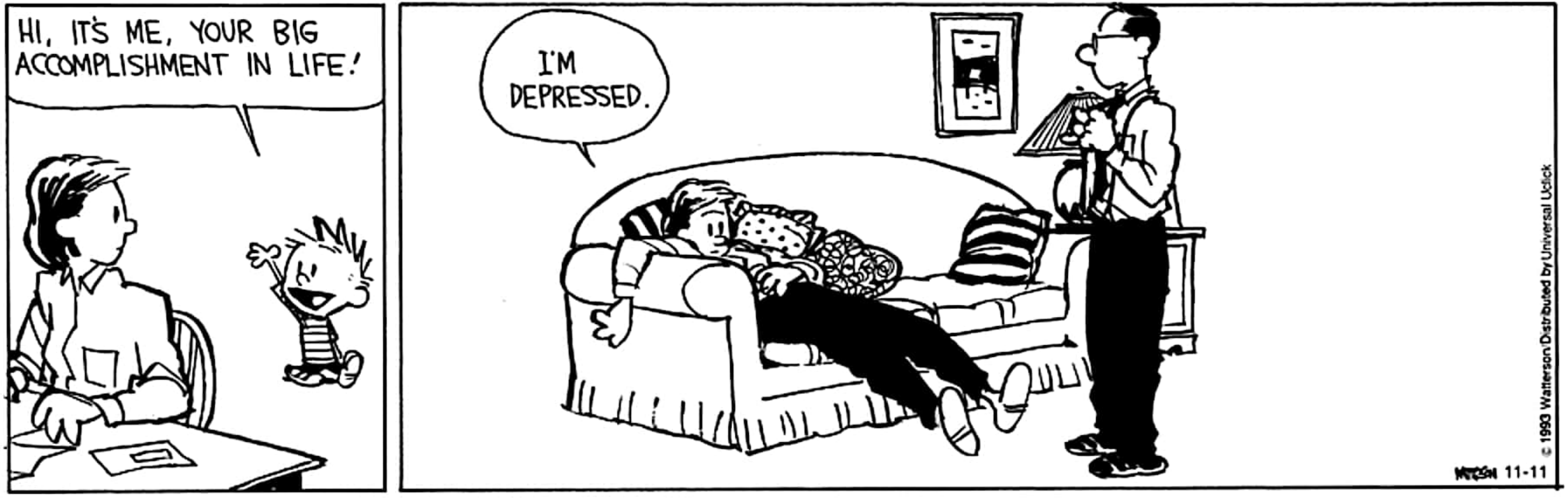


Syllabus

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]

MODEL TEST PAPER-1
FIRST TERM EXAMINATION
FOURTH SEMESTER (B. TECH)
THEORY OF COMPUTATION [ETCS-206]

Time : 1.30 hrs. **M.M. : 30**

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

Q.1.(a) What do you mean by Σ^+? **(5 × 2 = 10)**

Ans. The set of strings, exclude the empty string over an alphaset Σ^+ is denoted by Σ^+. For example, $\Sigma = \{0, 1\}$ then Σ^+ is a set containing all combinations of 0's and 1's. Thus

$$\Sigma^+ = \{0, 1, 01, 10, 00, 11, 000, 010, ...\}$$

similarly $\{0\}^+ = \{0, 00, 000, 0000 ....\}$

$$\{1\}^+ = \{1, 11, 111, 1111, ...\}$$

Q.1. (b) Give the formal statement of pumping Lemma for regular sets?

Ans. We give a necessary condition for an input string to belong to a regular set. The result is called pumming lemma as it gives a method of pumping (generating) many input strings from a given string. As pumping lemma gives a necessary condition, it can be used to show that certain sets are not regular.

Q.1. (c) What is the difference between DFA and NFA?

Ans.

DFA	NFA
1. DFA can be represented by a 5-tuple i.e. $(Q, \Sigma, \delta, q_0, F)\ \delta :- Q \times \Sigma \to Q$.	1. NFA can be represented by a s-type i.e. $(Q, \Sigma, \delta, q_0, F)\ \delta :- Q \times \Sigma \to 2^Q$.
2. Null moves is not allowed.	2. Null moves is allowed here.
3. Dead configuration does not allowed.	3. Dead configuration is allowed.
4. Time complexity is low.	4. Time complexity is High.
5. Space complexity is High.	5. Space complexity is low.

Q.1. (d) Give the regular expression for set of all strings in which every 0 is followed immediately by 11.

Ans. Let us analyze strings, as follows:

1, 111, 011, 011011, 0111, 1011 etc

So regular expression will be

$(1 + 011)^*$

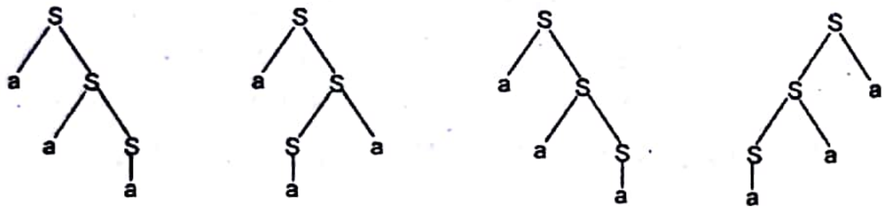
Q.1. (e) What do you mean by ambiguous grammer?

