsilon$, $|vxy| \leq p$, and for all $i \leq 0$, $uv^i xy^i z \varepsilon L$.

**Applications of Pumping Lemma**

Pumping lemma is used to check whether a grammar is context free or not. Let us take an example and show how it is checked.

**Problem**

To check whether the language $L = \{x^n y^n z^n \mid n \geq 1\}$ is context free or not.

**Solution.**

Let **L** is context free. Then, **L** must satisfy pumping lemma.

At first, choose a number **n** of the pumping lemma. Then, take **z** as $0^n 1^n 2^n$.

Break **z** into **uvwxy,** where

$|vwx| \leq n$ and $vx \neq \varepsilon$.

Hence **vwx** cannot involve both 0s and 2s, since the last 0 and the first 2 are at least (n+1) positions apart. There are two cases–

**Case 1– vwx** has no 2s. Then **vx** has only 0s and 1s. Then **uwy**, which would have to be in **L**, has **n** 2s, but fewer than **n** 0s or 1s.

**Case 2**–vwx has no 0s.

Here contradiction occurs.

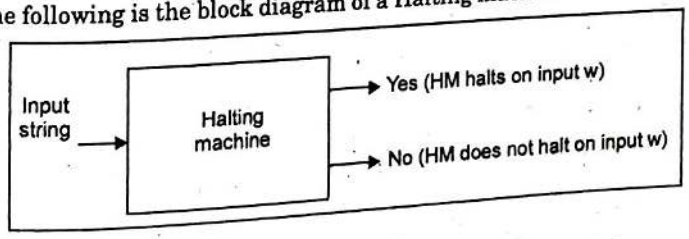
Hence, **L** is not a context-free language.

**Q.6. (a) State and prove Halting Problem.**

**Ans. Input** – A Turing machine and an input string **w**.

**Problem:** Does the Turing machine finish computing of the string **w** in a finite number of steps? The answer must be either yes or no.

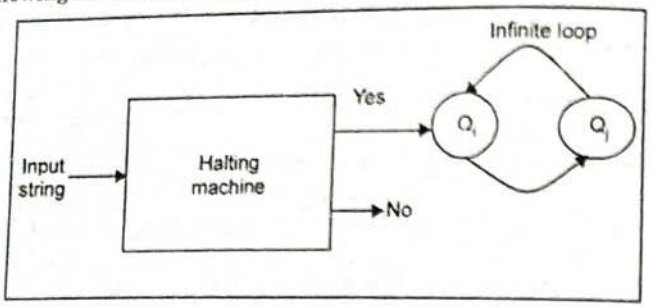
**Proof:** At first, we will assume that such a Turing machine exists to solve this problem and then we will show it is contradicting itself. We will call this Turing machine as a **Halting machine** that produces a 'yes' or 'no' in a finite amount of time. If the halting machine finishes in a finite amount of time, the output comes as 'yes', otherwise as 'no'. The following is the block diagram of a Halting machine–



Now we will design an **inverted halting machine (HM)'** as –

- If **H** returns YES, then loop forever.
- If **H** returns NO, then halt.

The following is the block diagram of an 'Inverted halting machine' –




Further, a machine $(HM)_2$ which input itself is constructed as follows–

- If $(HM)_2$ halts on input, loop forever.
- Else, halt.

Here, we have got a contradiction. Hence, the halting problem is **undecidable.**

**Q.6. (*b*)What is Turing Machine? What are it's different variant? Explain.**

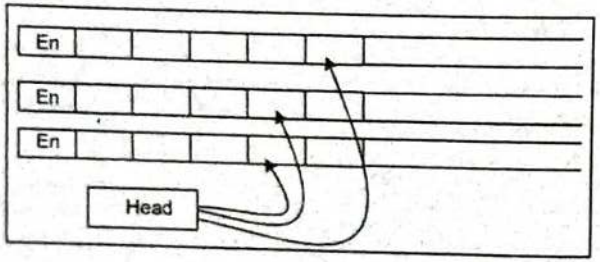
**Ans**: A Turing Machine (TM) is a mathematical model which consists of an infinite length tape divided into cells on which input is given. It consists of a head which reads the input tape. A state register stores the state of the Turing machine. After reading an input symbol, it is replaced with another symbol, its internal state is changed, and it moves from one cell to the right or left. If the TM reaches the final state, the input string is accepted, otherwise rejected.

A TM can be formally described as a 7-tuple (Q, X, Σ, δ, $q_0$, B, F) where –

- **Q** is a finite set of states
- **X** is the tape alphabet
- Σ is the input alphabet
- δ is a transition function; $\delta : Q \times X \rightarrow Q \times X \times \{\text{Left\_shift, Right\_shift}\}$.
- $q_0$ is the initial state
- $B$ is the blank symbol
- $F$ is the set of final states.

**Different variants of turing machine are:**

**1. Multi-tape Turing Machines**: It have multiple tapes where each tape is accessed with a separate head. Each head can move independently of the other heads. Initially the input is on tape 1 and others are blank. At first, the first tape is occupied by the input and the other tapes are kept blank. Next, the machine reads consecutive symbols under its heads and the TM prints a symbol on each tape and moves its heads.






A Multi-tape Turing machine can be formally described as a 6-tuple (Q, X, B, δ, $q_0$, F) where –

- $Q$ is a finite set of states
- $X$ is the tape alphabet
- $B$ is the blank symbol
- δ is a relation on states and symbols where

$\delta: Q \times X^k \rightarrow Q \times (X \times \{\text{Left\_shift, Right\_shift, No\_shift}\})^k$

where there is $k$ number of tapes

- $q_0$ is the initial state
- $F$ is the set of final states

**2. Multi-track Turing Machines:** It is a specific type of Multi-tape Turing machine, contain multiple tracks but just one tape head reads and writes on all tracks. Here, a single tape head reads $n$ symbols from $n$ tracks at one step. It accepts recursively enumerable languages like a normal single-track single-tape Turing Machine accepts.

A Multi-track Turing machine can be formally described as a 6-tuple ($Q, X, \Sigma, \delta, q_0$, F) where –

- **Q** is a finite set of states
- **X** is the tape alphabet
- Σ is the input alphabet
- δ is a relation on states and symbols where

$\delta(Q_i, [a_1, a_2, a_3, \ldots]) = (Q_j, [b_1, b_2, b_3, \ldots], \text{Left\_shift or Right\_shift})$

- $q_0$ is the initial state
- $F$ is the set of final states

**3. Non-Deterministic Turing Machine:** In non-deterministic turing machine , for every state and symbol, there are a group of actions the TM can have. So, here the transitions are not deterministic. The computation of a non-deterministic Turing Machine is a tree of configurations that can be reached from the start configuration.

An input is accepted if there is at least one node of the tree which is an accept configuration, otherwise it is not accepted. If all branches of the computational tree halt on all inputs, the non-deterministic Turing Machine is called a **Decider** and if for some input, all branches are rejected, the input is also rejected.

A non-deterministic Turing machine can be formally defined as a 6-tuple ($Q, X, \Sigma, \delta$ $q_0, B, F$) where –

- $Q$ is a finite set of states
- $X$ is the tape alphabet
- Σ is the input alphabet
- δ is a transition function;

$\delta : Q \times X' \rightarrow P(Q \times X \times \{\text{Left\_shift, Right\_shift}\})$.

- $q_0$ is the initial state
- $B$ is the blank symbol
- $F$ is the set of final states.

**Q.7. (*a*) State and Prove Savitch's Theorem**

**Ans. Savitch's Theorem:** Savitch theorem gives a relationship between deterministic and non-deterministic space complexity. It states that for any function.

$$f \in \Omega(\log(n)),$$

$$\text{NSPACE}(f(n)) \subseteq \text{DSPACE}((f(n))^2)$$

In other way, if a non-deterministic Turing machine can solve a probelm using $f(n)$ space, an ordinary deterministic Turing machine can solve the same problem in the square of that space bound.

**Proof:** There is a proof of the theorem that is constructive; is demonstrates an algorithm for STCON, the problem of determining whether there is a path between two vertices in a directed graph, which runs in $0(\log n)^2$).

Space for $n$ vertices. S TCON can be solved from this problem by setting $k$ to $n$. To test for a k edge path from, one way test whether each vertex $u$ may be the midpoint of the path by recursively searching for path of half the length from $s$ to $u$ and $u$ to $t$.

**Code:**

```
def k - edge - path (s, t, k):
if                    k  = =0
return                s  = =t
if                    k  = =1
return                s  = =t or (s, t)
in edges
for u in vertices
if k- edge- path (s, 4, floor (k/2)) and k- edge-path return true return false.
                      (u, t, ceil) (k/2)) :
return true
return false.
```

**Q.7. (*b*) Briefly Explain Cook's Theorem.**

**Ans. Defination:** The satisfiability problem (SAT) is the problem: Given a boolean expression, is it satisfiable?

**Theorem:** SAT is NP-Complete.

**Proof: PART I:** SAT ε NP.

If the encoded expression E is of Length n, then the number of variables is [n/2]. Hence, for guessing a truth assignment t we can use multitape TM for E. The time taken by a multitape NTM M is 0(n). The M evaluates the value of E for a truth assignment t. This is done in $O(n^2)$ time. An equivalent single tape TM takes $O(n^4)$ time. Once an accepting truth assignment is found, M accepts E and M and halts. Thus we have found a polynomials time NTM for SAT. Hence SAT ε NP.

PART II: POLYNOMIAL-TIME REDUCTION OF ANY L IN NP TO SAT.

**1. Construction of NTM for L:** Let L be any language in NP. Then there exists a single-tape NTM M and a polynomial p(n) such that the time taken by M for an input of length n is at most p(n) along any branch. If M accepts an input w and |w| = n, then there exists a sequence of moves of M such that

- $\alpha_0$ is the initial ID of M with input w.
- $\alpha_0 \vdash \alpha_1 \vdash \ldots \vdash \alpha_k$, $k \le P(n)$
- $\alpha_k$ is an ID with an accepting state.

**2. Representation of sequence of moves of M:** As the maximum number of steps on w is p(n) we need not bother about the contents beyond p(n) cells. We can write $\alpha_i$ as a sequence of $p(n) + 1$ symbols (one symbol for the state and the remaining symbols for the tape symbols. So $a_i = x_{i_0} x_{i_1} \ldots x_{ip(n)}$. If $\alpha_m$ is an accepting ID in the course of processing $w$ then we write $\alpha_0 \vdash \ldots \vdash \alpha a \vdash \alpha_m = \alpha_{i, p(n)}$.

**3. Representation of IDs in terms of Boolean Variables:** We define a boolean variable $y_{ijA}$ corresponding to $(i, j)$th entry in the $i$th ID. The variable $y_{ijA}$ represents the proposition that $x_{ij} = A$, where $A$ is a state or tape symbol and $0 \le i, j \le p(n)$.

**4. Polynomial Reduction of M to SAT:** In order to check that the reduction of M to SAT is correct, we have to ensure the correctness of



($a$) the initial ID.

($b$) the accepting ID and

($c$) the intermediate moves between successive IDs.

**($i$) Simulation of initial ID:** $x_{00}$ must start with the initial state $q_0$ of $M$ followed by the symbols of $w = a_1 a_2 \ldots a_n$ of length $n$ and ending with b's. The corresponding boolean expression S is defined as:

$$S = y_{00}q_0 \wedge y_{01a_1} \wedge \ldots \wedge y_{0n\,an} \wedge y_{0n\,a_{n+1}} \ldots \wedge y_{0,p(n),s}$$

**($ii$) Simulation of accepting ID:** $\alpha_{p(n)}$ is the accepting ID. If $p_1, p_2, \ldots p_k$ are the accepting states of $M$, then $\alpha_{p(n)}$ contains one of $p_i$'s $1 \le i \le k$ in any place j.

It is given by

$$F = F_0 \vee \ldots \vee F_{p(n)}$$

where $F_j = y_{p(n),j,p_1} \vee y_{p(n),j,p2} \vee \ldots \vee y_{p(n),n,pk}$

**($iii$) Simulation of Intermediate Moves:** We have to simulate valid moves $\alpha_i \vdash \alpha_{i+1}$, i = 0, 1, 2... p(n). corresponding to each move, we have to define a boolean variable $N_i$. Hence the entire sequence of IDs leading to acceptance of w is.

$$N = N_0 \wedge N_1 \wedge \ldots \wedge N_{p(n)-1}$$

**Formulation of $B_{ij}$:** When the state of $\alpha_i$ is none of $x_{i,j-1}$, $x_{ij}$, $x_{i,j+1}$, then the transition corresponding to $\alpha_{i+1}$ will not affect $x_{i,j+1}$. In this case $x_{i+1,j} = x_{ij}$.

- **Formulation of $A_{ij}$:** This step corresponds to the correctness of the 2 × 3 array

| | | |
|---|---|---|
| $x_{i,j-1}$ | $x_{ij}$ | $x_{i,j+1}$ |
| $x_{i+1,j-1}$ | $x_{i+1,j}$ | $x_{i+1,j+1}$ |

**Defination of $N_i$ and N:** We define $N_i$ and N by

$N_i = (A_{io} \vee B_{io}) \wedge (A_{i1} \vee B_{i1}) \wedge \ldots \wedge (A_i, p(n) \vee Bi, p(n))$

$N = N_0 \wedge N_1 \wedge N_2 \wedge \ldots \wedge N_{p(n)-1}$

**5. Completion of Proof:** Let $E_{m,w} = S \wedge N \wedge F$

We have seen that the time taken to write S and F are 0(p(n)) and the time taken for N is 0 ($P^2$(n)). Hence the time taken to write $E_{M,w}$ its ($p^2$(n).

Also M accepts w if and only if $E_{M,w}$ is satisfiable Hence the determinsitic multiple TM $M_1$ can convert W to a boolean expression $E_{M,w}$ in $0(p^2(n))$ time. An equivalent single tape TM taken 0 ($p^4$(n)) time. This proves the Part II, of the cook's theorem, thus completing the proof of this theorem.

**Q.8 Write short notes on *any two:***

**($a$) Space and Time Complexity**

**($b$) Turing Church's Thesis**

**($c$) Chomsky Normal Form.**

**Ans. ($a$) Space and Time Complexity:** The complexity of an algorithm is a function describing the efficiency of the algorithm in terms of the amount of data the algorithm must process. Usually there are natural units for the domain and range of this function. There are two main complexity measures of the efficiency of an algorithm:

**Time complexity** is a function describing the amount of time an algorithm takes in terms of the amount of input to the algorithm. "Time" can mean the number of memory accesses performed, the number of comparisons between integers, the number

of times some inner loop is executed, or some other natural unit related to the amount of real time the algorithm will take. We try to keep this idea of time separate from "wall clock" time, since many factors unrelated to the algorithm itself can affect the real time (like the language used, type of computing hardware, proficiency of the programmer, optimization in the compiler, etc.). It turns out that, if we chose the units wisely, all of the other stuff doesn't matter and we can get an independent measure of the efficiency of the algorithm.

**Space complexity** is a function describing the amount of memory (space) an algorithm takes in terms of the amount of input to the algorithm. We often speak of "extra" memory needed, not counting the memory needed to store the input itself. Again, we use natural (but fixed-length) units to measure this. We can use bytes, but it's easier to use, say, number of integers used, number of fixed-sized structures, etc. In the end, the function we come up with will be independent of the actual number of bytes needed to represent the unit. Space complexity is sometimes ignored because the space used is minimal and/or obvious, but sometimes it becomes as important an issue as time.

**(*b*)Turing Church's Thesis:**

• As stated by Kleene:

Every effectively calculable function (effectively decidable predicate) is general recursive.

• Any mechanical computation can be performed by a Turing Machine

• There is a TM-n corresponding to every computable problem

• We can model any mechanical computer with a TM

• The set of languages that can be decided by a TM is identical to the set of languages that can be decided by any mechanical computing machine

• If there is no TM that decides problem P, there is no algorithm that solves problem P.

**(c) Chomsky Normal Form:**

A CFG is in Chomsky Normal Form if the Productions are in the following forms–

• $A \rightarrow a$

• $A \rightarrow BC$

• $S \rightarrow \varepsilon$

where A, B, and C are non-terminals and **a** is terminal.

**Algorithm to Convert into Chomsky Normal Form –**

**Step 1:** If the start symbol $S$ occurs on some right side, create a new start symbol $S'$ and a new production $S \rightarrow S$.

**Step 2:** Remove Null productions. (Using the Null production removal algorithm).

**Step 3:** Remove unit productions. (Using the Unit production removal algorithm discussed earlier).

**Step 4:** Replace each production $A \rightarrow B_1 \ldots B_n$ where **n > 2** with $A \rightarrow B_1 C$ where $C \rightarrow B_2 \ldots B_n$. Repeat this step for all productions having two or more symbols in the right side.

**Step 5:** If the right side of any production is in the form $A \rightarrow aB$ where **a** is a terminal and $A$, $B$ are non-terminal, then the production is replaced by $A \rightarrow XB$ and $X \rightarrow a$. Repeat this step for every production which is in the form $A \rightarrow aB$.

# FIRST TERM EXAMINATION [FEB. 2017]
# FOURTH SEMESTER [B.TECH.]
# THEORY OF COMPUTATION [ETCS–206]

**Time : 1½ hrs.** **M.M. : 30**

*Note: Q. No. 1 is compulsory. Attempt any two questions from the rest.*

**Q.1. (*a*) Define Automata and differentiate between DFA and NDFA.** **(2)**

**Ans.** The term "Automata" is derived from the Greek word "αὐτόματα" which means "self-acting". An automaton (Automata in plural) is an abstract self-propelled computing device which follows a predetermined sequence of operations automatically.

An automaton with a finite number of states is called a **Finite Automaton**(FA) or **Finite State Machine** (FSM)

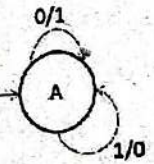
| Deterministic Finite Automata | Non-Deterministic Finite Automata |
|---|---|
| 1. Transition function ($\delta : Q \times \Sigma \rightarrow Q$) | Transition relation $\Delta : Q \times \Sigma \rightarrow P(Q)$. |
| 2. DFA cannot use empty string transition | NFA can use empty string transition |
| 3. DFA requires more space | NFA requires less space. |
| 4. Backtracking is allowed in DFA | In NFA it may or may not be allowed |
| 5. DFA will reject the string if it end at other than accepting state. | If all of the branches of NFA dies or rejects the string, we can say that NFA reject the string. |

**Q.1. (b) Differentiate Mealy and Moore machines.Design a mealy machine to find 1's complement of given binary number.** **(2)**

**Ans.**

| Mealy Machine | Moore Machine |
|---|---|
| 1. Output depends both upon present state and present input. | Output depends only upon the present state. |
| 2. Generally, it has fewer states than Moore Machine. | Generally, it has more states than Mealy Machine. |
| 3. Output changes at the clock edges. | Input change can cause change in output change as soon as logic is done. |
| 4. Mealy machines react faster to inputs | In Moore machines, more logic is needed to decode the outputs since it has more circuit delays. |

So we can see that every '0' will be replaced by '1' and vice-versa




**Example**

Suppose string is 10001 and we will start parsing from left to right.

Every 0 will be replaced by 1 and vice versa.