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.

Q.2. (a) Construct a Melay Machine equivalent to the Moore machine given by the following table: **(5)**

Ans.

Present State	Next State		Output
	$a = 0$	$a = 1$	
$\rightarrow q_0$	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

Ans. 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
$\rightarrow q_0$	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.2. (b) Construct a DFA for the following: **(5)**

(i) The set of all string that end with 00 over $\Sigma = \{0, 1\}$.

(ii) The set of all string that contain sub-string 011 over $\Sigma = \{0, 1\}$.

Ans. (i) Let $M = (Q, \Sigma, \delta, q_0, F)$ be the DFA.

q_0 : initial state.

Σ : $\{0, 1\}$ is given.

So, according to the problem we have to design a FA which accepts strings $w = w'00$ where w' can be any combination of 0's and 1's for example 011100, 111100, 00, 100, 01010100 etc.

Now first we will decide the approach to design FA. It is not an easy task to think FA as a whole. So, first we have to fulfill minimum condition i.e. every string end in 00.

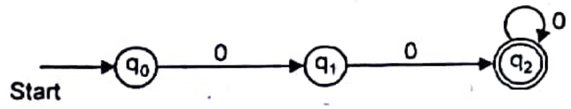


Fig. 1

So Fig. 1 satisfied the minimum condition and Fig-2 is the transition diagram of required FA.

$$Q = \{q_0, q_1, q_2\}$$
$$F = \{q_2\}$$

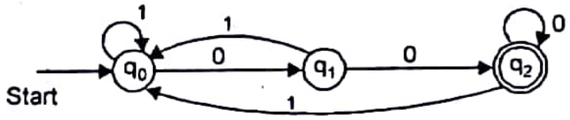
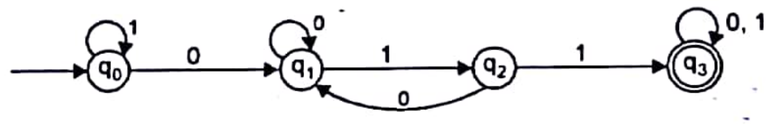


Fig. 2

Ans. (ii) So after analysing the problem we find that every string of the language is like

011, 0011, 010110, 0011110 ....



Let FA is $M = (Q, \Sigma, \delta, q_0, F)$

$$Q = \{q_0, q_1, q_2, q_3\}$$
$$F = \{q_3\}$$

Q.3. (a) Show that the set $L = \{a^{i^2} / i \geq 1\}$ is not regular?

Ans. Step 1: Suppose L is regular. Let n be the number of states in the finite automaton accepting L.

Step 2: Let $w = a^{n^2}$. Then $|w| = n^2 > n$. By pumping Lemma, we can write $w = xyz$ with $|xy| \leq n$ and $|y| > 0$.

Step 3. Consider xy^2z. $|xy^2z| = |x| + 2|y| + |z| > |x| + |y| + |z|$ as $|y| > 0$. This means $n^2 = |xyz| = |x| + |y| + |z| < |xy^2z|$. As $|xy| \leq n$, we have $|y| \leq n$. Therefore,

$$|xy^2z| = |x| + 2|y| + |z| \leq n^2 + n$$

i.e.,
$$n^2 < |xy^2z| \leq n^2 + n < n^2 + n + n + 1$$

Hence, $|xy^2z|$ strictly lies between n^2 and $(n + 1)^2$ but is not equal to any one of them. Thus $|xy^2z)$ is not a perfect square and so $xy^2z \notin L$. But by pumping Lemma, $xy^2z \in L$ This is a contradiction.

So, L is not Regular.

Q.3. (b) Construct the regular expression for the following:

(i) The set of all strings over [0, 1] with three consective 0's?

(ii) The set of all strings over [0, 1] ending with 00 and beginning with 1.

Ans. (*i*) $\Sigma = \{0, 1\}$ given the set of strings, according to the required regular expression 000, 00011, 1100001 ... So clearly we concern about only three consecutive zero's. So regular expression is

$$r = (0 + 1)^* 000 (0 + 1)^*$$

Ans. (*ii*) Let us first analyse the Language of regular expression. Problem is very simple and the language will have set of strings like 100, 1100, 101100 ... So regular expression is

$$r = 1 (0 + 1)^* 00.$$

Q.3. (*a*) Prove that the set of regular lnguages are closed under complementation? **(5)**

Ans. Let $M = (Q, \Sigma, \delta, q_0 F)$ be a DFA that accepts L_1. Then the DFA.

$$M' = (Q, \Sigma, \delta, q_0, Q - F)$$

accepts $\bar{L}_1$. This is rather straigforward; we have already suggested the result. Note that in the defination of a DFA we assumed δ^* to be a total function, so that $\delta^* (q_0, w)$ is defined for all $w \in \Sigma^*$. Consequently either $\delta^*(q_0, w)$ is a final state, in which case $w \varepsilon L$, or $\delta^*(q_0, w) \Sigma Q - F$ and $\omega \varepsilon \bar{L}$.

Q.4. (*b*) Find the regular expression accepted by following automata? **(5)**

Ans. We can apply the Arden's Method directly as the transition diagram does not contain more than one initial state and are no ε-moves. We get the following equations for 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 Theorem $\wedge R = R$ to the q_1-equations we get

$$q_1 = \wedge 0^* = 0^*$$

So,

$$q_2 = q_1 1 + q_2 1$$
$$q_2 = 0^* 1 + q_2 1$$
$$q_2 = (0^* 1) 1^*$$

As, the final states are q_1 and q_2, we need not solve for q_3:

$$q_1 + q_2 = 0^* + 0^* (11^*)$$
$$= 0^* (\wedge + 11^*) \qquad [\because \wedge + RR^* = R^*]$$
$$= 0^* 1^*$$

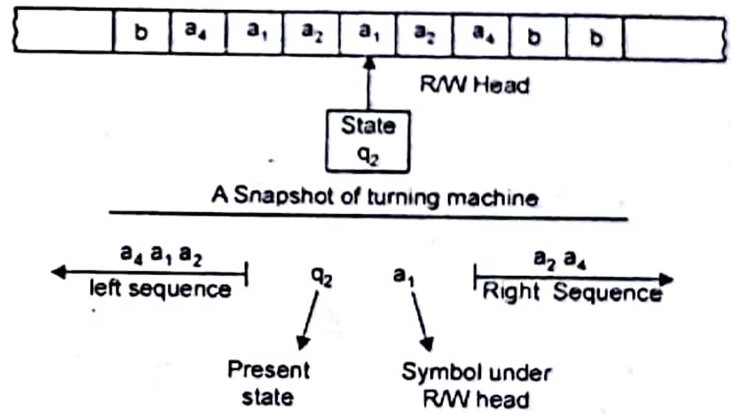
MODEL TEST PAPER-1
SECOND TERM EXAMINATION
FOURTH SEMESTER (B. TECH)
THEORY OF COMPUTATION [ETCS-206]

Time : 1.30 hrs. **M.M. : 30**

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

Q.1.(*i*) What do you mean by ID in Turning M/C with example? **(2)**

Ans. An ID (Instantaneous Descriptions) of a Turning machine M is a string QBv, where β is the present state of M, the entire input string is split as αv, the first symbol of v is the current symbol a under the R/w head and v has all the subsequent symbols of the input string, and the string α is the substring of the input string formed by all the symbols to the left of a.



A Snapshot of turning machine

Q.1. (*ii*) What is the difference between a recursive language and recursively enumerable language? **(2)**

Ans. Recursively Enumerable Language: A language L is said to be recursively enumerable if there exists a Turning machine that accepts it. This defination implies only that there exists a Turning machine M, such that, for every $w \varepsilon L$,

$$q_0 w \xleftarrow{*}_{M} x_1 q_f x_2$$

with q_f a final state. The defination says nothing about what happens for w not in L; it may be that the machine halts in a non-final state or that it never halts and goes into an infinite Loop.

Recursive Language: A language L an Σ is said to be recursive if there exists a Turning machine M that accepts L and that halts on every w in Σ^+. In other words, a language is recursive if and only if there exists a membership algorithm for it.

Q.1. (*iii*) What is the purpose of studying Turning machines? **(2)**

Ans. Purpose of Turning Machines:

• Turning machine is used for determining the undecidability of certain Languages and measuring the space and time complexity of problems.

• As compared last automaton Like Finite Automation (No storage) and Push down automaton (Storage is stack). Extrapolating from this observation, we can expect to discover even more powerful language families if we give the automation more flexible storage. This leads to the fundamental concept of a Turning machine.

Q.1. (*iv*) Write a CFG, which generates palindrome for binary numbers? (2)

Ans. Grammar will generate polindrome for binary numbers, that is 00, 010, 11, 101, 11100111....

Let CFG be $G = (V, \Sigma, P, S)$

where V = set of non-terminal = $\{S\}$

Σ = set of terminals = $\{0, 1\}$

and production rule P is defined as

$$S \rightarrow 0S0/1S1$$

$$S \rightarrow 0/1/\varepsilon$$

Q.1. (*v*) What are defects arises in CFG? (2)

Ans. Defects arises in CFG:

• Grammar holds useless symbols like non-generating symbol and Non-rechable symbol.

• Grammar has ε-Productions.

• Grammar has unit production.

Q.2. (*a*) Consider the following productions. (10)

$$S \rightarrow 0B/1A$$

$$A \rightarrow 0/0S/1AA$$

$$B \rightarrow 1/1S/0BB.$$

For the string 00110101. Find

(*a*) the Leftmost derivation.