So we will get the output as = 01110

**Q.1. (c) Represent the following set by a regular expression $\{1^{2n+1} \mid n>0\}$ and describe ,in english language,the set represented by following regular expression a(a +b)*b** **(2)**

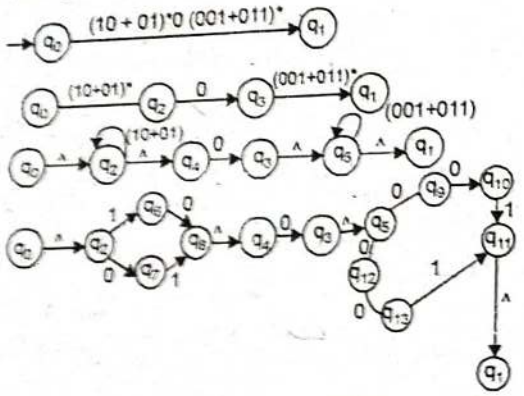
**Ans.** $1(11)^+$

Set of all strings starting with a and ending with b containing any no of a's &b's

**Q.1. (d) Find all strings of length 4 or less in the regular set represented by following regular expression (a*b+b*a)* (2)**

**(10+01)*0(001+011)***

**Ans.** Strings of length 4 or less are: ba,aba,bba,abba,aabbba,aaabbbba etc.




**Q.1. (e) Define Construct free grammar (CFG).Consider a CFG g whose productions are**

**S → 0B/1A, A → 0/0S/1AA, B → 1/1S/0BB. Find the right most derivation for the string 00110101 in grammar (2)**

Ans. A context-free grammar (CFG) consisting of a finite set of grammar rules is a quadruple (N, T, P, S) where

- N is a set of non-terminal symbols.
- T is a set of terminals where N ∩ T = NULL.
- P is a set of rules, P: N → (N ∪ T)*, i.e., the left-hand side of the production rule P does have any right context or left context.
- S is the start symbol.

**Example**

- The grammar ({A}, {a, b, c}, P, A), P : A → aA, A → abc.
- The grammar ({S, a, b}, {a, b}, P, S), P: S → aSa, S → bSb, S → ε
- The grammar ({S, F}, {0, 1}, P, S), P: S → 00S | 11F, F → 00F | ε

S → 0B
→ 00BB
→ 00B1S
→ 00B10B
→ 00B101S
→ 00B1010B
→ 00B10101
→ 00110101

**Q.2. (a) Minimize the following DFA M = ({ q0,q1,q2,q3,q4,q5,q6} ,{a,b} , delta,q0,{q6}) where delta is given by (5)**

Ans. state

| state | a | b |
|---|---|---|
| $q_0$ | $q_0$ | $q_0$ |
| $q_1$ | $q_2$ | $q_5$ |
| $q_2$ | $q_6$ | $q_4$ |
| $q_3$ | $q_0$ | $q_5$ |
| $q_4$ | $q_0$ | $q_6$ |



| | | |
|---|---|---|
| $q_5$ | $q_1$ | $q_4$ |
| $q_6$ | $q_1$ | $q_3$ |

$\prod = \{q_0, q_1, q_2, q_3, q_4, q_5, \} \{q_6\}$

$\prod_2 = \{ q_0, q_1, q_2, q_3, q_5\}, \{q_4\}, \{q_6\}$

$\prod_3 = \{ q_0, q_1, q_3\}\{q_4\}, \{q_6\}, \{q_2, q_5\}$

$\prod_4 = \{q_0, q_3\}\{q_1\}\{q_4\}, \{q_6\}, \{q_2, q_5\}$

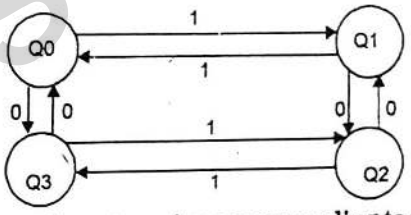
$\prod_5 = \{q_0\}, \{q_1\}, \{q_2\}, \{q_3\}, \{q_4\}, \{q_5, \} \{q_6\}$

it can not be minimized

**Q.2. (b) Design a DFA for the following regular expression P = (00)*(11)*.Afterwords,convert that DFA in a way (i) it would accept complement of a given regular expression (p') (5)**

**(ii) It Would accept reverse of given regular expression ($p^R$)**

**Ans.**




**Q.3. (a) Find the regular expression corresponding to figure 1. (5)**

**Ans.** We form the equations:

$q_1 = q_1 0 + q_3 0 + q_4 0 + \epsilon$

$q_2 = q_1 1 + q_2 1 + q_4 1$

$q_3 = q_2 0$

$q_4 = q_3 1$

putting the value of $q_3$ in(iv) we get

$q_4 = q_2 1 = q_3 01$

putting the value of q4 in equation (ii) ,we get

$q_2 = q_1 1 + q_2 1 = q_2 011 = q_1 1 + q_2(1 + 011)$

$= q_1 1(1 + 011)^*$

$== q_1(1(1 + 011)^*)$

Putting $q_3$ and $q_4$ in equation (i) ,we get

$q_1 = q_1 0 + q_2 00 + q_2 010 + \epsilon$

$= q_1 0 + q_2(00 + 010) + \epsilon$

$= q_1 0 + q_1 1(1+011)^*(00+010) + \epsilon$

$= \epsilon\ (0 + 1(1+011)^*(00+010))^*$

Put $q_2$ in $q_4 = q_2 01$

$= q_1 1(1+011)^*01$

$q_4 = (0 + 1(1+011)^*(00+010))^*1(1+011)^*01$

$q_4$ is the final state so required regular expression is:

$(0+1(1+011)^*(00+010))^*1\ (1+011)^*01)$

**Q.3. (b) Prove that L = { $a^P$ | P is Prime number} is not regular. (5)**

**Ans.** 1.We don't know m, but assume there is one.

2. Choose a string $w = a^n$ where n is a prime number and $|xyz| = n > m+1$. (This can always be done because there is no largest prime number.) Any prefix of w consists entirely of a's.

3. We don't know the decomposition of w into xyz, but since $|xy| \le m$, it follows that $|z| > 1$. As usual, $|y| > 0$,

4. Since $|z| > 1$, $|xz| > 1$. Choose $i = |xz|$. Then $|xy^iz| = |xz| + |y||xz| = (1 + |y|)|xz|$. Since $(1 + |y|)$ and $|xz|$ are each greater than 1, the product must be a composite number. Thus $|xy^iz|$ is a composite number

**Q.4. (a) Reduce the following grammar G in to Chomsky Normal Form (CNF) { S → 0AD,A → 0B/1AB,B → 1,D → 2} here S,A,B & D are the variables ,S is the Start Variable and 0,1,2 are the terminals.** **(5)**

**Ans.** S → 0AD,
A → 0B/1AB,
B → 1 is in CNF
D → 2 is in CNF
S → $C_0$AD
$C_0$ → 0
A → 0B
A → $C_0$b
A → 1AB
A → $C_1$AB
$C_1$ → 1
S → $C_0$ AD
S → $C_0C_1$
$C_1$ → AD
A → $C_1$AB
A → $C_1$C2
$C_2$ → AB

**Q.4. (b) Remove the null Production from the given grammar-{ S →aS/AB,A→ ∈ B → ∈ ,D → b} here S,A,B & D are the variables , S is the Start Variable,a,b are the terminls and ∈ the null symbol.** **(5)**

**Ans. Step 1.** Construction of set W of all nullable variables.
A variable A in a context-free grammar is nullable if A ⇒ ^.

$W_1 = \{A_1 \in V_n : A_1 \to \hat{} \text{ is a production in G}\}$
$= \{A, B\}$
$W_2 = W_1 \cup \{A \in V_N : \text{there exists a production } A \to \alpha \text{ with } \alpha \in W_1\}$
$= \{A, B\} \cup \{S\} = \{S, A, B\}$
$W_3 = \varnothing \cup W_2$, so $W = \{S, A, B\}$
$= W_2$.

**Step 2.** Construction of P'.
(a) Any production whose R.H.S does not have any nullable variable is included in P'.
D → b is included in P'.
(b) The productions are obtained either by not erasing any nullable variable on the R.H.S. of $A \to X_1 X_2....X_k$ or by erasing some or all nullable variables provided some symbol appears on the R.H.S. after erasing
S → aS gives, S → aS and S → a
S → AB gives, S → AB, S → A, S → B.
The required grammer without null production is

$G_1 = \{S, A, B, D\}, \{a, b\}, P', S)$

# END TERM EXAMINATION [MAY-JUNE 2017]
# FOURTH SEMESTER [B.TECH]
# THEORY OF COMPUTETION [ETCS–206]

Time : 3 Hrs. M.M. : 75

**Note:** *Attempt any five questions including Q. No. 1 which is compulsory.*

**Q.1. (a) Explain deterministic and non deterministic automata with example.** **(5)**

**Ans:** Refer Q.1. (a) First Term Examination 2017.

**Q.1. (b) State and prove Pumping Lemma for languages.** **(5)**

**Ans.** Pumping Lemma is to be applied to show that certain languages are not regular. It should never be used to show a language is regular.

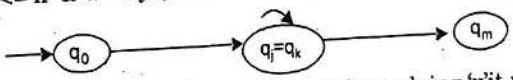
**Statement:** Let M = (Q, Σ, δ, $q_0$, F) be a finite automata with 'n'states.Let L be the regular set accepted by M.Let w ε L and |w|>= m.If m>= n,then there exist x,y,z such that w= xyz,y !=∈ and $xy^iz$ belongs to L for each i>= 0.

**Proof**: Find a non-empty string $y^i$ not too far from the beginning of w that can be pumped i.e. repeating $y^i$any no of times,keeps the resulting string in the language.

Let $w = a_1,a_2,a_3$--------$a_m$ m>=n is a string.
$\delta(q_0, a_1,a_2,a_3,$------$a_i) = q_i$ for i= 1,2-----m
$Q_1 = \{ q_0q_1q_2$-------$q_m \}$
i.e. $Q_1$ = Sequence of states in the path with path value $w = a_1,a_2,a_3,$--------$a_m$
as there are only 'n'distinct states ,at least two states in $Q_1$ must coincide.So,among the various similar states ,we take the first pair of states .Let us take them $q_j,q_k$ ($q_j = q_k$).Then j & k satisfy the condition 0<= j<k<=n.

The string 'w' can be decomposed in to 3 substrings.
$a_1,a_2,a_3,$---------$a_j$ , $a_{j+1}$,--------$a_k$ , $a_{k+1}$,--------$a_m$. Let x,y,z denote the strings respectively.

As k<=n, |xy|<= n & w = xyz,so the transition diagram




Automata 'M' starts at the initial state $q_0$. Only applying 'x'it reaches $q_j(=q_k)$,On applying $y^i$it comes back to $q_j(=q_k)$so after application of $y^i$ for each i>=0,the automaton is in the same state $q_j$.on applying z,it reaches $q_m$,a final state

Hence $xy^iz$ belongs to L

**Q.1. (c) Differentiate between Chomsky Normal Form and Greibach Normal Form.** **(5)**

**Ans.** A CFG is in **Chomsky Normal Form** if the Productions are in the following forms-

- A → a
- A → BC
- S → ε
-

where A, B, and C are non-terminals and a is terminal.
Algorithm to Convert into Chomsky Normal Form–
**Step 1** – If the start symbol **S** occurs on some right side, create a new start symbol **S'** and a new production **S' → S**.

**Step 2**– Remove Null productions. (Using the Null production removal algorithm discussed earlier)

**Step 3**– Remove unit productions. (Using the Unit production removal algorithm discussed earlier)

**Step 4**– Replace each production $A \rightarrow B_1..B_n$ where $n > 2$ with $A \rightarrow B_1C$where $C \rightarrow B_2..B_n$. Repeat this step for all productions having two or more symbols in the right side.

**Step 5**– If the right side of any production is in the form $A \rightarrow aB$ where a is a terminal and A, B are non-terminal, then the production is replaced by $A \rightarrow XB$ and $X \rightarrow a$. Repeat this step for every production which is in the form $A \rightarrow aB$.

A CFG is in **Greibach Normal Form** if the Productions are in the following forms

$$A \rightarrow b$$
$$A \rightarrow bD_1..D_n$$
$$S \rightarrow \varepsilon$$

where $A, D_1,....,D_n$ are non-terminals and b is a terminal

Algorithm to Convert a CFG into Greibach Normal Form

**Step 1** – If the start symbol **S** occurs on some right side, create a new start symbol **S'** and a new production **S'→ S**.

**Step 2** – Remove Null productions. (Using the Null production removal algorithm discussed earlier)

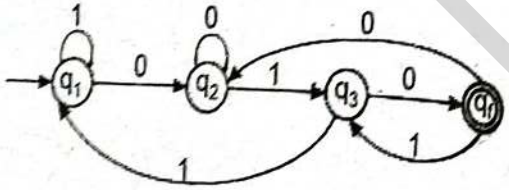
**Step 3** – Remove unit productions. (Using the Unit production removal algorithm discussed earlier)

**Step 4** – Remove all direct and indirect left-recursion.

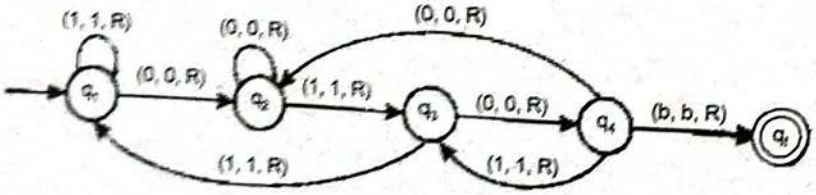
**Step 5** – Do proper substitutions of productions to convert it into the proper form of GNF.

**Q.1. (d) Construct a Turing Machine M to accept the set L of all string over {0,1} ending with 010.** **(5)**

**Ans.** The DFA is as follows:




Now convert this DFA to Turing machine M. In DFA, the move is always right so the Turing machine M, move will always be to the right.






| Present State | 0 | 1 | b |
|---|---|---|---|
| $\rightarrow q_1$ | $(0, q_2, R)$ | $(1, q_1, R)$ | – |
| $q_2$ | $(0, q_2, R)$ | $(1, q_3, R)$ | – |
| $q_3$ | $(0, q_4, R)$ | $(1, q_1, R)$ | – |
| $q_4$ | $(0, q_2, R)$ | $(1, q_3, R)$ | $(b, q_f, R)$ |
| $(q_f)$ | – | – | – |

$q_f$ is the unique final state of M.

**Q.1. (e) State and prove Savitch Theorem.** **(5)**

**Ans. Savitch's Theorem:** Savitch theorem gives a relationship between determinstic and non-deterministic space complexity. It states that for any function.

$$f \in \Omega. (\log (n),$$
$$\text{NSPACE } (f(n)) \subseteq \text{DSPACE } ((f(n))^2)$$

In other way, if a non-deterministic Turing machine can solve a probelm using f(n) space, an ordinary deterministic Turing machine can solve the same problem in the square of that space bound.

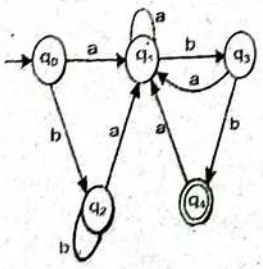
**Proof:** There is a proof of the theorem that is constructive; is demonstrates an algorithm for STCON, the problem of determining whether there is a path between two vertices in a directed graph, which runs in $0(\log n)^2$.

Space for *n* vertices. S TCON can be solved from this problem by setting *k* to *n*. To test for a k edge path from, one way test whether each vertex *u* may be the midpoint of the path by recursively searching for path of half the length from *s* to *u* and *u* to *t*.

**Code:**

```
def k - edge - path (s, t, k):
if                          k  =  =0
return                      s  =  =t
if                          k  =  =1
return                      s  =  =t or (s, t)
in edges
for u in vertices
if k- edge- path (s, 4, floor (k/2)) and k- edge-path return true return false.
                                    (u,t,ceil)(k/2)):
return true
return false.
```

**Q.2. (a) Design a minimum state automation for the following DFA** **(6.5)**



**Ans.** First we will construct a transition table as follows

| State Σ | a | b |
|---|---|---|
| $\rightarrow q_0$ | $q_1$ | $q_3$ |
| $q_1$ | $q_1$ | $q_3$ |
| $q_2$ | $q_1$ | $q_3$ |
| $q_3$ | $q_1$ | $q_4$ |
| $(q_4)$ | $q_1$ | $q_2$ |

Now we construct

$$\pi_0 = \{(q_4), \{q_0, q_1, q_2, q_3)\}$$

i.e., it is the partitioned into two group–final state and non-final state.

Now we will compare $q_0$ and $q_1$ under 'a' column we get $q_1$ state and under b-column. We get $q_2$ and $q_3$. Both $q_2$ and $q_3$ lie in the same class hence $q_0$ and $q_1$ are 0-equivalent, Similarly, $q_0$ and $q_2$ are 0-equivalent.

But $q_0$ and $q_3$ are not 0-equivalent because under 'a'– column of $q_0$ and $q_3$, we get $q_1$ and $q_1$ but under b-column of $q_0$ and $q_3$, we get $q_2$ and $q_3$, which do lie in same class. Hence, we get

$$\pi_1 = (\{q_4\}, \{q_0, q_1, q_2\}, \{q_3\})$$

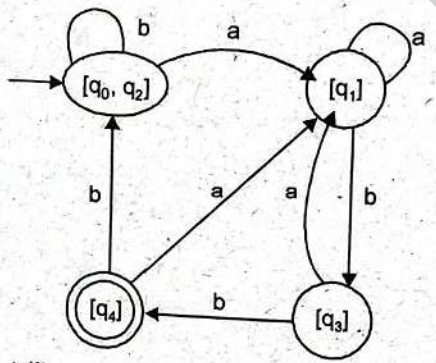
Now, the states $q_0$ and $q_1$ are not 1-equivalent because under b-column of $q_0$ and $q_1$ we get $q_2$ and $q_3$ where $q_2$ and $q_3$ do not lie in the same class.

But $q_0$ and $q_2$ are 1-equivalent. Hence,

$$\pi_2 = \{(q_4), \{q_0, q_2\}, \{q_1\}, \{q_3\})$$

Now the states $q_0$ and $q_2$ are 2-equivalent. Thus we can not further partition states in $\pi_2$. So get the minimum automata as

| State/Σ | a | b |
|---|---|---|
| $\rightarrow [q_0, q_2]$ | $[q_1]$ | $[q_0, q_2]$ |
| $[q_1]$ | $[q_1]$ | $[q_3]$ |
| $[q_3]$ | $[q_1]$ | $[q_4]$ |
| $(q_4)$ | $[q_1]$ | $[q_0, q_2]$ |




**Q.2. (b) Prove that (i) a*(ab)*(a*(ab)*(a*(ab)*)* = (a + ab) ***

**(ii) (1 +00*1)+(1+00*1)(0+10*1)*(0+10)*1) = (0+10*1)* (6)**



**Ans.** (i)

$$\text{L.H.S.} = \underbrace{a^*(ab)^*}_{p}(a^*(ab)^*)^*$$

Let $P = a^*(ab)^*$

$\therefore$ $\text{L.H.S.} = P(P)^* = P^* = (a^*(ab)^*)^* = (a + ab)^*$ $[\because (P^*Q^*)^* = (P + Q)^*]$

(ii) $\text{L.H.S.} = (1 + 00^*1)(\wedge + (0 + 10^*1))^*(0 + 10^*1))$

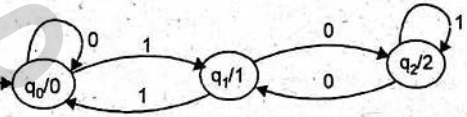
$= (1 + 00^*1)(0 + 10^*1)^*$

$= 1(\wedge + 00^*)(0 + 10^*1)^*$ $[\wedge + R^*R = R^*]$

$= 0^*1(0 + 10^*1)^* = \text{R.H.S.}$

**Q.3. (a) Convert the following Moore Machine to its equivalent Mealy Machine.** **(6.5)**

**Ans.** The transition diagram for the given problem can be drawn as




The output function λ' can be obtained using the following rule:

$$\lambda'(q, a) = \lambda(\delta(q, a))$$

Now, we will obtain for every transition corresponding to input symbol.

$\lambda'(q_0, 0) = \lambda(\delta(q_0, 0))$

$= \lambda(q_0)$ i.e., output of $q_0$.

$\lambda'(q_0, 0) = 0.$

$\lambda'(q_0, 1) = \lambda(\delta(q_0, 1))$

$= \lambda(q_1)$ i.e., output of $q_1$.

$\lambda'(q_0, 1) = 1$

$\lambda'(q_1, 0) = \lambda(\delta(q_1, 0))$

$= \lambda(q_2)$

$\lambda'(q_1, 0) = 2.$

$\lambda'(q_1, 1) = \lambda(\delta(q_1, 1))$

$= \lambda(q_0)$

$\lambda'(q_1, 1) = 0$

$\lambda'(q_2, 0) = \lambda(\delta(q_2, 0))$

$= \lambda(q_1)$

$\lambda'(q_2, 0) = 1$

$\lambda'(q_2, 1) = \lambda(\delta(q_2, 1))$

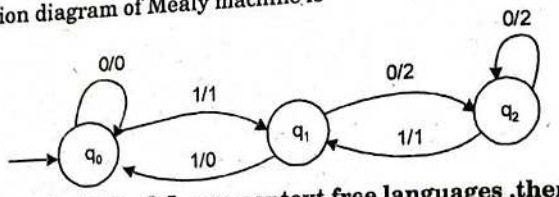
$= \lambda(q_2)$

$\lambda'(q_2, 1) = 2.$

Hence the transition table can be drawn as

| Present state | Next State | | | |
|---|---|---|---|---|
| | Input 0 | | Input 1 | |
| | State | Output | State | Output |
| $q_0$ | $q_0$ | 0 | $q_1$ | 1 |
| $q_1$ | $q_2$ | 2 | $q_0$ | 0 |
| $q_2$ | $q_1$ | 1 | $q_2$ | 2 |

The transition diagram of Mealy machine is




**Q.3. (b) Prove that if $L_1$ & $L_2$ are context free languages ,then $L = L_1 \cup L_2$ is also context free.** **(6)**

**Ans.** 1. Let $L_1$ and $L_2$ be generated by the CFG, G1 = (V1, T1, P1, S1) and G2 = (V2, T2, P2, S2), respectively.

2. Without loss of generality, subscript each nonterminal of G1 with a 1, and each nonterminal of G2 with a 2 (so that $V1 \cap V2 = \emptyset$).

3. Define the CFG, G, that generates $L_1 \cup L_2$ as follows: $G = (V1 \cup V2 \cup \{S\}, T1 \cup T2, P1 \cup P2 \cup \{S \to S1 \mid S2\}, S)$.

4. A derivation starts with either $S \Rightarrow S1$ or $S \Rightarrow S2$.

5. Subsequent steps use productions entirely from G1 or entirely from G2.

6. Each word generated thus is either a word in $L_1$ or a word in $L_2$.

Example • Let $L_1$ be PALINDROME, defined by: $S \to aSa \mid bSb \mid a \mid b \mid \wedge$

• Let $L_2$ be $\{a^n b^n \mid n \geq 0\}$ defined by: $S \to aSb \mid \wedge$

• Then the union language is defined by: $S \to S1 \mid S2$ $S1 \to aS1a \mid bS1b \mid a \mid b \mid \wedge$ $S2 \to aS2b \mid \wedge$

**Q.4. (a) Convert the following grammar G to Chomsky Normal Form: $S \to ABCa$ $A \to aAbb, A \to e, B \to bB, B \to b, B \to AC, C \to aCa, C \to e$** **(6)**

**Ans.** $A \to e, B \to b, C \to e$ are in CNF

$S \to ABCa, F \to b$ therefore $B \to FB$

Now we have to convert $S \to ABCa$

$A \to aAbb$, $C \to aCa$ now $M \to a$

Therefore

$S \to ABCM$

$A \to MAFF$

$C \to MCM$

$G \to CM$

Therefore

$S \to ABG$ ,$H \to BG$,$S \to AH$

$C \to MG$

$L \to FF$

So $A \to MAL$,$V \to AL$,$A \to MV$ are in CNF

**Q.4. (b) Prove that intersection of a CFL L with a regular language M is a CFL** **(6.5)**

**Ans.** If L1 is a context free language and L2 is a regular language then $L1 \cap L2$ is context free.

**Proof:** We do the case where $e \in L1$ & $L2 = \phi$. All other cases we leave to the reader.

By Lemma we can assume there exists a Chomsky normal form grammar G = (N, Σ, S, P) for L1.



By Lemma L2 = A1 U···U An where each Ai where each Ai is recognized by a DFA with exactly one final state.

Note that $L1 \cap L2 = L1 \cap (A1 \cup \cdots \cup An) = [n\ i=1\ (L1 \cap Ai)$. Since CFL's are closed under union (and this can be proven using CFG's, so this is not a cheat) we need only show that the intersection of L1 with a regular language recognized by a DFA with one final state is CFL.

Let M = (Q, Σ, δ, s, {f}) be a DFA with exactly one final state. We construct the CFG G' = (N' , Σ, S0 , P0 ) for $L1 \cap L(M)$.

1. The nonterminals N' are triples [p, V, r] where $V \in N$ and $p, r \in Q$.

2. For each production $A \to BC$ in P, for every $p, q, r \in Q$ we have the production $[p, A, r] \to [p, B, q][q, C, r]$ 2 in P'.

3. For every production $A \to \sigma$ in P, for every $(p, \sigma, q) \in Q \times \Sigma \times Q$ such that $\delta(p, \sigma) = q$ we have the production $[p, A, q] \to \sigma$ in P'

4. S 0 = [s, S, f]

**Q.5. (a) Prove that class of deterministic context free languages is closed under complement.** **(6.5)**

**Ans.** If the L = L(M), where M is a DPDA, then L = L(M'), where M' is obtained from M by changing accept states to reject states and vice versa.

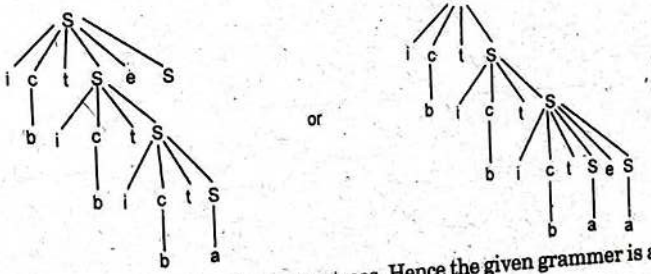
Proof, Let $L_1$ and $L_2$ are two CFLs. We will assume that complement of a context-free language is a CFL itself. Hence $L'_1$ and $L'_2$ both are CFLs. We can also state that $(L'_1 \cup L'_2)$ is context free (since CFLs are closed under union). But $(L'_1 \cup L'_2) = [L'_1 \cup L_2)$'i.e., $L = L_1 \cap L_2$ may or may not be CFL. The $L_1$ and $L_2$ are arbitrary CFLs, there may exist $L'_1$ and $L'_2$ which are not CFL. Hence complement of certain language may be context-free or may not be. Therefore we can say that CFL is not closed complement operation.

**Q.5. (b) Check whether the following grammer is ambiguous.If it is ambiguous remove the ambiguity and write an equivalent unambiguous grammar.S → i ctS/ictSeS/a,c →b** **(6)**

**Ans.** To check whether given grammer is ambiguous or not we will have some string for derivation tree such as

i b t i b t i b t a e a.

Now, we draw the derivation tree.



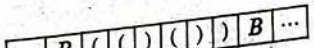

Thus, we have got more than two parse trees. Hence the given grammer is ambiguous

**inherent Ambiguity**

If every grammer that generates L is ambiguous, then the language is said to be "inherently ambiguous".

**Q.6. (a) Construct a turing machine for checking if a set of parenthesis is well formed** **(6)**

**Ans.**



$\delta$ is defined as:

1. $\delta(q_0, ) = (q_0 R$
2. $\delta(q_0, ) ) = Xq_1 L$
3. $\delta(q_1, ) = (q_0, Y, R)$ or $Y q_0 R$
4. $\delta(q_0, X) = (X q_0 R)$
5. $\delta(q_0, Y) = (Y q_0 R)$
6. $\delta(q_0, B) = Bq_2 L$
7. $\delta(q_2, X) = Xq_1 L$
8. $\delta(q_1, Y) = Y q_1 L$
9. $\delta(q_2, X) = Xq_2 L$
10. $\delta(q_1, Y) = Yq_1 L$
11. $\delta(q_2, B) = Bq_3 R$

**Transition Table**

| State | ( | ) | X | Y | B |
|---|---|---|---|---|---|
| $q_0$ | $(q_0, (, R)$ | $(q_1, X, L)$ | $(q_0, X, R)$ | $(q_0, Y, R)$ | $(q_2, B, L)$ |
| $q_1$ | $(q_0, Y, R)$ | – | $(q_1, X, L)$ | $(q_1, Y, L)$ | – |
| $q_2$ | – | – | $(q_2, X, L)$ | $(q_2, Y, L)$ | $(q_3, B, R)$ |

**Q.6. (b) Define Turing Machine. Give a block diagram with specified functions of each part of it. What is the difference between a Turing Machine and Two Way Finite Automata?** **(6.5)**

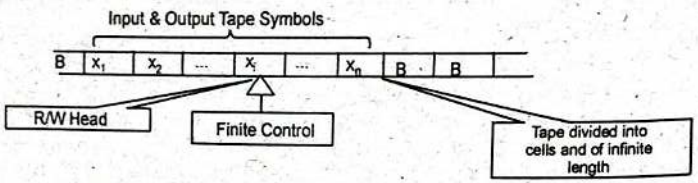
**Ans.** A Turing Machine (TM) is a mathematical model which consists of an infinite length tape divided into cells on which input is given. It consists of a head which reads the input tape. A state register stores the state of the Turing machine. After reading an input symbol, it is replaced with another symbol, its internal state is changed, and it moves from one cell to the right or left. If the TM reaches the final state, the input string is accepted, otherwise rejected.

A TM can be formally described as a 7-tuple (Q, X, $\Sigma$, $\delta$, $q_0$, B, F) where-

- **Q** is a finite set of states
- **X** is the tape alphabet
- $\Sigma$ is the input alphabet
- $\delta$ is a transition function; $\delta : Q \times X \rightarrow Q \times X \times \{Left\_shift, Right\_shift\}$.
- $q_0$ is the initial state
- **B** is the blank symbol

**F** is the set of final states

**THE TURING MACHINE MODEL**




**Turing machines vs. Finite Automata**

- TMs can both write on the tape and read from it
  - FSAs can only read (or only write if they are generators)
- The read/write head can move to the left and right
  - FSAs cannot "go back" on the input
- The tape is infinite
  - In FSAs the input and output is always finite



- The accept/reject states take immediate effect
  - There is no need to "consume" all the input
- TMs come in different flavours.

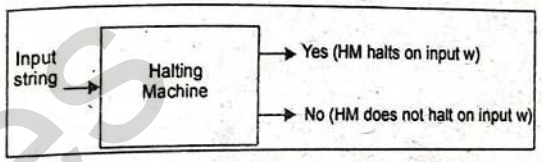
These differences are unimportant.

**Q.7. (a) State and prove Halting Problem for Turing Machine.** **(6.5)**

**Ans. Input** - A Turing machine and an input string **w.**

**Problem** - Does the Turing machine finish computing of the string **w** in a finite number of steps? The answer must be either yes or no.

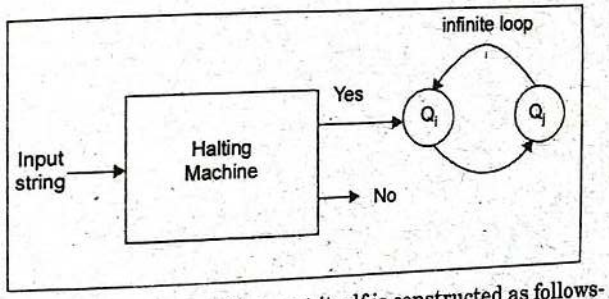
**Proof** - At first, we will assume that such a Turing machine exists to solve this problem and then we will show it is contradicting itself. We will call this Turing machine as a **Halting machine** that produces a 'yes' or 'no' in a finite amount of time. If the halting machine finishes in a finite amount of time, the output comes as 'yes', otherwise as 'no'. The following is the block diagram of a Halting machine-




Now we will design an **inverted halting machine (HM)'** as-

- If **H** returns YES, then loop forever.
- If **H** returns NO, then halt.

The following is the block diagram of an 'Inverted halting machine' -




Further, a machine $(HM)_2$ which input itself is constructed as follows-

- If $(HM)_2$ halts on input, loop forever.
- Else, halt.

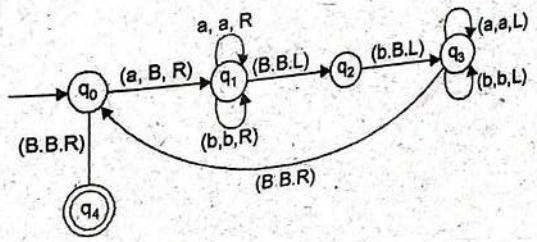
Here, we have got a contradiction. Hence, the halting problem is undecidable.

**Q.7. (b) Design a Turing Machine to accept the language L = {$a^n b^n$,n>1}.Show an ID for the string áaabbb'with tape symbols.** **(6)**

**Ans:** Let the turing machine be

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

Here M accepts the language L in which same number of a 's are followed by same no. of b's.



**Transition Table**

| Present state | Input symbol | | |
|---|---|---|---|
| | a | b | B |
| $\rightarrow q_0$ | $(q_1, B, R)$ | – | $(q_4, B, R)$ |
| $q_1$ | $(q_1, a, R)$ | $(q_1, b, R)$ | $(q_2, B, L)$ |
| $q_2$ | – | $(q_3, B, L)$ | – |
| $q_3$ | $(q_3, a, L)$ | $(q_3, b, L)$ | $(q_0, B, R)$ |
| $(q_4)$ | – | – | – |

Suppose the string is $a^3b^3$ i.e., aaabbb

| B | a | a | a | b | b | b | B |
|---|---|---|---|---|---|---|---|
| | ↑ | | | | | | |

Starting the leftmost 'a' at state $q_0$. Replacing it with B(blank symbol) and moves to state $q_1$ then R/w heads right to find the rightmost symbol and replacing it with blank symbol and after that R/w heads go left again to leftmost symbol replacing it with blank then move to right-most 'b' and replace it with Blank(B) and so no.

Here $q_0$ aaabbbB $\vdash$ B$q_1$ aabbb B Ba$q_1$abbbB $\vdash$ Ba$q_1$abbbB $\vdash$ Baa$q_1$bbbB

$\vdash$ Baab $q_1$bbB $\vdash$ Baabb $q_1$ bB

$\vdash$ Baabba$q_1$B $\vdash$ Baabb$q_2$bB

$\vdash$ Baab$q_3$bBB $\vdash$ Baa$q_3$bbBB $\vdash$ Ba$q_3$abbBB

$\vdash$ B$q_3$aabbBB $\vdash$ $q_3$BaabbBB

$\vdash$ B$q_0$aabbBB $\vdash$ BB$q_1$abbBB $\vdash$ BBa$q_1$bbBB

$\vdash$ BBab$q_1$BB $\vdash$ BBabb$q_1$BB $\vdash$ BBab$q_2$bBB

$\vdash$ BBa$q_3$bBBB $\vdash$ BB$q_3$abBBB $\vdash$ B$q_3$BabBBB

$\vdash$ BB$q_0$abBBB $\vdash$ BBB$q_1$bBBB $\vdash$ BBBb$q_1$ BBB

$\vdash$ BBB$q_2$bBBB $\vdash$ BB$q_3$BBBBB $\vdash$ BBB$q_0$BBBB $\vdash$BBBB$q_4$BBB

$q_4$ is final state so, string is accepted by turing machine.

**Q.8. Write Short notes on any two:** **(6.25)**

**Q.8. (a) Prove that a function f: $\Sigma^* \rightarrow \Sigma\ \Sigma^*$ is called polynomial-time computable if there is a polynomially bounded Turing Machine computing it.**

**Ans:** an NDTM accepts the language L if for each string w in L placed left-adjusted on the otherwise blank input tape there is a choice input c for M that leads to an accepting halt state. A NDTM M computes a partial function f : B* "→ B* if for each input string w for which f is defined, there is a sequence of moves by M that causes it to print f(w) on its output tape and enter a halt state and there is no choice input for which M prints an incorrect result. The oracle Turing machine (OTM), the multi-tape DTM or NDTM with a special oracle tape, is used to classify problems. Time on an OTM is the number of steps it takes, where one consultation of the oracle is one step, whereas space is the number of cells used on its work tapes not including the oracle tape

**Q.8. (b) Explain Classification of Problems with example.** **(6.25)**

**Ans:** A decision problem is a computational problem where the answer for every instance is either yes or no. An example of a decision problem is primality testing:

"Given a positive integer $n$, determine if $n$ is prime."



A decision problem is typically represented as the set of all instances for which the answer is *yes*. For example, primality testing can be represented as the infinite set

$L = \{2, 3, 5, 7, 11, ...\}$

In a search problem, the answers can be arbitrary strings. For example, factoring is a search problem where the instances are (string representations of) positive integers and the solutions are (string representations of) collections of primes.

A search problem is represented as a relation consisting of all the instance-solution pairs, called a search relation. For example, factoring can be represented as the relation

$R = \{(4, 2), (6, 2), (6, 3), (8, 2), (9, 3), (10, 2), (10, 5)...\}$

which consist of all pairs of numbers $(n, p)$, where $p$ is a nontrivial prime factor of $n$.

A counting problem asks for the number of solutions to a given search problem. For example, a counting problem associated with factoring is

"Given a positive integer $n$, count the number of nontrivial prime factors of $n$."

A counting problem can be represented by a function $f$ from $\{0, 1\}^*$ to the nonnegative integers. For a search relation $R$, the counting problem associated to $R$ is the function

$$f_R(x) = |\{y : R(x, y)\}|.$$

An optimization problem asks for finding a "best possible" solution among the set of all possible solutions to a search problem. One example is the maximum independent set problem:

"Given a graph $G$, find an independent set of $G$ of maximum size."

Optimization problems can be represented by their search relations.

In a function problem a single output (of a total function) is expected for every input, but the output is more complex than that of a decision problem, that is, it isn't just "yes" or "no". One of the most famous examples is the travelling salesman problem:

"Given a list of cities and the distances between each pair of cities, find the shortest possible route that visits each city exactly once and returns to the origin city."

It is an NP-hard problem in combinatorial optimization, important in operations research and theoretical computer science

**Q.8. (c) State and Prove Cook's Theorem.** **(6.25)**

**Ans: Defination:** The satisfiability problem (SAT) is the problem: Given a boolean expression:, is it satisfiable?

**Theorem:** SAT is NP-Complete.

**Proof: PART I: SAT ε NP.**

If the encoded expression E is of Length n, then the number of variables is [n/2]. Hence, for guessing a truth assignment t we can use multitape TM for E. The time taken by a multitape NTM M is O(n). The M evaluates the value of E for a truth assignment t.

This is done in $O(n^2)$ time. An equivalent single tape TM takes $O(n^4)$ time. Once an accepting truth assignment is found, M accepts E and M and halts. Thus we have found a polynomials time NTM for SAT. Hence SAT ε NP.

**PART II: POLYNOMIAL-TIME REDUCTION OF ANY L IN NP TO SAT.**

**1. Construction of NTM for L:** Let L be any language in NP. Then there exists a single-tape NTM M and a polynomial p(n) such that the time taken by M for an input of length n is at most p(n) along any branch. If M accepts an input w and |w| = n, then there exists a sequence of moves of M such that

- $\alpha_0$ is the initial ID of M with input w.
- $\alpha_0 \vdash \alpha_1 \vdash ....\vdash \alpha_k$, $k \leq P(n)$
- $\alpha_k$ is an ID with an accepting state.