(*b*) the rightmost derivation.

(*c*) the derivation tree.

Ans. (*a*) Leftmost Derivation:

$S \Rightarrow 0\underline{B}$

$\Rightarrow 00BB$

$\Rightarrow 001\underline{B}$

$\Rightarrow 0011\ \underline{S}$

$\Rightarrow 00110\underline{B}$

$\Rightarrow 001101\ \underline{S}$

$\Rightarrow 00\ 11\ 010\ \underline{B}$

$\Rightarrow 00\ 11\ 0101$

(*b*) Rightmost Derivation:

$S \Rightarrow 0B$

$\Rightarrow 00BB$

$\Rightarrow 00B1\underline{S}$

$\Rightarrow 00\ B\ 1\ 0\ \underline{B}$

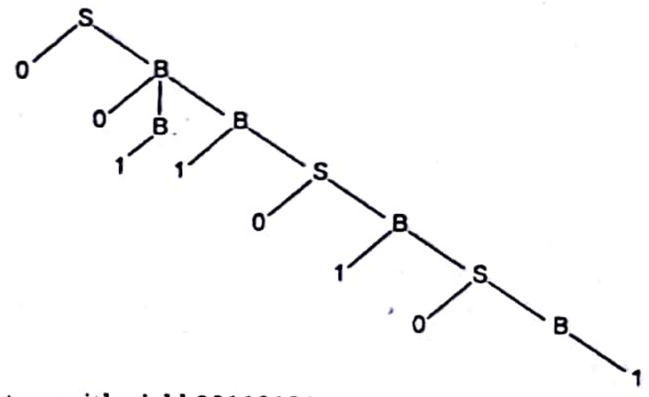
$\Rightarrow 00\ B\ 101\underline{S}$

$\Rightarrow 00\ B\ 101\ 0\underline{B}$

$\Rightarrow 00\ B\ 101\ 01$

$\Rightarrow 00\ 11\ 01\ 01$

Q.2. (*c*) The derivation tree is:



The derivation tree with yield 00110101.

Q.3. (*a*) What do you mean by PDA? Explain ID for PDA? (5)

Ans. Defination of PDA: It consists of 7-types $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

(*i*) A finite non-empty set of states denoted by Q.

(*ii*) A finite non-empty set of input symbols denoted by Σ.

(*iii*) A finite non-empty set of pushdown symbols denoted by Γ.

(*iv*) A special state called initial state by q_0.

(*v*) A special pushdown symbol called the initial symbol on the pushdown store denoted by Z_0.

(*vi*) A set of final states, a subset of Q denoted by F, and

(*vii*) A transition Function δ from $Q \times (\Sigma \cup \{\wedge\}) \times \Gamma$ to the set of finite subsets of $Q \times \Gamma^*$.

ID for PDA:

Let A $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. An instantaneous description (ID) is (q, x, α), where $q \in Q$, $x \in \Sigma^*$ and $\alpha \in \Gamma^*$.

For example, $(q, a_1, a_2 \ldots a_n, z_1 z_2 \ldots z_m)$ is an ID. This describes the PDA when the current state is q, the input string to be processed is $a_1, a_2 \ldots a_n$. The PDA will process $a_1 a_2 \ldots a_n$ in that order. The PDS has $z_1, z_2, \ldots, z_m$ with z_1 at the top z_2 is the second element from the top etc and z_m is the lowest element in PDS.

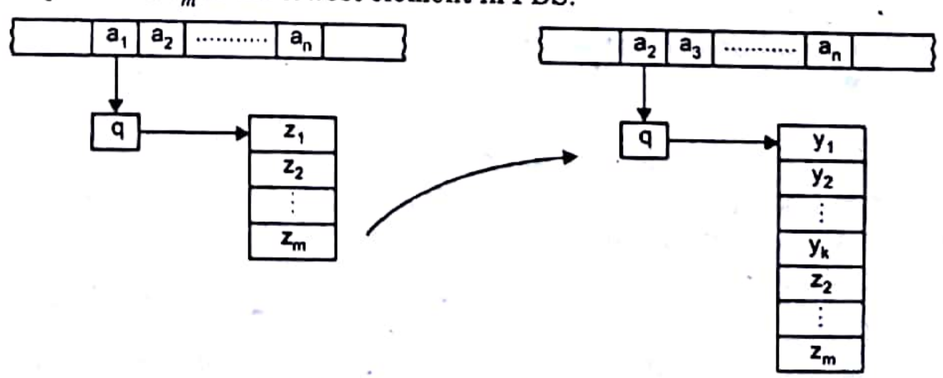


Fig. An Illustration of the Move Relation

Q.3. (b) Construct a PDA for accepting the language

$$L = \{a^n b^n / n \geq 1\}$$ **(5)**

Ans. $A = \{\{q_0, q_1\}, \{a, b\}, \{a, z_0\}, \delta, q_0, Z_0, \phi\}$

where δ is given by

$$R1 : \delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$$
$$R2 : \delta(q_0, a, a) = \{(q_0, aa)\}$$
$$R3 : \delta(q_0, b, a) = \{(q_1, \wedge)\}$$
$$R4 : \delta(q_1, b, a) = \{(q_1, \wedge)\}$$
$$R5 : \delta(q_1, \wedge, Z_0) = \{q_1, \wedge)\}$$