**2. Representation of sequence of moves of M:** As the maximum number of steps on w is p(n) we need not bother about the contents beyond p(n) cells. We can write $\alpha_i$ as a sequence of $p(n) + 1$ symbols (one symbol for the state and the remaining symbols for the tape symbols. So $a_i = x_{i0} x_{i1} \ldots x_{ip(n)}$. If $\alpha_m$ is an accepting ID in the course of processing $w$ then we write $\alpha_0 \vdash .. \vdash \alpha_a \vdash a_m = \alpha_{i, p(n)}$.

**3. Representation of IDs in terms of Boolean Variables:** We define a boolean variable $y_{ijA}$ corresponding to $(i,j)$th entry in the i th ID. The variable $y_{ijA}$ represents the proposition that $x_{ij} = A$, where A is a state or tape symbol and $0 \le i, j \le p(n)$.

**4. Polynomial Reduction of M to SAT:** In order to check that the reduction of M to SAT is correct, we have to ensure the correctness of

(a) the initial ID.

(b) the accepting ID and

(c) the intermediate moves between successive IDs.

**(i) Simulation of initial ID:** $x_{00}$ must start with the initial state $q_0$ of M followed by the symbols of $w = a_1 a_2 \ldots a_n$ of length $n$ and ending with b's. The corresponding boolean expression S is defined as:

$$S = y_{00q0} \wedge y_{01a1} \wedge \ldots \wedge y_{on\,an} \wedge y_{on\,an+1} \ldots \wedge y_{0,p(n),n}$$

**(ii) Simulation of accepting ID:** $\alpha_{p(n)}$ is the accepting ID. If $p_1, p_2, \ldots p_k$ are the accepting states of M, then $\alpha_{p(n)}$ contains one of $p_i$'s $1 \le i \le k$ in any place j.

It is given by

$$F = F_0 \vee \ldots \vee F_{p(n)}$$

where

$$F_j = y_{p(n),j,p_1} \vee y_{p(n)j,p2} \vee \ldots \vee y_{p(n),n,pk}$$

**(iii) Simulation of Intermediate Moves:** We have to simulate valid moves $\alpha_i \vdash \alpha_{i+1}$ i = 0,1,2...p(n). corresponding to each move, we have to define a boolean variable $N_i$. Hence the entire sequence of IDs leading to acceptance of w is.

$$N = N_0 \wedge N_1 \wedge \ldots \wedge N_{p(n)-1}$$

**Formulation of $B_{ij}$:** When the state of $\alpha_i$ is none of $x_{i,j-1}$, $x_{ij}$, $x_{i,j+1}$, then the transition corresponding to $\alpha_{i+1}$ will not affect $x_{i,j+1}$. In this case $x_{i+1,j} = x_{ij}$.

- **Formulation of $A_{ij}$:** This step corresponds to the correctness of the $2 \times 3$ array

| | | |
|---|---|---|
| $x_{i,j-1}$ | $x_{ij}$ | $x_{i,j+1}$ |
| $x_{i+1,j-1}$ | $x_{i+1,j}$ | $x_{i+1,j+1}$ |

**Defination of $N_i$ and N:** We define $N_i$ and N by

$N_i = (A_{i0} \vee B_{i0}) \wedge (A_{i1} \vee B_{i1}) \wedge \ldots \wedge (A_i, p(n) \vee Bi, p(n))$

$N = N_0 \wedge N_1 \wedge N_2 \wedge \ldots \wedge N_{p(n)-1}$

**5. Completion of Proof:** Let $E_{M,w} = S \wedge N \wedge F$

We have seen that the time taken to write S and F are 0(p(n)) and the time taken for N is 0 ($P^2(n)$). Hence, the time taken to write $E_{M,w}$ its ($p^2(n)$.

Also M accepts w if and only if $E_{M,w}$ is satisfiable Hence the determinsitic multiple TM $M_1$ can convert W to a boolean expression $E_{M,w}$ in $O(p^2(n))$ time. An equivalent single tape TM taken $O(p^4(n))$ time. This proves the Part II, of the Cook's theorem, thus completing the proof of this theorem.

# FIRST TERM EXAMINATION [FEB. 2018]
# FOURTH SEMESTER [B.TECH]
# THEORY OF COMPUTATION
# [ETCS–206]

**Time : 1.5 hrs.** **M.M. : 30**

***Note:*** *Q.No.1 which is compulsory. Attempt any two questions from the rest.*

**Q.1. (a) Write down the regular expression for the following diagram.** **(2)**

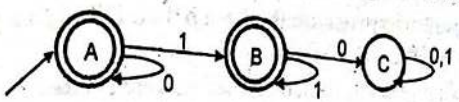



**Fig. 1.**

**Ans.** We can apply Arden's theorem :- We get the following equations for the $q_1, q_2, q_3$.

$$q_1 = q_1 0 + \wedge$$

$$q_2 = q_1 1 + q_2 1$$

$$q_3 = q_2 0 + q_3 (0+1)$$

By Applying Arden's Theorem to $q_1$ equation we get

$$q_1 = \wedge 0^* = 0^*$$

So,

$$q_2 = q_1 1 + q_2 1 = 0^* 1 + q_2 1$$

Therefore,

$$q_2 = (0^* 1) 1^*$$

As the final states are $q_1$ and $q_2$, we need not solve for $q_3$ :

$$q_1 + q_2 = 0^* + (0^* 1) 1^*$$

$$= 0^* (\wedge + 11^*)$$

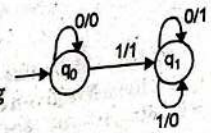
$$= 0^* 1^*.$$

**Q. 1. (b) Design a mealy machine to find 2's complement of given binary number which is processed from right to left instead of left to right.** **(2)**

**Ans.** Input alphabet: $\Sigma = \{0, 1\}$

Output alphabet: $\Gamma = \{0, 1\}$

Read from LSB

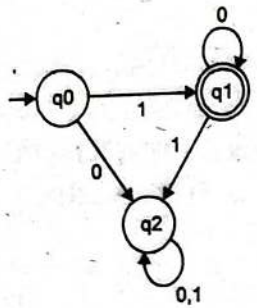



Keep the first from LSB as it is and toggle the remaining bits we will get 0101

Thus 2's complement of 1011 is 0101. The required Mealy machine will be-

**Q. 1. (c) Design a DFA accepting the binary equivalent of the elements of the set $\{2^n \mid n > 0\}$ and describe, in English language, the following regular expression** **(2)**
**(a)*b(a)*b(a)*b(a)*.**

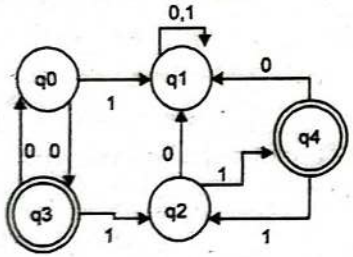
**Ans.** DFA for Accepting binary equivalent of $\{2^n \mid n >= 0\}$




**Q. 1. (d) Find all strings of length 4 or less in the regular set represented by the following regular expression (b a* + a b*) and Design a DFA for the regular expression (00)*(11)*. (2)**

**Ans.** String of length 4 or less in regular set represented by the following regular expression (ba* + ab*)*- baaa, abbb, ab,abab,ba,ba,ba,b ,a, abb,baa

DFA for (00)*(11)*-




**Q. 1. (e) Define regular grammar for the following DFA. (2)**

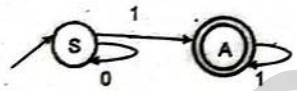



**Fig. 2.**

**Ans. Regular grammar :-**

Let G= ({S ,A }, {0,1},P,S ), Where P is given by

$S \longrightarrow 0S$ ,
$S \longrightarrow 1A$,
$S \longrightarrow 1$,
$A \longrightarrow 1A$
$A \longrightarrow 1$

**Q. 2. (a) Minimize the following DFA $M = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{a, b\}, \delta, q_0, [q_2, q_5])$ where δ is given as: $\delta(q_0, a) = q_0$, $\delta(q_0, b) = q_3$, $\delta(q_1, a) = q_2$, $\delta(q_1, b) = q_5$, $\delta(q_2, a) = q_2$, $\delta(q_2, b) = q_4$, $\delta(q_3, a) = q_0$, $\delta(q_3, b) = q_5$, $\delta(q_4, a) = q_0$, $\delta(q_4, b) = q_5$, $\delta(q_5, a) = q_1$, $\delta(q_5, b) = q_4$. (5)**

**Ans.** $\pi_i$'S are given below

$\pi_0 = \{\{q_6\}, \{q_0, q_1, q_2, q_3, q_4, q_5\}\}$
$\pi_1 = \{\{q_6\}, \{q_0, q_1, q_2, q_3, q_5\}, \{q_4\}\}$



$\pi_2 = \{\{q_6\}, \{q_4\}, \{q_0, q_1, q_3\}, \{q_2, q_5\}\}$
$\pi_3 = \{\{q_6\}, \{q_4\}, \{q_0\}, \{q_1\}, \{q_3\}, \{q_2, q_5\}\}$
$\pi_4 = \{\{q_6\}, \{q_4\}, \{q_0\}, \{q_1\}, \{q_3\}, \{q_2\}, \{q_5\}\}$

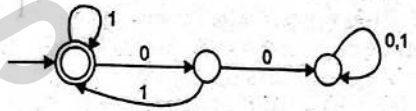
Here π = Q. The minimum state automation is simply the given automation.

**Q. 2. (b) Find the regular expression corresponding for the Figure 1 using Arden's Theorem. (5)**

**Ans.** Refer Q. 1. (a) of First Term Exam 2018.

**Q. 3. (a) Design a DFA for the following regular expression P = (1 + 01)*. Afterwards, convert that DFA in a way (i) It would accept complement of given regular expression ($P'$) (ii) It would accept reverse of given regular expression ($P^R$). (5)**

**Ans.** DFA for (1+01)*




**Q. 3. (b) Prove that $L = \{0^{n!n} \mid n \geq 0\}$ is not regular. (5)**

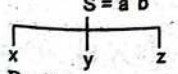
**Ans.** If A is a Regular language, then A has a Pumping Length 'P' such that any string 'S' where $|S| \geq P$ may be divided into 3 parts S = x y z such that the following conditions must be true:

(1) $x y^i z \in A$ for every $i \geq 0$
(2) $|y| > 0$
(3) $|xy| \leq P$

To prove that a language is not Regular using PUMPING LEMMA. Follow the below steps:

(We prove using Contradiction)
→ Assume that A is Regular
→ It has to have a Pumping Length (say P)
→ All string longer than P can be pumped $|S| \geq P$
→ Now find a string 'S' in A such that $|S| \geq P$
→ Divide S into x y z
→ Show that $x y^i z \notin A$ for some i
→ Then consider all ways that S can be divided into x y z
→ Show that none of these can satisfy all the 3 pumping conditions at the same time.
→ S cannot be pumped == Contradiction

Assume that A is Regular
Pumping length = P




$\Rightarrow$ S = aaaaaaabbbbbbb

P = 7. $xy^iz \Rightarrow xy^2z$

Case1: The y is in the 'a' part

$\underbrace{aa}_{x}\underbrace{aaaa}_{y}\underbrace{abbbbbbb}_{z}$ aa aaaaaaaa a bbbbbbb

$11 \neq 7$

$xy^iz \Rightarrow xy^2z$

Case 2: The y is the 'b' part

$\underbrace{aaaaaaa}_{x}\underbrace{bb}\ \underbrace{bbbb}_{y}\ \underbrace{b}_{z}$ aaaaaaa bb bbbb bbbb b

$7 \neq 11$

$xy^iz \Rightarrow xy^2z$

Case 3: The y is in the 'a' and 'b' part

$\underbrace{aaaaa}_{x}\ \underbrace{aabb}_{y}\ \underbrace{bbbbb}_{z}$ aaaaa aabbaabb bbbbbb

$|XY| \leq P, P = 7$

**Q. 4. (a) Reduce the following grammar G into Chomsky Normal Form (CNF) - {S → aAD, A → aB | bAB, B → b, D → a} Here, S, A, B and D are the variables; S is the start variable; and a and b are the terminals.** **(5)**

**Ans.** As there are no null productions or unit productions, we can proceed to step 1.

**Step 1** Let $G_1 = (V'_N, \{a, b, d\}, P_1, S)$, where $P_1$ and $V'_N$ are constructed as follows:

(i) $B \to b, D \to d$ are included in $P_1$.

(ii) $S \to aAD$ gives rise to $S \to C_aAD$ and $C_a \to a$.

$A \to aB$ gives rise to $A \to C_aB$.

$A \to bAB$ gives rise to $A \to C_bAB$ and $C_b \to b$.

$V'_N = \{S, A, B, D, C_a, C_b\}$.

**Step 2** $P_1$ consists of $S \to C_aAD, A \to C_aB \mid C_bAB, B \to b, D \to d, C_a \to a, C_b \to b$.

$A \to C_aB, B \to b, D \to d, C_a \to a, C_b \to b$ are added to $P_2$

$S \to C_aAD$ is replaced by $S \to C_aC_1$ and $C_1 \to AD$.

$A \to C_bAB$ is replaced by $A \to C_bC_2$ and $C_2 \to AB$.

Let

$G_2 = (\{S, A, B, D, C_a, C_b, C_1, C_2\}, \{a, b, d\}, P_2, S)$

where $P_2$ consists of $S \to C_aC_1, A \to C_aB \mid C_bC_2, C_1 \to AD, C_2 \to AB$,

$B \to b, D \to d, C_a \to a, C_b \to b$, $G_2$ is in CNF and equivalent to G.

**Q. 4. (b) Remove the Null production from the given grammar-{S → λ | aS | AB, A → λ | c, B → b}. Here, S, A and B are the variables; S is the start variables; a, b and c are the terminals and λ is the Null symbol.** **(5)**

**Ans. Step 1** Construction of the set W of all nullable variables:

$W_1 = \{A_1 \in V_N \mid A1 \to A \text{ is a production in G}\}$
$= \{A, B\}$

$W_2 = \{A, B\} \cup \{S\}$ as $S \to AB$ is a production with $\in W^*_i$.
$= \{S, A, B\}$

$W_3 = W_2 \cup \phi = W_2$

Thus,

$W = W_2 = \{S, A, B\}$

**Step 2** Construction of P':

(i) $B \to b$ is included in P'.

(ii) $S \to AB$ give rise to $S \to B, S \to AB$

(iii) $A \to c$ is included in P'.

Hence the required grammar without null productions is $G' = (\{S, A, B\}, \{a, b\}, P', S)$ where P' consists of

$S \to aS \mid AB \mid B, A \to c, B \to b$

# END TERM EXAMINATION [MAY-JUNE 2018]
# FOURTH SEMESTER [B.TECH]
# THEORY OF COMPUTATION
# [ETCS–206]

**Time : 3 hrs.** **M.M. : 75**

*Note: Q.No. 1 which is compulsory. Attempt five questions from the rest.*

**Q.1. (a) What is Finite Automation? Differentiate between DFA and NFA?** **(5)**

**Ans.** A finite-state automaton (FSA) is an abstract computing device composed of a finite number of states with zero or more labeled transitions between them. An FSA receives an input sequence of symbols, computes a finite series of actions, and halts in a configuration that indicates acceptance or rejection of the input. Finite Automata(FA) is the simplest machine to recognize patterns.

**A Finite Automata consists of the following :**

Q : Finite set of states.

Σ : set of Input Symbols.

q : Initial state.

F : set of Final States.

δ : Transition Function.

Formal specification of machine is

{Q, Σ, q, F, δ}.

FA is characterized into two types:

**(1) Deterministic Finite Automata (DFA)**

DFA consists of 5 tuples {Q, Σ, q, F, δ}.

Q : set of all states.

Σ : set of input symbols. ( Symbols which machine takes as input )

q : Initial state. ( Starting state of a machine )

F : set of final state.

δ : Transition Function, defined as $\delta : Q \times \Sigma \to Q$.

In a DFA, for a particular input character, the machine goes to one state only. A transition function is defined on every state for every input symbol. Also in DFA null (or ε) move is not allowed, i.e., DFA cannot change state without any input character.

**Nondeterministic Finite Automata(NFA)**

NFA is similar to DFA except following additional features:

1. Null (or ε) move is allowed i.e., it can move forward without reading symbols.

2. Ability to transmit to any number of states for a particular input.

However, these above features don't add any power to NFA. If we compare both in terms of power, both are equivalent.

Due to above additional features, NFA has a different transition function, rest is same as DFA.

δ : Transition Function

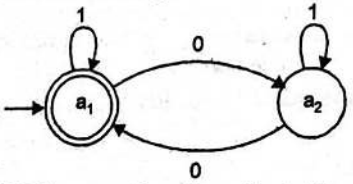
$\delta : Q \times (\Sigma \cup \in) \to 2^{\wedge} Q$.

As you can see in transition function is for any input including null (or ε), NFA can go to any state number of states.

| Deterministic Finite Automata | Non-Deterministic Finite Automata |
|---|---|
| 1. Transition function ($\delta : Q \times S \to Q$) | Transition function $\Delta : Q \times \Sigma \to P(Q)$. |
| 2 . DFA cannot use empty string transition | NFA can use empty string transition |
| 3. DFA requires more space | NFA requires less space. |
| 4. Backtracking is allowed in DFA | In NFA it may or may not be allowed |
| 5. DFA will reject the string if it end at other than accepting state. | If all of the branches of NFA dies or rejects the string, we can say that NFA reject the string. |

**Q. 1. (b) Construct a DFA over the alphabet {0, 3}, such that number of 0's in the string is always even.** **(5)**

**Ans.** DFA for even no. of 0's in a string:-

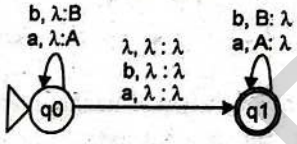



**Q. 1. (c) Construct a PDA accepting the set of all even-length palindromes over a,b by empty store,** **(5)**

**Ans.** PDA accepting set of all even length palindrome:-

The set of palindromes over {a, b}

A PDA is given that accepts the even length palindromes, i.e., $\{ww^R \mid w \in \{a, b\}^*\}$. The odd length palindromes over {a, b} are $\{wcw^R \mid w \in \{a, b\}^*\}$, where c can be either a or b. The two PDA's can be easily combined in the following PDA:




**Q. 1. (d) State Church's Hypothesis about computability of a machine.** **(5)**

**Ans. )Turing Church's Thesis:**

• As stated by Kleene:

Every effectively calculable function (effectively decidable predicate) is general recursiye.

• Any mechanical computation can be performed by a Turing Machine

• There is a TM-n corresponding to every computable problem

• We can model any mechanical computer with a TM

• The set of languages that can be decided by a TM is identical to the set of languages that can be decided by any mechanical computing machine

• If there is no TM that decides problem P, there is no algorithm that solves problem P.

**Q. 1. (e) Prove that graph coloring problem is NP-complete.** **(5)**

**Ans. NP-Completeness:-**



Problem Q is NP-complete means:

(1) it's in NP, and

(2) any other problem Q' in NP is poly-time reducible to Q.

NP-completeness proofs:

1. Prove Q is in NP

2. Select a known NP-complete problem Q'

3. Describe a poly-time algorithm that computes a function f mapping every instance x' of Q' to an instance x = f(x') of Q

4. Prove for all instances x' of Q': x' is YES instance of Q' iff x = f(x') is YES instance of Q.

A k-coloring of an undirected graph G is an assignment of colors to nodes such that each node is assigned a different color from all its neighbors, and at most k colors are used.

Theorem: 3-COLORING is NP-Complete.

Proof: (1) In NP: witness is a 3-coloring.

(2) Reduce 3-SAT to 3-COLORING.

(3) Given a 3-SAT formula of m clauses on n variables x1,x2,...,xn, we construct a graph G as follows. We have

(a) a vertex vi for each variable xi,

(b) a vertex vi' for the negation of each variable xi,

(c) 5 vertices j1-j5 for each clause j,

(d) 3 special vertices: T, F, R

We would like T, F, and R to be forced to different colors, so we will add edges between them to form a triangle. For the remaining nodes, and node that is colored the same color as T/F/R will be called colored TRUE/FALSE/RED, respectively.

We would like the edges to enforce the constraints on satisfying assignments.

Constraint: For all i, exactly one of vi and vi' is colored TRUE and one is colored FALSE.

Edges: for each i, form a triangle between vi, vi', and R.

Constraint: For each clause j, at least one of the literals in the clause is colored TRUE.

Edges: for each clause j, say = (xi or not(xj) or xk), we have the following gadget

```
vi ---j1
      |\
      | j3---j4
      |/   | \
Vj'---j2   |  T
           | /
vk --------j5
```

Claim: If each of vi, vj', and vk is colored TRUE or FALSE, then gadget is 3-colorable iff at least one of vi, vj', and vk is colored TRUE.

Proof: If vi, vj', and vk are all colored false, then we are forced to the following colors:

```
F ---j1
     |\
```

```
      | j3---- F
      |/    | \
F ---j2       |T
            | /
F ---------- R
```

But then j1, j2, j3 all must be colored different colors and NONE can be colored F, so there is no legal coloring.

The remainder of the proof considers the 7 possible combinations of coloring vi, vj', and vk such that at least one is colored TRUE and the rest are colored FALSE, and shows that a 3 coloring exists in each case.As an example, if vk is colored TRUE but vi and vj' are colored FALSE, we have the following legal 3-coloring:

```
F — T
   |\
   | F --- R
   |/    | \
F --- R     |T
        | /
T ---------- F
```

The other cases are similar and were presented in class. The construction takes polynomial time.

(4) Follows from the above arguments.

Thus 3-COLORING is NP-complete.

**Q. 2. (a) State and prove Kleen's Theorem.** **(6.25)**

**Ans. Kleen's Theorem, Part 1:** To each regular expression three corresponds a NFA.

**Strategy:** The corresponding NFA is the one constructed by Thompson's.

**Algorithm:** Proof that it is equivalent is by induction over regular expressions.

**Kleen's Theorem, part 2:** To each NFA there corresponds a regular expression.

**Strategy:** Proof by induction over the states of the NFA.

**Proof:** (Kleen's Theorem, part 2)

**Best Case:** (J = 1) we must be able to get from $p$ to $q$ without passing through any states; therefore either $p = q$ or we go directly in one transition from $p$ to $q$. The only strings that can be recognised in this case are $\in$ and single characters from $\Sigma$. The rules $R_1$ and $R_2$ give us a regular expression corresponding to these situations.

**Inductive Step: ($j = k + 1$)**

The induction hypothesis is that for some k such that $1 \le k \le N$, the Language $L(p, q, k)$ has a corresponding regular expression. Now we must prove that, based on this assumption, $L(p, q, k + 1)$ has a corresponding regular expression. Suppose the machine $L(p, q, k + 1)$ consumes some string; we must find a regular expression corresponding to $x$. Now suppose also that it passes through state $k$ on its way (if it doesn't, then $x$ is in $L(p, q, k)$, which has a corresponding regular expression by the induction hypothesis). Since the machine can "loop" on $K$ arbitrarily many times, we can split $x$ up into three substrings:

(a) Which moves the m/c from state $p$ to state $k$.

(b) Which causes the m/c to "Loop" on state $k$.

(c) Which moves the m/c from state $k$ to state $q$.



We note that while any of the above strings may cause us to start or finish at state k, none of them actually cause us to move through $k$. Thus we have:

$a \in L(p, k, k)$

$b \in L(k, k, k)$

$c \in L(k, q, k)$

By the induction hypothesis each of these have corresponding regular expressions, say A, B and C, respectively, and thus the regular expression for $x$ is A (B*) C.

Since our choice of $x$ was arbitrary, we have proved the theorem.

**Q. 2. (b) Construct a Mealy machine which is equivalent to the More machine given in Table 1:** **(6.25)**

**Table 1: Transition table for Mealy machine.**

| Present state | Next State a = 0 | Next State a = 1 | Output |
|---|---|---|---|
| $\to q_0$ | $q_1$ | $q_0$ | 1 |
| $q_1$ | $q_3$ | $q_2$ | 0 |
| $q_2$ | $q_2$ | $q_1$ | 1 |
| $q_3$ | $q_0$ | $q_3$ | 1 |

**Ans. Input** – Moore Machine

**Output** – Mealy Machine

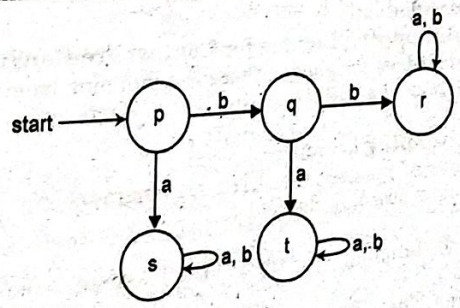
**Step 1** – Take a blank Mealy Machine transition table format.

**Step 2** – Copy all the Moore Machine transition states into this table format.

**Step 3** – Check the present states and their corresponding outputs in the Moore Machine state table; if for a state $Q_i$ output is m, copy it into the output columns of the Mealy Machine state table wherever $Q_i$ appears in the next state.

| Present state | a = 0 | output | a = 1 | output |
|---|---|---|---|---|
| $\to q_0$ | $q_1$ | 0 | $q_2$ | 1 |
| $q_1$ | $q_3$ | 1 | $q_2$ | 1 |
| $q_2$ | $q_2$ | 1 | $q_1$ | 0 |
| $q_3$ | $q_0$ | 1 | $q_3$ | 1 |

**Q. 3. (a) Minimize the following FDA:** **(6.25)**




**Ans. Minimization Algorithm for a DFA**

1. Identify and remove the unreachable states. These states are the states with no incoming transition, but only outgoing transition.

2. Draw two transition tables $T_1$ and $T_2$ where $T_1$ contains all rows which contains states from Q-F and $T_2$ contains all the states from set F.

3. All trap states are indistinguishable. So we remove all the trap states except for the one with the lowest index and replace the trap states reference with the only trap state left.

4. Find the similar rows from $T_1$ such that the states after transition on a given input are same for those states. From the set of similar rows remove all the rows from the table except the one with the smallest index and make corresponding changes to the table.

5. Repeat the above step till all redundant rows have been eliminated.

6. Repeat step 4 and 5 for table $T_2$

7. Now combine the tables to get the minimized DFA.

The given DFA is the minimum DFA.

**Q. 3. (b) Construct a DFA equivalent to the NFA M whose transition table is given in table 2.** **(6.25)**

**Table 2: Transition table of NFA M**

| Present State | 0 | 1 | 2 |
|---|---|---|---|
| $\rightarrow q_0$ | $q_1, q_4$ | $q_2$ | $q_2, q_3$ |
| $q_1$ | – | $q_4$ | |
| $q_2$ | $q_1, q_3$ | – | $q_2, q_4$ |
| $q_3$ | – | $q_4$ | – |
| $q_4$ | – | $q_1, q_3$ | – |

**Ans. Transition Table for DFA**

| Present State | 0 | 1 | 2 |
|---|---|---|---|
| $\rightarrow q_0$ | $\{q_1,q_4\}$ | $\{q_2\}$ | $\{q_2,q_3\}$ |
| $\{q_2\}$ | $\{q_1,q_3\}$ | {D} | $\{q_2,q_4\}$ |
| $\{q_1,q_4\}$ | $\{q_1,q_4\}$ | $\{q_1, q_3,q_4\}$ | {D} |
| $\{q_2,q_3\}$ | $\{q_1,q_3\}$ | $\{q_4\}$ | $\{q_2,q_4\}$ |
| $\{q_1,q_3\}$ | {D} | $\{q_4\}$ | {D} |
| $\{q_2,q_4\}$ | $\{q_1,q_3\}$ | $\{q_1,q_3\}$ | $\{q_2,q_4\}$ |
| $\{q_4\}$ | {D} | $\{q_1,q_3\}$ | {D} |
| $\{q_1, q_3,q_4\}$ | $\{q_1,q_4\}$ | $\{q_1, q_3,q_4\}$ | {D} |
| {D} | {D} | {D} | {D} |

**Note:- {D} is the dead state**

**Q. 4. (a) State and Pumping Lemma for Context Free languages.** **(6.25)**

**Ans.** If L is a context-free language,. there is a pumping length p such that any string $w \in L$ of length $\geq p$ can be written as $w = uvxyz$, where $vy \neq \varepsilon$, $|vxy| \leq p$, and for all $i \leq 0$, $uv^ixy^iz \varepsilon L$.

**Applications of Pumping Lemma**

Pumping lemma is used to check whether a grammar is context free or not. Let us take an example and show how it is checked.

**Problem**

To check whether the language $L = \{x^n y^n z^n \mid n \geq 1\}$ is context free or not.

**Solution.**

Let L is context free. Then, L must satisfy pumping lemma.

At first, choose a number n of the pumping lemma. Then, take z as $0^n1^n2^n$.

Break z into **uvwxy**, where

$|vwx| \leq n$ and $vx \neq \varepsilon$.

Hence vwx cannot involve both 0s and 2s, since the last 0 and the first 2 are at least (n+1) positions apart. There are two cases-



**Case 1– vwx** has no-2s. Then vx has only 0s and 1s. Then uwy, which would have to be in L, has n 2s, but fewer than n 0s or 1s.

**Case 2–** vwx has no 0s.

Here contradiction occurs.

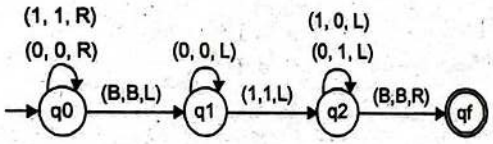
Hence, L is not a context-free language.

**Q. 4. (b) Construct a PDA to find the 2's- complement of binary number.**

**Ans.** 2's Complement of a binary number:- **(6.25)**

We have designed state transition diagram for 2's complement as follows:

1. Go to right of string using state q0
2. When BLANK is reached take one step left
3. Start scanning string from right to left
4. Pass all '0's and keep moving left
5. Pass single '1' and move left
6. Take complement of '1' and '0' after that.
7. When BLANK (in left) is reached move one step right and point to start of string.



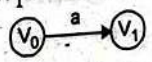

**Q. 5. (a) Construct a DFA equivalent to the grammer $s \rightarrow aS \mid bS \mid aA, A \rightarrow bB, B \rightarrow aC, C \rightarrow A$.** **(6.25)**

**Ans. Converting Regular Grammer to DFA**

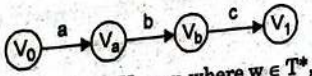
Assume that a regular grammar is given in its right-linear form, this grammar may be easily converted to a DFA. A right – linear grammar, defined by G = (V, T, S, P), may be converted to a DFA, defined by M = (Q, Σ, δ, $q_0$, F) by:

1. Create a state for each variable.
2. Convert each prodution rule into a transition.

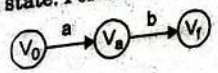
a. If the production rule is of the form $V_i \rightarrow aV_j$, where $a \in T$, add the transition $\delta(V_i,a) = V_j$ to M. For example, $V_0 \rightarrow aV_1$ becomes:




b. If the production rule of the form $V_i \rightarrow wV_j$, where $w \in T^*$, create a series of states which derive w and end in $V_j$. Add the states in between to Q. For example, $V_0 \rightarrow abcV_1$ becomes:




c. If the production rule is of the form $V_i \rightarrow w$, where $w \in T^*$, create a series of states which derive w and end in a final state. For example, $V_0 \rightarrow a$ becomes:



**Q. 5. (b) Reduce the grammer S → AB, A → a, B → C | b, C → D, D → E, E → a to Chomsky Normal Form.** **(6.25)**

**Ans.** Given CFG

$S \to AB$
$A \to a$
$B \to C/b.$
$C \to D$
$D \to E$
$E \to a$

Contain three UNIT productions

$B \to C$
$C \to D$
$D \to E$

Now to remove UNIT production B → C, we see if there exists a production whose left side has C and right side contains a terminal (i.e. C → a), but there is no such productions in G. Similar things holds for production C → D. Now we try to remove UNIT production D → E, because there is a production E → a. Therefore, eliminate D → E and introduced D → a, grammar becomes

$S \to AB$
$A \to a$
$B \to C/b$
$C \to D$
$D \to G$
$E \to a$

Now we can remove → D by using D → a, we get

$S \to AB$
$A \to a$
$B \to C/b$
$C \to a$
$D \to a$
$E \to a$

Similarly, we can remove B → C by using C → a, we obtain

$S \to AB$
$A \to a$
$B \to a/b$
$C \to a$
$D \to a$
$E \to a$

Now it can be easily seen that productions C → a, D → a, E → a, are useless because if we start deriving from S, these productions will never be used. Hence, eliminating them gives,

$S \to AB$
$A \to a$
$B \to a/b$

which is completely reduced grammar.



**Q. 6. (a) Construct a Turing machine that enumerates $\{0^n | n \geq 1\}$.** **(6.25)**

**Ans.** We require the following moves:

(a) If the leftmost symbol in the given input string w is 0, replace it by *x* and move right till we encounter a leftmost 1 in w. Change it to y and move backwards.

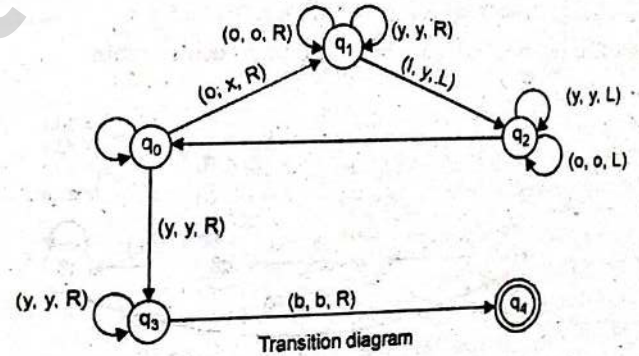
(b) Repeat (a) with the leftmost 0. If we move back and forth and no 0 or 1 remains, move to a final state.

(c) For strings not in the form $0^n 1^n$, the resulting state has to be nonfinal. Keeping these ideas in our mind, we construct a TM M as follows:–

where

$M = (Q, Z, r, \delta, q_0, F, b)$
$Q = \{q_0, q_1 q_2, q_3, q_f\}$
$F = \{q_f\}$
$S = \{0, 1\}$
$T = \{0, 1, x, y, b\}$

M accepts $\{0^n 1^n / n \geq 1\}$. The moves for 0011 and 010 are given below just to familiarize the moves of M to the reader:



Transition diagram

$q0011 \vdash xq_1, 001 + \vdash x0q_111 \vdash xq_20y1$
$\vdash q_2x0y1 \vdash xq_00y1 \vdash xx\,q_1y1 \vdash xx\,yq_1\,1$
$\vdash xx\,q_2yy \vdash xq_2xxy \vdash xx\,q_0yy \vdash xx\,yq_3\,y$
$\vdash xx\,yy\,q_3 \vdash xx\,yy\,q_3b \vdash xx\,yybq_4\,b$

Hence 0011 is accepted by M.

$q_0010 \vdash xq_110 \vdash q_2xy0 \vdash xq_0y_0 \vdash xy\,q_3\,0$

As $\delta(q_3, 0)$ is not defined, M habts. So 010 is not accepted by M.

**Q. 6. (b) Construct a Turing machine for language $\{a^n b^n c^n | n \geq 1\}$.** **(6.25)**

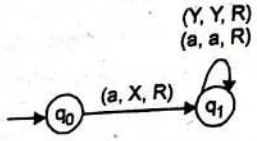
**Ans. Turing for language $\{a^n b^n c^n\}$**

1. Following steps:

a. Mark 'a' with 'X' and move towards unmarked 'b'

b. Move towards unmarked 'b' by passing all 'a's

c. To move towards unmarked 'b' also pass 'Y's if exist.

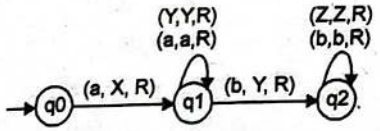



2. Following Steps:

a. Mark 'b' with 'Y' and move towards unmarked 'c'

b. Move towards unmarked 'c' by passing all 'b's

c. To move towards unmarked 'c' also pass all 'Z's if exist

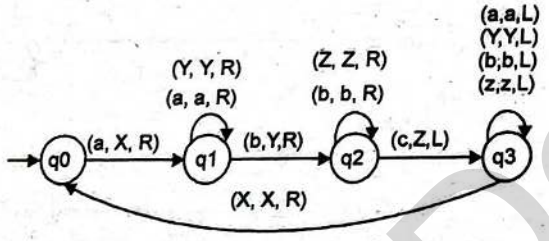



3. Following Steps:

a. Mark 'c' with 'Z' and move towards first 'X' (in left)

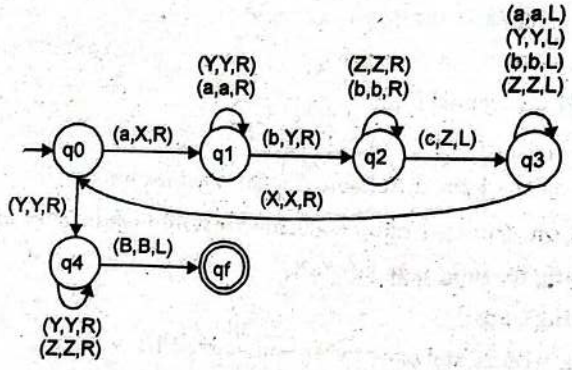
b. Move towards first 'X' by passing all 'Z's 'b's, 'Y's and 'a's

c. When 'X' is reached just move one step right by doing nothing.




4. To check all the 'a's, 'b's and 'c's are over add loops for checking 'Y'and 'Z' after we get 'X' followed by 'Y".

To reach final state (qf) just replace BLANK with BLANK and move either direction.






**Q. 7. (a) Explain Universal Turing Machine with help of an example.** (6.25)

**Ans.** A Turing machine is said to be universal Turing machine if it can accept:

- The input data, and
- An algorithm (description) for computing.

- This is precisely what a general purpose digital computer does. A digital computer accepts a program written in high level language. Thus, a general purpose Turing machine will be called a universal Turing machine if it is powerful enough to simulate the behavior of any digital computer, including any Turing machine itself.
- More precisely, a universal Turing machine can simulate the behavior of an arbitrary Turing machine over any set of input symbols. Thus, it is possible to create a single machine that can be used to compute any computable sequence. If this machine is supposed to be supplied with the tape on the beginning of which is written the input string of quintuple separated with some special symbol of some computing machine M, then the universal Turing machine U will compute the same strings as those by M.
- The model of a Universal Turing machine is considered to be a theoretical breakthrough that led to the concept of stored programmer computing device.
- Designing a general purpose Turing machine is a more complex task. Once the transition of Turing machine is defined, the machine is restricted to carrying out one particular type of computation.
- Digital computers, on the other hands, are general purpose machines that cannot be considered equivalent to general purpose digital computers until they are designed to be reprogrammed.
- By modifying our basic model of a Turing machine we can design a universal Turing machine. The modified Turing machine must have a large number of states for stimulating even a simple behavior. We modify our basic model by:
  - Increase the number of read/write heads
  - Increase the number of dimensions of input tape
  - Adding a special purpose memory
- All the above modification in the basic model of a Turing machine will almost speed up the operations of the machine can do.
- A number of ways can be used to explain to show that Turing machines are useful models of real computers. Anything that can be computed by a real computer can also be algorithm.
- A Turing machine is not very capable of handling it in a given finite amount of time. Also, Turing machines are not designed to receive unbounded input as many real programmers like word processors, operating system, and other system software.