$R1$ is used to store a in PDA if it is first symbol of an input string. $R2$ can be used repeatedly to store a^n in PDS. When b is encountered for the first time in the input string, a is erased (in PDS) using $R3$. Also, the PDA makes a transition to state q_1. After processing the entire input string, if z_0 remains in PDS, it can be erased using the null move given by RS So, if $w = a^n b^n$, then we have

Acceptance in PDA for $a^n b^n$:

$$(q_0, a^n b^n, z_0) \xrightarrow{*} (q_0, b^n, a^n z_0) \text{ by } R1 \text{ and } R2$$
$$\xrightarrow{*} (q_1, \wedge, Z_0) \text{ by } R3 \text{ and } R4$$
$$\rightarrow (q_1, \wedge, \wedge) \text{ by R5 only.}$$

Therefore, $a^n b^n \varepsilon N(A)$.

If $w \varepsilon N(A)$, then $(q_0, \omega, Z_0) \xrightarrow{*} (q_1, \wedge, \wedge)$ Note that the PDS can be empty only when A is in state q_1.

Q.4. Explain the Partial Recursive Function with example?

Ans. Partial Recursive Function: Partial recursive function are turning computable. It means that there exist a turning machine TM for every partial recursive Function.

A function is called partial recursive if it is defined for some of its arguments. Let $f(a1, a2 \ldots, a_n)$ is a function and defined on function $g(b1, b2 \ldots b_m)$, then f is a partial function if some elements of f is assigned to almost one element of function g.

For example: If R denotes the set of all real numbers, the rule of from R to itself given by $f(r) = +\sqrt{r}$ is a partial function since $f(r)$ is not defined as a real number when r is negative.

Q.4. (b) Explain Primitive Recursive Function with example? **(5)**

Ans. Primitive Recursive Function: A total function of over N is called primitive recursive.

(*i*) If it is any one of the three initial Functions.

or (*ii*) If it can be oftained by applying composition and recursion a finite number of times to set of initial function.

For Example:

Show that the function $f(x, y) = x + y$ is primitive recursive.

Ans. f_1 is a function of two variables. If we want f_1 to be defined by recursion, we need a function g of a single variable and a function h of three variables.

$$f_1(x, 0) = x + 0 = x$$

By comparing $f_1(x, 0)$ with L.H.S., we see that g can be defined by

$$g(x) = x = U_1^1(x)$$

Also, $$f_1(x, y + 1) = x + (y + 1) = (x + y) + 1 = f_1(x, y) + 1$$

By comparing $f_1(x, y + 1)$ with L.H.S.,

we have $$h(x, y, f, (x, y)) = f_1(x, y) + 1$$
$$= S(f_1(x, y))$$
$$= S(U_3^3(x, y, f_1(x, y)))$$

MODEL TEST PAPER-1
END TERM EXAMINATION
FOURTH SEMESTER (B. TECH)
THEORY OF COMPUTATION [ETCS-206]

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

Note: *Attempt any one question from each unit. Q.No. 1. is compulsory.*

Q.1.(a) What do you mean by Alphabets? **(2.5)**

Ans. Alphabets are defined as a finite set of symbols.

Everything in mathematics is based on symbols. This is also true for automata theory of computation. The symbols are generally letters and digits.

Example: $\Sigma = \{0, 1\}$

$\Sigma = \{A, B, \ldots Z\}$

$\Sigma = \{a, b, \ldots z\}$

is also alphabet.

Q.1. (b) What is the meaning of concatenation of strings?

Ans. Let w_1 and w_2 be two strings. Then $w_1 w_2$ denotes the concatenation of w_1 and w_2, that is, the string formed by making a copy of w_1 and followed it by a copy of w_2.

For example:

If $w_1 = abc$

$w_2 = xyz$

$w_1 w_2 = abc\ xyz$

That is if $w = aba$

then $w^3 = (aba)^3$

$w^3 = aba\ aba\ aba$

Q.1. (c) What do you mean by LEX? **(2.5)**

Ans. Lex is a program (generator) that generates Lexical analyzers, (widely used on UNIX). It is mostly used with yacc Parser generator. It is written by Eric Schmidt and Mike Lesk. It reads the input stream (specifying the Lexical analyzer) and outputs source code implementing the lexical analyzer in the C programming Language. Lex will read patterns (regular expressions); then produces C code for a lexical analyzer.

For example: Lex is good at pattern matching.