**Q. 7. (b) Explain recursive and recursively enumerable languages and the relationship between them.** **(6.25)**

Ans. A recursively enumerable language is a formal language for which there exists a Turing machine (or other computable function) that will halt and accept when presented with any string in the language as input but may either halt and reject or loop forever when presented with a string not in the language. Contrast this to recursive languages, which require that the Turing machine halts in all cases, *e.g.* Halting Problem, Post correspondence problem.

**Different properties of Recursively Enumerable Language:**

1. If L, L 1 and L 2 are recursive languages, then so are L1 ∩ L2, L1L2, L*,L1) ∪ L 2 and L 1 - L 2.

2. If L, L 1 and L 2 are recursively enumerable languages, then so are L 1 ∪ L 2, LI L 2, L * and L 1 ∩ L2.

3. If Σ is an alphabet, L ⊆ Σ * is a recursively enumerable language, and Σ* – L is recursively enumerable, then L is recursive.

4. If S is an alphabet, L ⊆ Σ * is a recursively enumerable language, and Σ* – L is recursively enumerable, then L is recursive.

# MID TERM EXAMINATION [FEB. 2019]
# FOURTH SEMESTER [B.TECH]
# THEORY OF COMPUTATION [ETCS–206]

**Time : 1.30 hrs.** **M.M. : 30**

**Note :-** *Answer any three questions including Q.No.1. which is compulsory.*

**Q.1. Answer the following questions**

**(a) Define NFA with the help of an example.** **(2)**

**Ans.** In NDFA, for a particular input symbol, the machine can move to any combination of the states in the machine. In other words, the exact state to which the machine moves cannot be determined. Hence, it is called **Non-deterministic Automaton.** As it has finite number of states, the machine is called **Non-deterministic Finite Machine or Non-deterministic Finite Automaton.**

An NDFA can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where-

→ Q is a finite set of states,

→ Σ is a finite set of symbols called the alphabets,

→ δ is the transition function where $\delta : Q \times \Sigma \rightarrow 2^Q$

(Here the power set of Q ($2^Q$) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)

→ $q_0$ is the initial state from where any input is processed ($q_0 \in Q$).

→ F is a set of final state/states of Q ($F \subseteq Q$).

Graphical Representation of an NDFA: (same as DFA)

An NDFA is represented by digraphs called state diagram.

→ The vertices represent the states,

→ The arcs labeled with an input alphabet show the transitions,

→ The initial state is denoted by an empty single incoming arc.

→ The final state is indicated by double circles.

**Example**

Let a non-deterministic finite automaton be →

$Q = \{a, b, c\}$

$\Sigma = \{0, 1\}$

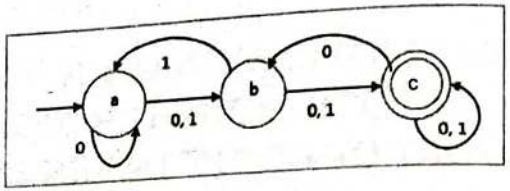
$q_0 = \{a\}$

$F = \{c\}$

The transition function δ as shown below–

| Present State | Next State for Input 0 | Next State for Input 1 |
|---|---|---|
| a | a, b | b |
| b | c | a, c |
| c | b,c | c |

Its graphical representation would be as follows–



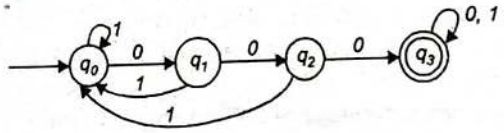

**Q. 1. (b) Construct a DFA to accept all strings over {0, 1} which contains three consecutive zeros.** **(2)**

**Ans.**

$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{0,,1\}$

Let $q_0$ be the initial state and $q_3$ be the final state. Then the required DFA will be




**Q. 1. (c) What is ambiguity in grammar? How is it removed?** **(2)**

**Ans.** If a context free grammar G has more than one derivation tree for string $w \in L(G)$, it is called an **ambiguous grammar**. There exist multiple right-most or left-most derivations for some string generated from that grammar.

**Problem**

To check whether the grammar G with production rules –

$X \rightarrow X+X \mid X*X \mid X \mid a$ is ambiguous or not.

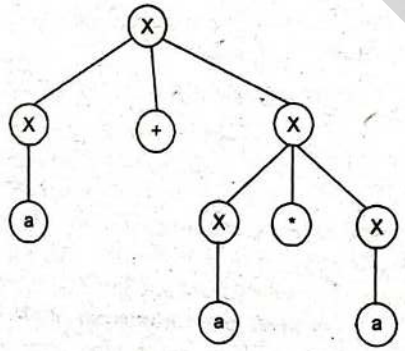
**Solution.**

Let's find out the derivation tree for the string "$a + a* a$". It has two leftmost derivations.

**Derivation 1 –**

$X \rightarrow X+X \rightarrow a+X \rightarrow a+X*X \rightarrow a+a*X \rightarrow a+a*a$

**Parse tree 1–**

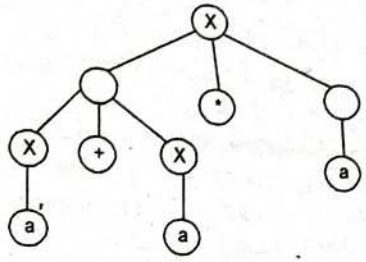



**Derivation 2 –**

$X \rightarrow X*X \rightarrow X+X*X \rightarrow a + X*X \rightarrow a+a*X \rightarrow a+a*a$



**Parse tree 2–**




As there are two parse trees for a single string "$a + a*a$", the grammar G is ambiguous.

**Removal of Ambiguity:**

1. Precedence of operators is not respected.

2. Sequence of identical operators can group either from left or from right .We would see two different parse tree for the above expression. Since addition is associative, it doesn't matter whether the group is from left or right, but to eliminate the ambiguity we must take one.

**Q. 1. (d) State three closure properties of Context Free Languages (CFL).** **(2)**

**Ans. Properties of Context Free Languages**

**Union :** If L1 and If L2 are two context free languages, their union L1 ∪ L2 will also be context free. For example,

$L1 = \{a^n b^n c^m \mid m >= 0 \text{ and } n >= 0\}$

and $L2 = \{a^n b^m c^m \mid n >= 0 \text{ and } m >= 0\}$

$L3 = L1 \cup L2 = \{a^n b^n c^m \cup a^n b^m c^m \mid n >= 0, m >= 0\}$ is also context free.

L1 says number of a's should be equal to number of b's and L2 says number of b's should be equal to number of c's. Their union says either of two conditions to be true. So it is also context free language.

**Note:** So CFL are closed under Union.

**Concatenation:** If L1 and If L2 are two context free languages, their concatenation L1 .L2 will also be context free. For example,

$L1 = \{a^n b^n \mid n >= 0\}$ and $L2 = \{c^m d^m \mid m >= 0\}$

$L3 = L1.L2 = \{a^n b^n c^m d^m \mid m >= 0 \text{ and } n >= 0\}$ is also context free.

L1 says number of a's should be equal to number of b's and L2 says number of c's should be equal to number of d's. Their concatenation says first number of a's should be equal to number of b's, then number of c's should be equal to number of d's. So, we can create a PDA which will first push for a's, pop for b's, push for c's then pop for d's. So it can be accepted by pushdown automata, hence context free.

**Note:** So CFL are closed under Concatenation.

**Kleene Closure:** If L1 is context free, its Kleene closure L1* will also be context free. For example,

$L1 = \{a^n b^n \mid n >= 0\}$

$L1^* = \{a^n b^n \mid n >= 0\}^*$ is also context free.

**Note:** So CFL are closed under Kleen Closure.

**Q. 1. (e) Differentiate between deterministic PDA and non-deterministic PDA.** **(2)**

**Ans. DPDA (Deterministic Pushdown Automata):**

1. In DPDA, the central symbol is known. Ex-abcba. Here, c denotes the central symbol and tells that after this symbol, pop operation needs to be performed.

2. Theie is only one move allowed in one situation.

3. The representation of a dpda is wcw^r.

**NDPDA/NPDA (Non-Deterministic Pushdown Automata)**

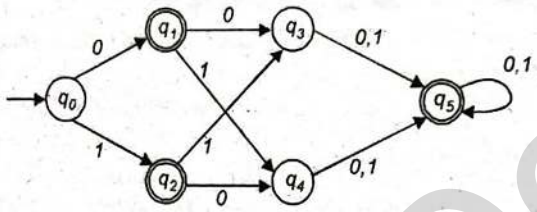
1. Central symbol is not known in case of a NPDA

2. There are multiple moves possible in one situation.

3. The representation of a NPDA is ww^r.

In DPDAs, only one move is possible when reading any input from any state but, in NPDAs, there can be multiple moves possible for an input symbol from a state. DPDAs are less powerful than NPDAs. There are Context Free Languages, such as the language of palindromes, that can be accepted by NPDAs but not by DPDAs.

**Q. 2. (a) Minimize the following DFA** **(5)**




**Ans.** Transition table is

| State | 0 | 1 |
|---|---|---|
| → $q_0$ | $q_1$ | $q_2$ |
| * $q_1$ | $q_3$ | $q_4$ |
| * $q_2$ | $q_4$ | $q_3$ |
| $q_3$ | $q_5$ | $q_5$ |
| $q_4$ | $q_5$ | $q_5$ |
| * $q_5$ | $q_5$ | $q_5$ |

Initially, we have two groups $\bar{x}_0 = \{(q_0, q_3, q_4), (q_1, q_2, q_5\}$. Now, we will compare $q_0$ and $q_3$ under '0' column we get $q_0$ and $q_5$ which lie in same class, and under '1' column we get $q_2$ and $q_5$ which again lie in same class. Hence $q_0$ and $q_3$ are 0– equivalent. Similarly $q_0$ and $q_4$ are 0– equivalent.



Similarly, we compare 2nd set $q_1$ and $q_2$. There transition lie in same class. Therefore $q_1$ and $q_2$ are 0-equivalent. On comparing $q_1$ and $q_5$, we find that they are not 0-equivalent. Hence, we get

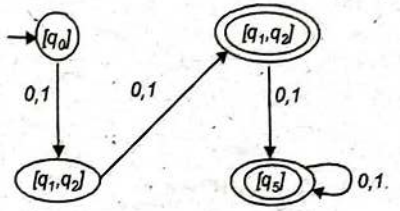
$$\bar{x}_1 = \{(q_0, q_3, q_4), (q_1, q_2), (q_5)\}$$

Now, the states $q_0$ and $q_3$ are not 1–equivalent because under '0' column of $q_0$, and $q_3$ we get $q_1$ and $q_5$ which donot lie in same class. But $q_3$ and $q_4$ are 1–equivalent as for both '0' and '1' column $q_3$ and $q_4$ give $q_5$. Hence, we get

$$\bar{x}_2 = \{(q_0), (q_3, q_4), (q_1, q_2), (q_5)\}$$

Now the states $q_3$ and $q_4$ are 2-equivalent. Similarly, $q_1$ and $q_2$ are 2-equivalent. Thus we cannot further partition states in $\bar{x}_2$. So, we get minimum automata as

| State | 0 | 1 |
|---|---|---|
| → [$q_0$] | [$q_1, q_2$] | [$q_1, q_2$] |
| * [$q_3, q_4$] | [$q_5$] | [$q_5$] |
| [$q_1, q_2$] | [$q_3, q_4$] | [$q_3, q_4$] |
| * [$q_5$] | [$q_5$] | [$q_5$] |




Initial state of minimized DFA is $q_0$ and final states are $[q_5]$ and $[q_3, q_4]$

**Q. 2. (b) (i) Find a regular expression for the set $\{a^n b^m$: (n + m) is odd}. (2.5)**

**(ii) Find a regular expression over {0, 1} for the all strings not ending in 10.** **(2.5)**

**Ans. (i) There are two cases:**

- n is even and m is odd: (aa)*b (bb)*:
- n is odd and m is even: a(aa)* (bb)*;

Thus a regular expression for the set $\{a^n b^m : (n + m)$ is odd$\}$ is (aa)*b (bb)* + a(aa)*(bb)*.

**(ii)** The cases for the desired regular expression are (0 + 1)* 00, (0 + 1)* 01, and (0 + 1)*11. The regular expression over {0, 1} for the all strings not ending in 10 is (0 + 1)* (00 + 01 + 11). **(5)**

**Q. 3. (a) Prove that the following grammar is ambiguous.**

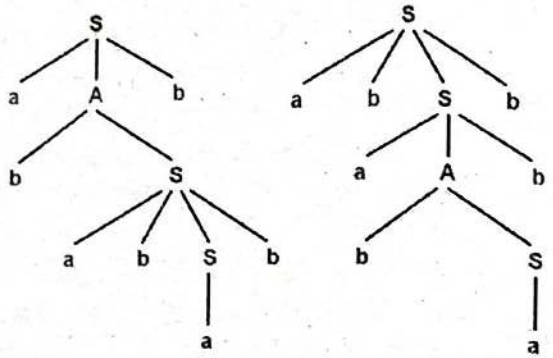
**S → a | aAB | abSb,**

**S → aAAb | bS**

**Ans.** Let us consider a string w generated by the given grammar-

w = ababbb

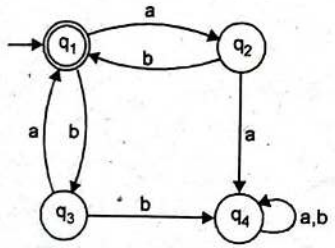
Now, let us draw parse trees for this string w.




Parse tree-01 Parse tree-02

Since two different parse trees exist for string w, therefore the given grammar is ambiguous.

**Q. 3. (b) Construct the regular expression for the DFA shown below:** (5)




**Ans.** We form the equations.

$$q_1 = q_2b + q_3a + \epsilon$$
$$q_2 = q_1a$$
$$q_3 = q_1b$$
$$q_4 = q_2a + q_3b + q_4a + q_4b$$

Putting $q_3$ and $q_2$ in $q_1$ as

$$q_1 = q_1ab + q_1ba + \epsilon \quad ...(1)$$

Using Arden's theorem in equation (1), we get

$$q_1 = \epsilon(ab + ba)^*$$
$$q_1 = (ab + ba)^*$$

So, the required regular expression is $(ab + ba)^*$

**Q. 4. (a) Construct a PDA (Pushdown Automation) to accept the language $L = \{a^nb^nc^md^m \mid m, n >= 1\}$ by empty stack and by final state** (5)

**Ans. By empty stack** Construct a PDA

where

$$M' = (Q', \Sigma', \Gamma, \delta', q_0', z_0', F')$$

$$Q' = \{q_0'\}$$



$$\Sigma' = T = \{a,b\}$$
$$\Gamma' = Z \cup T = \{S,A, a, b\}$$
$$z_0' = S$$

and $\delta'$ is defined as follows

$$\delta'(q_0', \lambda, S) = \{(q_0', bSa), (q_0', bAa)\}$$
$$\delta'(q_0', \lambda, A) = \{(q_0', aAb), (q_0', ab)\}$$
$$\delta'(q_0', a,a) = \{(q_0', \lambda)\}$$
$$\delta'(q_0', b, b) = \{(q_0', \lambda)$$

It is easily seen that $L = N(M')$

**By final state**

Construct a PDA

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

where

$$Q = Q' \cup \{q_0, q_j\}$$
$$= \{q_0', q_0, q_j\}$$
$$\Sigma = \Sigma' = \{a, b\}$$
$$\Gamma = \Gamma' \cup \{z_0\} = \{S, A, a, b, z_0\}$$
$$F = \{q_j\}$$

and $\delta$ is defined as follows:

$$\delta(q_0, \lambda, z_0) = \{(q_0', z_0z_0')\}$$
$$\delta(q_0', \lambda, S) = \{(q_0', bSa)\ (q_0', bAa)\}$$
$$\delta(q_0', \lambda, A) = \{(q_0', aAb)\ (q_0', ab)\}$$
$$\delta(q_0', a, a) = \{(q_0', \lambda)\}$$
$$\delta(q_0', b, b) = \{(q_0', \lambda)\}$$
$$\delta(q_0', \lambda, z_0) = \{(q_j, z_0)\}$$

It is easily seen that $L = T(M)$.

**Q. 4. (b) Eliminate useless symbols and productions from $G = (N, T, P, S)$ where $N = \{N, A,B,C\}$ and $T = \{a,b\}$ and $P = \{S \to aS/A/C, A \to a, B \to aa, C \to aCb\}$.** (5)

**Ans.** We identify the set of variables that can lead to a terminal string.

$$A \Rightarrow a$$
$$B \Rightarrow aa$$
$$S \Rightarrow A \Rightarrow a$$

$\therefore$ The variables S, *A*, *B* are of this type. Also *C* cannot lead to a terminal string. Removing *C* and its corresponding productions, we are lead to the grammar $G_1$ with variables.

$N_1 = (S, A, B]$ and terminals $T = [a]$ and productions

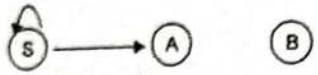
$$S \to aS/A,$$
$$A \to a$$
$$B \to aa.$$

**To eliminate the variables that cannot be reached from the start variable**

A dependency graph has its vertices labeled with variables, with an edge between vertices $E$ and F if and only if there is a production of the form

$$E \rightarrow xFy.$$

A dependency graph for $N_1$ is shown below.




From the graph it is clear that $B$ is useless.

Removing $B$ and its corresponding productions we obtain the final grammar $G_1$ as

$$G_1 = (N_1, S, T, P_1) \text{ where}$$
$$N_1 = \{S, A\}$$
$$T = \{a\}$$
$$P_1 = \{S \rightarrow aS/A, A \rightarrow a\}$$

# END TERM EXAMINATION [MAY. 2019]
# FOURTH SEMESTER [B.TECH]
# THEORY OF COMPUTATION [ETCS–206]

**Time : 3 hrs.** **M.M. : 75**

**Note :-** *Attempt all questions as directed. Internal choice is indicated.*

**Q.1. (a) Construct a Mealy machine equivalent to the Moore machine given in table.** **(5)**

| Present State | Next State a = 0 | Next State b = 1 | Output |
|---|---|---|---|
| → $q_0$ | $q_1$ | $q_2$ | 0 |
| $q_1$ | $q_3$ | $q_2$ | 1 |
| $q_2$ | $q_2$ | $q_1$ | 0 |
| $q_3$ | $q_0$ | $q_3$ | 1 |

**Ans.** The output function $\lambda'$ can be obtained using rule.

$$\lambda'(q, a) = \lambda(\delta(q, a))$$

Now, we will obtain for every transition corresponding to input

$$\lambda'(q_0,0) = \lambda(\delta(q_0, 0)) = \lambda(q_1) = 1$$
$$\lambda'(q_0,1) = \lambda(\delta(q_0, 1)) = \lambda(q_2) = 0$$
$$\lambda'(q_1,0) = \lambda(\delta(q_1, 0)) = \lambda(q_3) = 1$$
$$\lambda'(q_1,1) = \lambda(\delta(q_1, 1)) = \lambda(q_2) = 0$$
$$\lambda'(q_2,0) = \lambda(\delta(q_2, 0)) = \lambda(q_2) = 0$$
$$\lambda'(q_2,1) = \lambda(\delta(q_2, 1)) = \lambda(q_1) = 1$$
$$\lambda'(q_3,0) = \lambda(\delta(q_3, 0)) = \lambda(q_0) = 0$$
$$\lambda'(q_3,1) = \lambda(\delta(q_3, 1)) = \lambda(q_3) = 1$$

Hence the transition table for equivalent mealy machine is

| Present State | Next state | | | |
|---|---|---|---|---|
| | Input 0 | | Input 1 | |
| | State | O/P | State | O/P |
| $q_0$ | $q_1$ | 1 | $q_2$ | 0 |
| $q_1$ | $q_3$ | 1 | $q_2$ | 0 |
| $q_2$ | $q_2$ | 0 | $q_1$ | 1 |
| $q_3$ | $q_0$ | 0 | $q_3$ | 1 |

**Q. 1. (b) Describe the Chomsky classification of language with example.** **(5)**

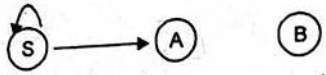
**Ans.** According to Noam Chomsky, there are four types of grammars "Type 0, Type1, Type 2, and Type 3." The following table shows how they differ from each other

**To eliminate the variables that cannot be reached from the start variable**

A dependency graph has its vertices labeled with variables, with an edge between vertices $E$ and F if and only if there is a production of the form

$$E \rightarrow xFy.$$

A dependency graph for $N_1$ is shown below.




From the graph it is clear that $B$ is useless.

Removing $B$ and its corresponding productions we obtain the final grammar $G_1$ as

$$G_1 = (N_1, S, T, P_1) \text{ where}$$
$$N_1 = \{S, A\}$$
$$T = \{a\}$$
$$P_1 = (S \rightarrow aS/A, A \rightarrow a\}$$

# END TERM EXAMINATION [MAY. 2019]
# FOURTH SEMESTER [B.TECH]
# THEORY OF COMPUTATION [ETCS–206]

**Time : 3 hrs.** **M.M. : 75**

**Note :-** *Attempt all questions as directed. Internal choice is indicated.*

**Q.1. (a) Construct a Mealy machine equivalent to the Moore machine given in table.** **(5)**

| Present State | Next State a = 0 | Next State b = 1 | Output |
|---|---|---|---|
| $\rightarrow q_0$ | $q_1$ | $q_2$ | 0 |
| $q_1$ | $q_3$ | $q_2$ | 1 |
| $q_2$ | $q_2$ | $q_1$ | 0 |
| $q_3$ | $q_0$ | $q_3$ | 1 |

**Ans.** The output function $\lambda'$ can be obtained using rule.

$$\lambda'(q, a) = \lambda(\delta(q, a))$$

Now, we will obtain for every transition corresponding to input

$$\lambda'(q_0,0) = \lambda(\delta(q_0, 0)) = \lambda(q_1) = 1$$
$$\lambda'(q_0,1) = \lambda(\delta(q_0, 1)) = \lambda(q_2) = 0$$
$$\lambda'(q_1,0) = \lambda(\delta(q_1, 0)) = \lambda(q_3) = 1$$
$$\lambda'(q_1,1) = \lambda(\delta(q_1, 1)) = \lambda(q_2) = 0$$
$$\lambda'(q_2,0) = \lambda(\delta(q_2, 0)) = \lambda(q_2) = 0$$
$$\lambda'(q_2,1) = \lambda(\delta(q_2, 1)) = \lambda(q_1) = 1$$
$$\lambda'(q_3,0) = \lambda(\delta(q_3, 0)) = \lambda(q_0) = 0$$
$$\lambda'(q_3,1) = \lambda(\delta(q_3, 1)) = \lambda(q_3) = 1$$

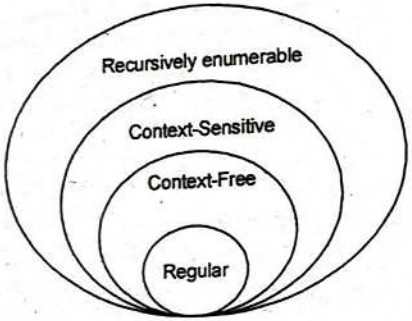
Hence the transition table for equivalent mealy machine is

| Present State | Next state | | | |
|---|---|---|---|---|
| | Input 0 | | Input 1 | |
| | State | O/P | State | O/P |
| $q_0$ | $q_1$ | 1 | $q_2$ | 0 |
| $q_1$ | $q_3$ | 1 | $q_2$ | 0 |
| $q_2$ | $q_2$ | 0 | $q_1$ | 1 |
| $q_3$ | $q_0$ | 0 | $q_3$ | 1 |

**Q. 1. (b) Describe the Chomsky classification of language with example.** **(5)**

**Ans.** According to Noam Chomsky, there are four types of grammars "Type 0, Type1, Type 2, and Type 3." The following table shows how they differ from each other

| Grammar Type | Grammar Accepted | Language Accepted | Automaton |
|---|---|---|---|
| Type 0 | Unrestricted grammar | Recursively enumerable language | Turing Machine |
| Type 1 | Context-sensitive grammar | Context-sensitive language | Linear- bounded automaton |
| Type 2 | Context-free grammar | Context-free language | Pushdown automaton |
| Type 3 | Regular grammar | Regular language | Finite state automaton |




**Type - 3 Grammar**

**Type-3 grammars** generate regular languages. Type-3 grammars must have a single non-terminal on the left-hand side and a right-hand side consisting of a single terminal or single terminal followed by a single non-terminal.

The productions must be in the form $X \to a$ or $X \to aY$, where $X, Y \in N$ (Non terminal) and $a \in T$ (Terminal). The rule $S \to \varepsilon$ is allowed if S does not appear on the right side of any rule. **Example**

$X \to \varepsilon$

$X \to a$

$X \to aY$

**Type - 2 Grammar**

**Type-2 grammars** generate context-free languages.

The productions must be in the form $A \to \gamma$ where $A \in N$ (Non terminal) and $\gamma \in (T \cup N)^*$ (String of terminals and non-terminals). These languages generated by these grammars are be recognized by a non-deterministic pushdown automaton.

**Example**

$S \to Xa$

$X \to a$

$X \to aX$

$X \to abc$

$X \to \epsilon$



**Type - 1 Grammar**

**Type-1 grammars** generate context-sensitive languages. The productions must be in the form $\alpha A \beta \to \alpha \gamma \beta$, where $A \in N$ (Non-terminal) and $\alpha, \beta, \gamma \in (T \cup N)^*$ (Strings of terminals and non-terminals)

The strings $\alpha$ and $\beta$ may be empty, but $\gamma$ must be non-empty.

The rule $S \to \epsilon$ is allowed if S does not appear on the right side of any rule. The languages generated by these grammars are recognized by a linear bounded automaton.

**Example**

$AB \to AbBc$ $A \to bcA$ $B \to b$

**Type - 0 Grammar**

**Type-0 grammars** generate recursively enumerable languages. The productions have no restrictions. They are any phase structure grammar including all formal grammars. They generate the languages that are recognized by a Turing machine.

The productions can be in the form of $\alpha \to \beta$ where $\alpha$ is a string of terminals and non-terminals with at least one non-terminal and $\alpha$ cannot be null. $\beta$ is a string of terminals and non-terminals.

**Example**

$S \to ACaB$ $Bc \to acB$ $CB \to DB$ $aD \to Db$

**Q. 1. (c) Draw the DFA equivalent to the CFG with productions** (5)

**$P = \{S \to 0S \mid 1A, A \to 1A \mid 0B, B \to 1B \mid \Lambda$. Here $\Lambda$ is used for null.**

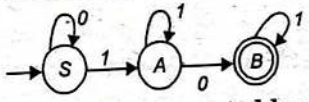
**Ans.** $P = \{S \to 0S/1A, A \to 1A/0B, B \to 1B/\Lambda)$

The transition for automata are obtained as

(i) For every production $A \to aB$ make $\delta(A, a) = B$

(ii) For every production $A \to a$ make $\delta(A, a)$ = final state

(iii) For every product $A \to \wedge$ make $\delta(A, \epsilon) = A$ and A will be final state.




**Q.1. (d) Prove that every language accepted by a multiple tape TM is acceptable by some single-tape TM.** (5)

**Ans. Multi-tape Turing machines:**

1. A multi-tape Turing machine is like an ordinary Turing machine with several tapes.

2. Each tape has its own head for reading and writing.

3. Initially, the input is written on the first tape, and all the other tapes are blank, with each head at the beginning of the corresponding tape.

4. The transition function is changed to allow for reading, writing, and moving the heads on some or all of the tapes simultaneously.

Formally, the transition function of the Multi-tape Turing machine is represented as

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k,$$

where k is the number of tapes. The expression

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$$

It means that, if the machine is in state $q_i$, and heads 1 through k are reading symbols $a_1$ through $a_k$, the machine goes to state $q_j$, writes symbols $b_1$ through $b_k$, and directs each head to move left or right, or to stay put, as specified.

**Theorem:** Every multi-tape Turing machine has an equivalent single-tape Turing machine.

**Proof:** We show how to convert a multi-tape TM M to an equivalent single-tape TM S. The key idea is to show how to simulate M with S.

Say that M has k tapes. Then S simulates the effect of k tapes by storing their information on its single tape. It uses the new symbol # as a delimiter to separate the contents of the different tapes. In addition to the contents of these tapes, S must keep track of the locations of the heads. It does so by writing a tape symbol with a dot above it to nark the place where the head on that would be. Think of these as "virtual" tapes and heads. The "dotted" tape symbols are simply new symbols that have been added to the tape alphabet. The following figure illustrates how one tape can be used to represent three tapes.

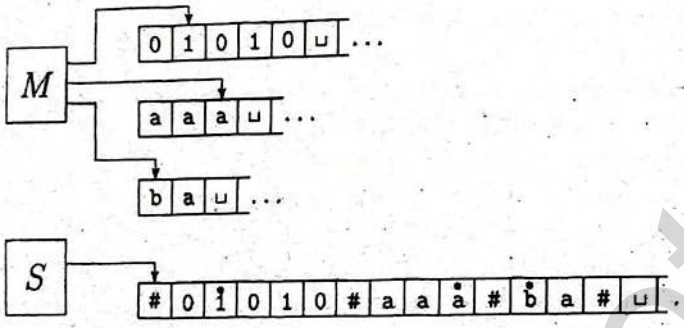



**Fig. Simulation of Multi-tape TM M to single-tape TM S.**

For a 3-tape TM, a transition will look like

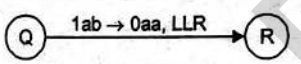



**Fig. Transition in Multi-tape TM**

If you are in state Q and see 1 on $tape_1$, a on $tape_2$ and b on $tape_3$,

- Write 0 on $tape_1$, a on $tape_2$ and a on $tape_3$
- Move $head_1$ left, $head_2$ left and $head_3$ right
- Go to state R."

$$S = \text{"On input } w = w_1 \ldots w_n\text{:}$$

1.First S puts its tape into the format that represents all k tapes of M. The formatted tape contains

$$\#\dot{w}_1 w_2 \ \ldots \ w_n \# \dot{\sqcup} \# \ \dot{\sqcup} \# \ldots\ldots \#$$

2.To simulate a single move, S scans its tape from the first #, which marks the left-hand end, to the (k + 1)st #, which marks the right-hand end, in order to determine the symbols under the virtual heads. The S makes a second pass to update the tapes according to the way that M's transition function dictates.



3.If at any point S moves one of the virtual heads to the right onto a '#', the action signifies that M has moved the corresponding head onto the previously unread blank portion of that tape. So S writes a blank symbol on this tape cell and shifts the tape contents, from this cell until the rightmost #, one unit to the right. Then it continues the simulation as before.

**Q. 1. (e) Prove that if there is a polynomial time reduction from $P_1$ to $P_2$ and if $P_2$ is in P then $P_1$ will be in P.** **(5)**

**Ans. Polynomial Reducibility**

**A problem $P_1$** is polynomial reducible **to** $P_2$ if there exists a polynomial time transformation from $P_1$ to $P_2$. We denote this as $P_1 \propto P_2$. In other words, if we have a problem $P_1$, then in polynomial time we can make a mapping so that a solution to $P_2$ will solve problem $P_1$ if we perform the polynomial time mapping.

This means $P_2$ is "at least as hard" as $P_1$

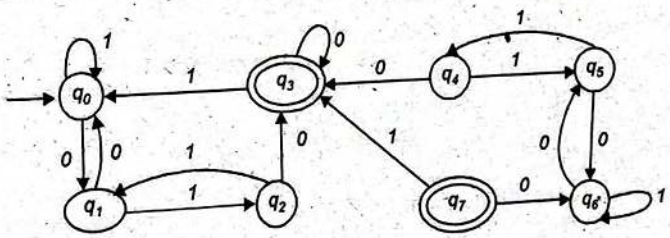
→ E.g. if $P_1$ was impossible to solve, so is $P_2$

→ E.g. if we can prove $P_1$ requires exponential time to solve, then so does $P_2$

**Q. 2. (a) Minimize the following DFA** **(6.25)**

| Q/Σ | 0 | 1 |
|---|---|---|
| → $q_0$ | $q_1$ | $q_0$ |
| $q_1$ | $q_0$ | $q_2$ |
| $q_2$ | $q_3$ | $q_1$ |
| ($q_3$) | $q_3$ | $q_0$ |
| $q_4$ | $q_3$ | $q_5$ |
| $q_5$ | $q_6$ | $q_4$ |
| $q_6$ | $q_5$ | $q_6$ |
| ($q_7$) | $q_6$ | $q_3$ |

**Ans.** The transition diagram of given DFA is-




From the diagram it is clear that the states $q_4$, $q_5$, $q_6$, $q_7$ are unreachable from initial state and hence we can ignore them. Now, initially we get two groups

$$\bar{x}_0 = \{[q_0, q_1, q_2], [q_3]\}$$

On comparing $q_0$ and $q_1$ under '0' column we get $q_1$ and $q_0$ respectively which lie in same class, and under '1' column they again lie in same class. Therefore $q_0$ and $q_1$ are 0-equivalent.

Similarly, on comparing $q_0$ and $q_3$, we find that their transition for '0' lie in different class. Hence they are not 0-equivalent

Therefore, we get

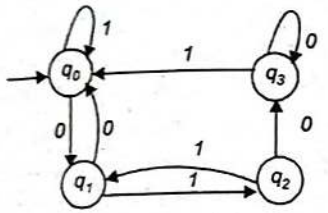
$$\bar{x}_1 = \{\{q_0, q_1\}, \{q_2\}, \{q_3\}\}$$

Now, the states $q_0 < q_1$ are not 1-equivalent because their transition under '1' lie in different classes.

$$\bar{x}_2 = \{\{q_0\}, \{q_1\}, \{q_2\}, \{q_3\}\}$$

We cannot further partition the states. So the minimized DFA is:

| State | 0 | 1 |
|---|---|---|
| → $q_0$ | $q_1$ | $q_0$ |
| $q_1$ | $q_0$ | $q_2$ |
| $q_2$ | $q_3$ | $q_1$ |
| * $q_3$ | $q_3$ | $q_0$ |




**Q. 2. (b) Prove that for any input string $x$ and $y$, $\delta(q, xy) = \delta(\delta(q, x), y)$. Where $\delta$ is the transition function of a DFA.** **(6.25)**

**Ans. Theorem 2:** For any state q of Q and any strings x and y over Σ for a DFA <Q, Σ,$q_0$,δ,A>,

$$\delta^*(q, xy) = \delta^*(\delta^*(q, x), y).$$

**Proof :** This is going to be proven by induction on string y. That is the statement to be proven is the following:

For an arbitrary fixed string x, $\delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$ holds for any arbitrary string y.

First let us review the recursive definition of $\Sigma^*$.

**Recursive definition of $\Sigma^*$:**

**Basis Clause:** $\Lambda \in \Sigma^*$.

**Inductive Clause:** If $x \in \Sigma^*$ and $a \in \Sigma$, then $xa \in \Sigma^*$

**Extremal Clause:** Nothing is in $\Sigma^*$ unless it is obtained from the above two clauses.

Now the proof of the theorem.

**Basis Step:** If $y = \Lambda$. then $\delta^*(q, xy) = \delta^*(q, x\Lambda) = \delta^*(q, x)$.

Also $\delta^*(\delta^*(q, x), y) = \delta^*(\delta^*(q, x)\Lambda = \delta^*(q, x)$ by the definition of $\delta^*$. Hence the theorem holds for $y = \Lambda$.

**Inductive Step:** Assume that $\delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$ holds for an arbitrary string y. This is the induction hypothesis.

We are going to prove that $\delta^*(q, xya) = \delta^*(\delta^*(q, x), ya)$ for any arbitrary symbol a of Σ.

$\delta^*(q, xya) = \delta(\delta^*(q, xy), a)$ by the definition of $\delta^*$

$= \delta(\delta^*(\delta^*(q, x), y), a)$ by the induction hypothesis.

$= \delta^*(\delta^*(q, x), ya)$ by the definition of $\delta^*$.

Thus the theorem has been proven.



**OR**

**Q. 3. (a) Draw the DFA for regular expression 00*11(1 1)*00* and find the complement and reverse of the DFA.** **(6.25)**

**Ans.** The DFA for the regular expression 00*11(11)*00* is:

Q = {A,B,C,D,E,F,G,H}

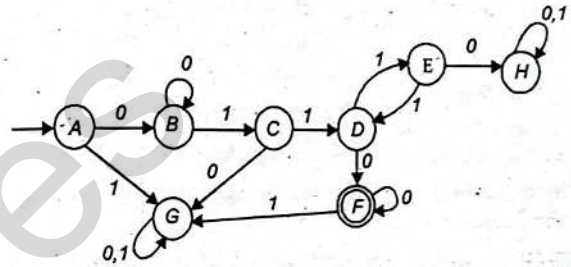
Σ = {0,1}

$Q_0$ = {A}

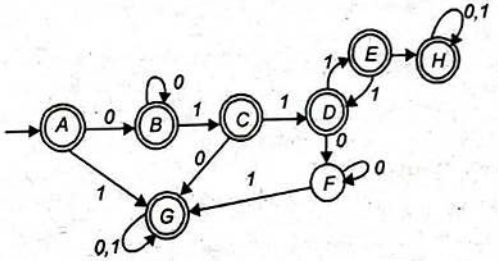
F = {F}

G and H are dead states.

DFA For regular expression




Complement of DFA :- Convert the non-final states to final states and final states to non-final states.

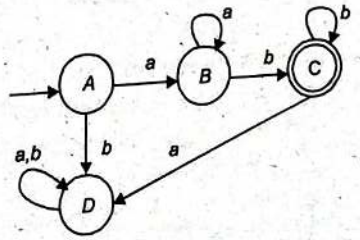



**Q. 3. (b) Draw the DFA for L = {$a^n b^m$ | n,m ≥ 1} and if L is not regular language then prove it using pumping lemma.** **(6.25)**

**Ans.**

L = {$a^n b^m$/n,m ≥ 1}

= {ab, aab, abb, aabb, aaabb....}




Since we can make a DFA of the required language ,it is a regular language.