Q.1. (d) For the Grammar $G\{S \to Aa\ S/a, A \to SS/\ ba/SbA\}$ find the Leftmost derivation for the string *aabaaa*. **(2.5)**

Ans. $S \to AaS$

$\to aa\ S$

$\to aa\ Aa\ S$

$\to aa\ baa\ S$

$\to aa\ baaa$

Q.1. (e) Give formal defination of PDA? **(2.5)**

Ans. A pushdown automata defined as:

(*i*) A finite nonempty set of states denoted by Q.

(*ii*) A finite non-empty set of Input symbols denoted by Σ.

(*iii*) A finite non-empty set of pushdown symbols by Γ.

(*iv*) A special state called the initial state denoted by q_0.

(*v*) A special pushdown symbol called the initial symbol on pushdown store denoted by z_0.

(*vi*) A set of final states, a subset of Q denoted by F.

(*vii*) A transition function δ from $Q \times (\Sigma \cup (\wedge)) \times \Gamma$ to the set of finite subsets of $Q \times \Gamma^*$.

Symbolicly, *a PDA* is a 7-tuple, namely $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$.

Q.1. (f) Give an example for a Regular Set? **(2.5)**

Ans. Any set represented by a regular expression is called a regular set.

For example:

- a denotes the set {a}
- $a + b$ denotes $\{a, b\}$.
- ab denotes $\{ab\}$
- a^* denotes the set $\{\wedge, a, aa, aaa \ldots.\}$
- $(a + b)^*$ denotes $\{\wedge, a, b, ab, ba, \ldots.\}$

Q.1. (g) List the four components used to form a context free Grammer? **(2.5)**

Ans. A grammar $G = (V, T, S, P)$ is said to be context-free if all productions in P have the form

$$A \to x$$

where $A \in V$ and $x \in (VUT)^*$

In this four components is

V:- Set of Non-terminals or Variables.

T: Set of terminals or inputs.

P: Set of Productions or rules.

S: Starting Symbol or Initial Symbol.

Q.1. (h) What is undecidable problem? Write on example? **(2.5)**

Ans. A turing machine that always answers "true" gives the correct answer, while in second case one that always answers "false" is appropriate. This may seem Like a factious answer, but it emphasizes an important point. The fact that we do not know what the correct answer is makes no difference, what matters is that there exists some Turing machine that does give the correct response.

Q.1. (i) What is the time complexity for the language $L = \{a^n\ b^n / n \geq 0\}$? (2.5)

Ans. Every Regular language can be recognised by a deterministic finite automation in time proportional to the length of the input.

Therefore, $L_{REG} \leq DTIME(n)$

But DTIME (n) includes much more than L_{REG}.

So, that context free language $\{a^n\ b^n / n \geq 0\}$ can be recognised in time $0(n)$.

Q.1. (f) What do you mean by polynomial-time reducible?

Ans. A language L_1 is said to be polynomial-time reducible to some language L_2 if there exists a deterministic turning machine by which any w_1 in the alphabet of L_1 can be transformed in polynomial time to a w_2 in the alphabet of L_2 in such a way that $w_1 \in L_1$ if and only if $w_2 \in L_2$

UNIT-1

Q.2. (a) Design FA for the language

$$L = \{(01)^i (1)^j / i \geq 1, j \geq 1\} \quad \textbf{(5)}$$

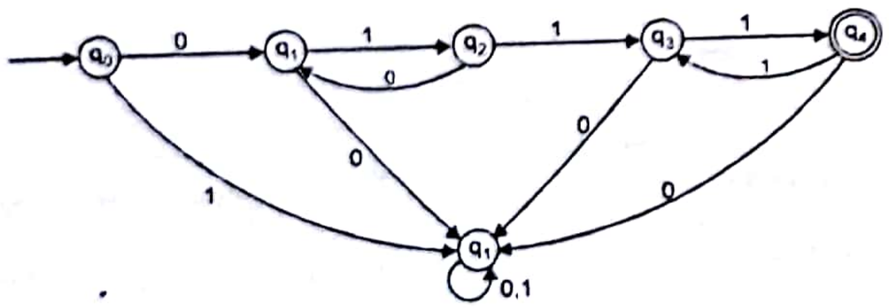
Ans. By analysing Language L, it is clear that FA will accepts strings start with any number of 01 (not empty) and end with even number of 1's.

Let FA be $M = (Q, \Sigma, \delta, q_0, F)$

q : Initial State

δ : Transition Function

Σ : {0, 1} given



$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$$
$$F = \{q_4\}$$

q_5 is dead state.

Q.2. (b) Construct a minimum state automaton equivalent to a DFA whose transition table is defined as:

State	a	b
q_0	q_1	q_2
q_1	q_4	q_3
q_2	q_4	q_3
(q_3)	q_5	q_6
(q_4)	q_7	q_6
q_5	q_3	q_6
q_6	q_6	q_6
q_7	q_4	q_6

Ans. $Q_1^0 = \{q_3, q_4\}$, $Q^0{}_2 = \{q_0, q_1, q_2, q_5, q_6, q_7\}$

$$\pi_0 = \{\{q_3, q_4\}, \{q_0, q_1, q_2, q_5, q_6, q_7\}\}$$

q_3 is 1-equivalent to q_4. So $\{q_3, q_4\} \in \pi_1$.

q_0 is not 1-equivalent to q_1, q_2, q_5 but q_0 is 1-equivalent to q_6.

Hence $\{q_0, q_6\} \in \pi_1$. q_1 is 1-equivalent to q_2 but not 1-equivalent to q_5, q_6 or q_7. So $\{q_1, q_2\} \in \pi_1$

q_5 is not 1-equivalent to q_6 but to q_7.

So, $\{q_5, q_7\} \in \pi_1$

Hence, $\pi_1 = \{\{q_3, q_4\}, \{q_0, q_6\}, \{q_1, q_2\}, \{q_5, q_7\}\}$

q_3 is 2-equivalent to q_4. So $\{q_3, q_4\} \in \pi_2$

q_0 is not 2-equivalent to q_6. So $\{q_0\}, \{q_6\} \in \pi_2$

q_1 is 2-equivalent to q_2. So $\{q_1, q_2\} \in \pi_2$

q_5 is 2-equivalent to q_7. So $\{q_5, q_7\} \in \pi_2$

Hence,

$\pi_2 = \{\{q_3, q_4\}, \{q_0\}, \{q_6\}, \{q_1, q_2\}, \{q_5, q_7\}\}$ q_3 is 3-equivalent to q_4. q_1 is 3-equivalent to q_2 and q_5 is 3-equivalent to q_7.

Hence,

$$\pi_3 = \{\{q_3\}, \{q_1, q_2\}, \{q_3, q_4\}, \{q_5, q_7\}, \{q_6\}\}$$

As $\pi_3 = \pi_2$, the minimum state automation is

$$M' = (Q', (a, b), \delta', [q_0], \{[q_3, q_4]\})$$

where δ' is defined by

Transition Table of DFA

State	a	b
$[q_0]$	$[q_1, q_2]$	$[q_1, q_2]$
$[q_1, q_2]$	$[q_3, q_4]$	$[q_3, q_4]$
$[q_3, q_4]$	$[q_5, q_7]$	$[q_6]$
$[q_5, q_7]$	$[q_3, q_4]$	$[q_0]$
$[q_6]$	$[q_6]$	$[q_6]$

Q.3. For every NDFA, there exists a DFA which simulates the behaviour of NDFA, Alternatively, if L is the set accepted by NDFA, then there exists a DFA which also accepts L? **[12.5]**

Ans. Let $M = (Q, \varepsilon, \delta, q_0, F)$ be an NDFA accepting. L. We construct a DFA M' as.

$$M' = \{Q', \Sigma, \delta', q'_0, F'\}$$

where

(*i*) $Q' = 2^Q$ (any state in Q' is denoted by $[q_1, q_2 \ldots q_j]$ where $q_1, q_2, \ldots q_j \in Q$.

(*ii*) $q'_0 = [q_0]$ and

(*iii*) F' is the set of all subsets of Q containing an element of F.

Before defining δ', Let us Look at the construction of Q', q_0' and F'. M is initially at q_0. But on application of an input symbol, say a, M can reach any of the states $\delta(q_0, a)$. To describe M, just after the application of the input symbol a, we require all the passible states at any instant of time. Hence the states of M' are defined as subsets of Q. As M starts with the initial state q_0, q_0' is defined as $[q_0]$. Now we can define δ'.

(*iv*) $\delta' ([q_1, q_2 \ldots q_i], a) = \delta(q_1, a) \cup \delta(q_2, a) \cup \ldots \cup \delta(q_i, a)$

Equivalently,

$$\delta'(q_1, q_2 \ldots q_i], a) = [p_i \ldots p_i]$$

if and only if

$$\delta(\{q_1, q_2 \dots q_i\}\ a) = \{p_1, p_2 \dots p_j\}$$

Before Proving $L = T(M')$, we prove an auxiliary result

$$\delta'(q_0', x) = [q_1, \dots q_i]$$

if and only if $\delta(q_0, x) = \{q_1 \dots q_i\}$ $\forall x$ in Σ^* we prove by induction on $|x|$, the if part, i.e.

$$\delta'(q_0', x) = [q_1, q_2 \dots q_i]$$

if $\delta(q_0, x) = [q_1, \dots q_i]$

When $|x| = 0$, $\delta(q_0, \wedge) = \{q_0\}$ and by defination of δ', $\delta'(q_0' \wedge) = q'_0 = [q_0]$

So i.e. true for $|x| = 0$. Thus there is basis for induction Assume that is true for all strings y with $|y| \le m$. Let x be a string of Length m + 1. We can write x as ya, where $|y| = m$ and $a \in \Sigma$. Let $\delta(q_0, y) = \{p_1 \dots p_j\}$ and $\delta(q_0, ya) = \{r_1, r_2 \dots r_k\}$. As $|y| \le m$ by induction hypothesis, we have

$$\delta'(q'_0, y) = [p_1 \dots p_j]$$

Also, $\{r_1, r_2 \dots r_k\} = \delta(q_0, ya) = \delta(\delta(q_0, y), a) = \delta(\{p_1 \dots p_j\}, a)$

By defination of δ',

$$\delta'([p_1 \dots p_j], a) = [r_1 \dots r_k]$$

Hence, $\delta'(q'_0, ya) = \delta'(\delta'(q'_0, y), a) = \delta'([p_1 \dots p_j], a)$

$$= [r_1 \dots r_k]$$

Thus we have proved for x = ya.