**Q. 4. (a) Simplify the CFG G = ({S, A, B, C, D, E}, {a, b, c}, P, S}) where P = {S → ABC/aB, A → aA/a, B → b/C/D, D → a/b, E → c/^} and convert the simplified CFG into CNF. Here ^ is used for null.** **(6.25)**

**Ans.** P = {S → AB/aB; A → aA/a, B → b/c/D, D → a/b, E → c/^}

**Step 1:** Elimination of null and unit production

(i) On eliminating E → ∧, we get the production

S → AB/aB
A → aA/a
B → b/c/D
D → a/b
E → c

(ii) On eliminating B → D

S → AB/aB
A → aA/a
B → b/c/a
E → c

(iii) Removing unreachable productions (E → c)

S → AB/aB
A → aA/a
B → b/c/a

**Step 2:** Forming productions either of form A → a or A → BC

S → AB
S → XB
A → XA
A → a
B → b/c/a
X → a

So, the required CNF is

S → AB/XB
A → XA/a
B → b/c/a
X → a

**Q. 4. (b) Construct tbe PDA for odd length palindrome over {a, b}.** **(6.25)**

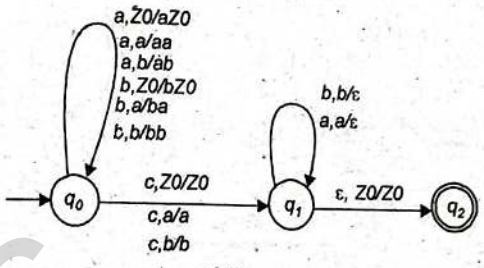
**Ans.**

$\delta(q_0, a, Z_0) = \{(q_0, aZ_0), (q_1, Z_0)\}$
$\delta(q_0, b, Z_0) = \{(q_0, bZ_0), (q_1, Z_0)\}$
$\delta(q_0, a, a) = \{(q_0, aa), (q_1, a)\}$
$\delta(q_0, b, b) = \{(q_0, bb), (q_1, b)\}$
$\delta(q_0, a, b) = \{(q_0, ab), (q_1, b)\}$
$\delta(q_0, b, a) = \{(q_0, ba), (q_1, a)\}$



$\delta(q_1, a, a) = \{(q_1, \in)\}$
$\delta(q_1, b, b) = \{(q_1, \in)\}$

$[\delta(q_1, \in, Z_0) = \{q_f, Z_0)\}$
[Accepts]




**OR**

**Q.5. (a) Design a PDA for the L = {$a^n b^n c^n$ | n ≥ 1} and if is not C F L then prove it using pumping lemma.** **(6.25)**

**Ans.** Assume L is context free. Then L satisfies P.L. Then there exists n by the P.L. Let z = a^n b^n c^n Because |z| >= n and z in L, by PL there exist uvwxy s.t.

z = uvwxy
|vwx| < = n
|vx| > = 1
for all i > = θ, u v^i w x^i y in L

But if there exist u,v,w,x,y satisfying the first three constraints, we can show that there exists an i s.t. the fourth constraint fails.

Case uvw consists only of "a". Then when i=0, the string is not in L (fewer "a"s than "b"s or "c"s).

Case vwx contains only "b"s - similar.

Case vwx contains only "c" s - similar.

Case vwx contains only two types of characters from {a,b,c}. Then uwy has more of the remaining type of characters.

The string vw cannot contain "a"s, "b"s, and "c"s, since vwx is a substring of z and has length < = n (cannot contain the last "a" and the first "c" both).i.e this is a contradiction and our assumption is wrong that L is CFL.

Since the language is not a CFL therefore we cannot construct a PDA for the language.

**Q. 5. (b) Prove that if A is a PDA accepting language L by empty store then we can design a PDA B which can accept the same language L by final state.** **(6.25)**

**Ans.** We need to show the conversion from a PDA $P_n$ that accepts a language L by empty stack to a PDA $P_f$ that accepts L by final state.

**Theorem:** If L = N($P_n$) for some PDA $P_n = (Q_n, \Sigma, \Gamma_n, \delta_n, q_0, Z_0, F_n)$, then there is a PDA $P_f = (Q_f, E, \Gamma_f, \delta_f, p_0, X_0, F_f)$ such that L = L($P_f$).

**Proof:** The idea behind the proof is in Figure. We use a new symbol $X_0$, which must not be a symbol of $\Gamma_n$; $X_0$ is both the start symbol of $P_f$ and a marker on the bottom of the stack that lets us know when $P_n$ has reached an empty stack. That is, if $P_f$ sees $X_0$ on top of the stack, then it knows that $P_n$ would empty its stack on the same input.

We also need a new start state, $p_0$, whose sole function is to push $Z_0$, the start state of $P_n$, onto the top of the stack and enter state $q_0$, the start state of $P_n$. Then, $P_f$ simulates $P_n$, until the stack of $P_n$ is empty, which $P_f$ detects because it sees $X_0$ on the top of the stack. Finally, we need another new state, $P_f$, which is the accepting state of $P_f$; this PDA transfers to state $p_f$ whenever it discover that $P_n$ would have emptied its stack.

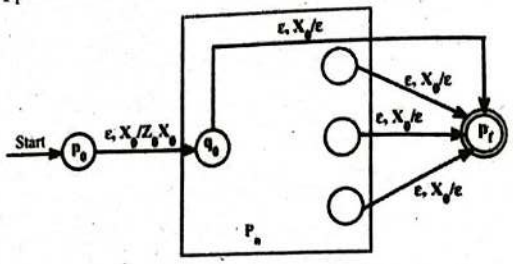



**Fig. $P_f$ simulates $P_n$ and accepts if $P_n$ empties its stack**

Tlie specification of $P_f$ is as follows:

$$Q_f = Q_n \cup \{p_0, p_f\}$$
$$\Gamma_f = \Gamma_n \cup \{X_0\}.$$
$$F_f = (p_f).$$

$\delta_f$ is defined by

1. $\delta_f(p_0, \in, X_0) = \{(q_0, Z_0X_0)\}$. In its start state, $P_f$ makes a spontaneous transition to the start state of $P_n$, pushing its start symbol $Z_0$ onto the stack.

2. For all state $q \in Q_n$, inputs $a \in \Sigma_0$ or $a = \in$ and stack symbol $Y \in \Gamma_n$, $\delta_f(q,a,Y)$ contains all the pairs in $\delta_n(q, a, Y)$.

3. In addition to rule (2). $\delta_f(q, \in, X_0)$ contains $(p_f, \in)$ for every state $q \in Q_n$.

We must show that $w$ is in $L(P_f)$ if and only if $w$ is in $N(P_n)$.

(If) We are given that $(q_0,w, Z_0) * \vdash p_n(q,\in, \in)$ for some state $q$. Insert $X_0$ at the bottom of the stack and conclude $(q_0, w, Z_0X_0) \vdash p_n(q, \in, X_0)$. Since by rule (2) above, $P_f$ has all the moves of $P_n$, we may also conclude that $(q_0,w, Z_0X_0)^* \vdash p_f(q, \in, X_0)$. If we put this sequence of moves with the initial and final moves from rules (1) and (3) above, we get:

$(p_0, w, X_0) \vdash p_f(q_0, w, Z_0X_0) \vdash^* p_f(q, \in, X_0) \vdash p_f(p_f, \in, \in)$... (A)

Thus, $P_f$ accepts $w$ by final state.

**Only-if** The converse requires only that we observe the additional transitions of rules (1) and (3) gives us very limited ways to accept $w$ by final state. We must use rule (3) at the last step, and we can only use that rule if the stack of $P_f$ contains only $X_0$. No $X_0$'s ever appear on the stack except at the bottommost position. Further, rule (1) is only used at the first step, and it must be used at the first step.

Thus, any computation of $P_f$ that accept $w$ must look like sequence (A). Moreover, the middle of the computation - all but the first and last steps - must also be a computation of $P_n$ with $X_0$ below the stack. The reason is that, except for the first and last steps, $P_f$ cannot use any transition that is not also a transition of $P_n$, and $X_0$ cannot be exposed or the computation would end at the next step. We conclude that $(q_0,w,Z_0) * \vdash p_n(q, \in, \in)$. That is $w \in N(P_n)$.

**Q. 6. (a) Design a Turing machine to concatenate two string of {0, 1} separated by a black tape symbol.** **(6.25)**



**Ans.** A number is represented in binary format in different finite automatas like 5 is represented as (101) but in case of addition using a turing machine unary format is followed. In unary format a number is represented by either all ones or all zeroes. For example, 5 will be represented by a sequence of five zeroes or five ones. 5 = 1 1 1 1 1 or 0 0 0 0 0. Lets use zeroes for representation.

**Approach used –** Convert a 0 in the first number in to X and then traverse entire input and convert the first blank encountered into 0. Then move towards left ignoring all 0's and "c". Come the position just next to X and then repeat the same procedure till the time we get a "c" instead of X on returning. Convert the c into blank and addition is completed.

**Steps –**

**Step-1:** Convert 0 into X and goto step 2. If symbol is "c" then convert it into blank(B), move right and goto step-6.

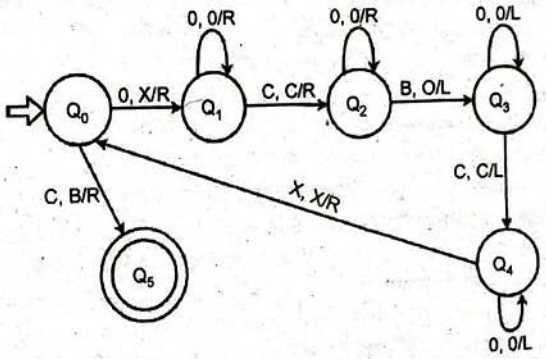
**Step-2:** Keep ignoring 0's and move towards right. Ignore "c", move right and goto step-3.

**Step-3:** Keep ignoring 0's and move towards right. Convert a blank(B) into 0, move left and goto step-4.

**Step-4:** Keep ignoring 0's and move towards left. Ignore "c", move left and goto step-3.

**Step-5:** Keep ignoring 0's and move towards left. Ignore an X, move left and goto step-1.

**Step-6:** End.




**Q. 6. (b) State and prove Turing Church's Thesis.** **(6.25)**

**Ans.** In computability theory, the **Church–Turing thesis** is a hypothesis about the nature of computable functions. It states that a function on the natural numbers can be calculated by an effective method if and only if it is computable by a Turing machine.Earlier several independent attempts were made to formalize the notion of computability:

- In 1933, Kurt Gödel, with Jacques Herbrand, created a formal definition of a class called general recursive functions. The class of general recursive functions is the smallest class of functions (possibly with more than one argument) which includes all constant functions, projections, the successor function, and which is closed under function composition, recursion, and minimization.

• In 1936, Alonzo Church created a method for defining functions called the λ-calculus. Within λ-calculus, he defined an encoding of the natural numbers called the Church numerals. A function on the natural numbers is called λ-computable if the corresponding function on the Church numerals can be represented by a term of the λ-calculus.

• Also in 1936, before learning of Church's work, Alan Turing created a theoretical model for machines, now called Turing machines, that could carry out calculations from inputs by manipulating symbols on a tape. Given a suitable encoding of the natural numbers as sequences of symbols, a function on the natural numbers is called Turing computable if some Turing machine computes the corresponding function on encoded natural numbers.

Church and Turing proved that these three formally defined classes of computable functions coincide: a function is λ-computable if and only if it is Turing computable if and only if it is *general recursive*. This has led mathematicians and computer scientists to believe that the concept of computability is accurately characterized by these three equivalent processes.

On the other hand, the Church–Turing thesis states that the above three formally-defined classes of computable functions coincide with the *informal* notion of an effectively calculable function. Since, as an informal notion, the concept of effective calculability does not have a formal definition, the thesis, although it has near-universal acceptance, cannot be formally proven.

**OR**

**Q. 7. (a) Design Turing machine for L = $\{a^n b^m c^k \mid k = n + m$ and $n, m, k \geq 1\}$** **(6.25)**

**Ans. Steps –**

**Step-0:** Convert "a" into "X", move right and go to state 1. If symbol is "b", ignore it, move right and go to state-4.

**Step-1:** If symbol is "a", ignore it and move right, remain on same state. If symbol is "b", ignore it, move right and go to state-2.

**Step-2:** If symbol is "Z", ignore it and move right, remain on same state. If symbol is "b", ignore it and move right, remain on same state and if symbol is "c", convert it into "Z", move left and go to state-3.

**Step-3:** If symbol is "Z", ignore it and move left, remain on same state. If symbol is "b", ignore it and move left, remain on same state. If symbol is "a", ignore it and move left, remain on same state, and if symbol is "X", ignore it and move right, go to state-0.

**Step-4:** If symbol is "b", ignore it and move left, go to state 5, and if symbol is "Z", ignore it and move left, go to state-5.

**Step-5:** Convert "b" into "Y", move right and go to state 6, and if symbol is "Z", ignore it and move right, go to state-8.

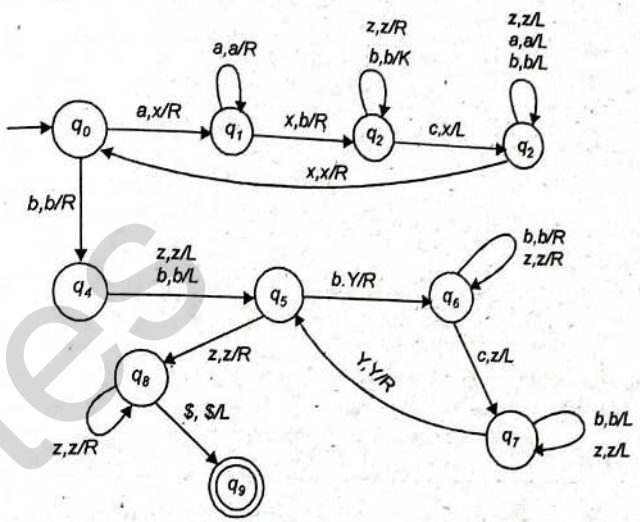
**Step-6:** If symbol is "Z", ignore it and move right, remain on same state. If symbol is "b", ignore it and move right, remain on same state, and if symbol is "c", convert it into "Z", move left and go to state-7.

**Step-7:** If symbol is "Z", ignore it and move left, remain on same state. If symbol is "b", ignore it and move left, remain on same state, and if symbol is "Y", ignore it and move right, go to state-5.



**Step-8:** If symbol is "Z", ignore it and move right, remain on same state, and if symbol is "$", ignore it and move left, go to state-9.

**Step-9:** String ACCEPTED




**Q.7. (b) What do you mean by recursive and recursively enumerable language. Explain with example.** **(6.25)**

**Ans.** Refer Q. 7. (b) of End Term Examination 2018. (Page No. 16-2018)

**Q.8. Attempt any two:**

**Q.8. (a) Construct the time complexity T(n) for the Turing machine M accepting string of type $0^n1^n$ where $n \geq 1$.** **(6.25)**

**Ans.** A step of a Turing machine is one event where the TM takes a transition. Running a TM on different inputs might take a different number of steps.The number of steps a TM takes on some input is sensitive to

• The structure of that input.

• The length of the input

The time complexity of a TM M is a function (typically denoted f(n)) that measures the worst-case number of steps M takes on any input of length n.

• By convention, n denotes the length of the input.

• If M loops on some input of length k, then f(k) = ∞

Now on constructing the turing machine for L={$0^n1^n$ | n>=1}

• Q = {q0,q1,q2,q3} where q0 is initial state.

• T = {0,1,X,Y,B} where B represents blank.

• Σ = {0,1}

• F = {q3}

**Transition function ä is given in Table 1 as:**

| | 0 | 1 | X | Y | B |
|---|---|---|---|---|---|
| q0 | (q1,X,R) | | | (q3,Y,R) | |
| q1 | (q1,0,R) | (q2,Y,L) | | (q1,Y,R) | |
| q2 | (q2,0,L) | | (q0,X,R) | (q2,Y,L) | |
| q3 | | | | (q3,Y,R) | Halt |

**Illustration**

Let us see how this turing machine works for 0011. Initially head points to 0 which is underlined and state is q0 as:

| B | 0 | 0 | 1 | 1 | B |
|---|---|---|---|---|---|

The move will be δ(q0, 0) = (q1, X, R). It means, it will go to state q1, replace 0 by X and head will move to right as:

| B | X | 0 | 1 | 1 | B |
|---|---|---|---|---|---|

The move will be δ(q1, 0) = (q1, 0, R) which means it will remain in same state and without changing any symbol, it will move to right as:

| B | X | 0 | 1 | 1 | B |
|---|---|---|---|---|---|

The move will be δ(q1, 1) = (q2, Y, L) which means it will move to q2 state and changing 1 to Y, it will move to left as:

| B | X | 0 | Y | 1 | B |
|---|---|---|---|---|---|

Working on it in the same way, the machine will reach state q3 and head will point to B as shown:

| B | X | X | Y | Y | B |
|---|---|---|---|---|---|

Using move δ(q3, B) = halt, it will stop and accepted.

The required time complexity of the turing machine is n+1.

**Q.8. (b) Explain different time based complexity classes.** **(6.25)**

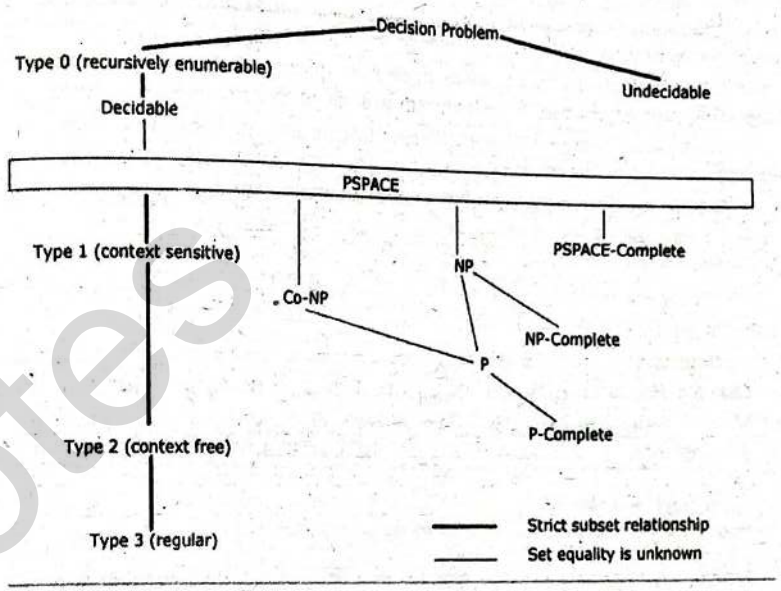
**Ans. Complexity Classes**

• A complexity class is the set of all of the computational problems which can be solved using a certain amount of a certain computational resource.

• The complexity class P is the set of decision problems that can be solved by a deterministic machine in polynomial time. This class corresponds to an intuitive idea of the problems which can be effectively solved in the worst cases.



• The complexity class NP is the set of decision problems that can be solved by a non-deterministic machine in polynomial time. This class contains many problem that people would like to be able to solve effectively. All the problems in this class have the property that their solutions can be checked effectively.




**Complexity Class P**

• P is the complexity class containing decision problems which can be solved by a deterministic Turing machine using a polynomial amount of computation time, or polynomial time.

• P is often taken to be the class of computational problmes which are "efficiently solvable" or "tractable".

• Problems that are solvable in theory, but cannot be solved in practice, are called intractable.

• There exist problems in P which are intractable in practical terms; for example, some require at least $n^{1000000}$ operations.

**Complexity Class NP**

• In computational complexity theory, NP ("Non-deterministic Polynomial time") is the set of decision problems solvable in polynomial time on a non-deterministic Turing machine.

• It is the set of problems that can be "verified" by a deterministic Turing machine in polynomial time.

• All the problems in this class have the property that their solutions can be checked effectively.

• This class contains many problems that people would like to be able to solve effectively, including

→ the Boolean satisfiability problem (SAT)

→ the Hamiltonian path problem (special case of TSP)

→ the Vertex cover probler

**Complexity Class NP-Complete**

• In complexity theory, the NP-complete problems are the most difficult problems in NP ("non-deterministic polynomial time") in the sense that they are the ones most likely not be in P.

• If one could find a way to solve any NP-complete problem quickly (in polynomial time), then they could use that algorithm to sovle all NP problems quickly.

• At present, all known algorithms for NP-complete problems require time that is superpolynomial in the input size.

• To solve an NP-complete problem for any nontrivial problem size, generally one of the following approaches is used.

→ Approximation

→ Probabilistic

→ Special cases

→ Heuristic

**Q.8. (c) Prove that if $P_1$ is N P-complete and there is a polynomial time reduction from $P_1$ to $P_2$, then $P_2$ is also N P-complete.**

**Ans.** Refer Q. 1. (e) of End Term Examination 2019.