Hence $x \in T(M)$ if and only if $x \in T(M')$. This proves that DFA M' accepts L.

UNIT-II

Q.4. (a) Consider the following grammar. **(6)**

$$S \to aA$$
$$A \to b/\varepsilon$$

remove the Nullable non-terminal.

Ans. Given CFG is $S \to aA$

$$A \to b/\varepsilon$$

Here $A \to \varepsilon$ is ε–production, so let us apply above described concept, put ε in place of A, at the right side of productions and add the resulted production to the grammar.

Here is only one productions $S \to aA$, whose right side contains A. So, repelacing A by ε in this production, we get $S \to a$; now add this new productions to keep the language generated by this grammar same. Therefore, the ε-free gramm, equivalent to above grammar is

$$S \to aA$$
$$S \to a$$
$$A \to b$$

or

$$\boxed{\begin{array}{l} S \to aA/a \\ A \to b \end{array}}$$

there is no ε-production in above CFG.

Q.4. (b) If L_1 and L_2 are regular Languages, then so are $L_1 \cap L_2$. **[6.5]**

Ans. The proof of closure under intersection is a good example of a construction proof. Not only does it establish the desired result, but it also shows explicity how to construct a finite acceptor for the intersection of two regular languages. Constructive proofs occur throughout this book; they are important because they give us insight into the results and often serve as the starting point for practical algorithms. Here, as in many cases, there are shorter but non-constructive arguments. For closure under intersection, we start with Demorgan's Law equation, taking the complement of both sides. Then

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

for any language L_1 and L_2. Now if L_1 and L_2 are regular, then by closure under complementation, so are $\overline{L}_1$ and $\overline{L}_2$.

Q.5. Find a GNF grammar equivalent to the following CFG: **(12.5)**

$$S \to AA/a$$
$$A \to SS/b$$

Ans. The given grammar is in CNF. S and A are renamed as A_1 and A_2, respectively. So the productions are $A_1 \to A_2A_2/a$ and $A_2 \to A_1A_1/b$ As the given grammar has no null productions and is in CNF we need not carry out step 1 So we proceed to step 2.

Step 2. (*i*) A_1 –productions are in the required form. They are $A_1 \to A_2A_2/a$.

(*ii*) $A_2 \to b$ is in the required form. Apply Lemma 1 to $A_2 \to A_1A_1$.

The resulting productions are $A_2 \to A_2\,A_2\,A_1$,

$A_2 \to aA_1$. Thus the A_2 – productions are

$$A_2 \to A_2A_2A_1$$
$$A_2 \to aA_1$$
$$A_2 \to b$$

Step 3: We have to apply Lemma 2 to A_2-productions as we have $A_2 \to A_2A_2A_1$. Let Z be the new variable. The resulting productions are:

$$A_2 \to aA_1, A_2 \to b$$
$$A_2 \to a\,A_1Z, A_2 \to bZ$$
$$Z \to A_2A, \quad Z \to A_2\,A_1Z$$

Step 4: (*i*) The A_2-productions are

$$A_2 \to aA_1/b/aA_1\,Z/bZ$$

(*ii*) Among the A_1-productions we retain $A_1 \to a$ and eliminate $A_1 \to A_2A_2$ using Lemma 1. The resulting productions are $A_1 \to aA_1A_2/bA_2$. $A_1 \to aA_1\,ZA_2/bZA_2$. The set of modified A_1-productions is

$$A_1 \to a/aA_1\,A_2/bA_2/aA_1\,ZA_2/bZA_2$$

Step 5: The Z-productions to be modified are

$$Z \to A_2A_1, Z \to A_2\,A_1\,Z.$$

We apply Lemma 1 and we get

$$Z \to aA_1A_1/bA_1/aA_1\,ZA_1/bZA_1$$
$$Z \to aA_1A_1Z/bA_1Z/aA_1ZA_1\,Z/bZA_1Z$$

Hence the equivalent grammar is

$$G' = (\{A_1, A_2, Z\}, \{a, b\}\ P_1, A_1)$$

Where P_1 consists of

$$A_1 \rightarrow a/aA_1\,A_2/bA_2/aA_1\,ZA_1/bZA_2$$
$$A_2 \rightarrow aA_1/b/aA_1\,Z/bZ$$
$$Z \rightarrow aA_1\,A_1/bA_1/aA_1\,ZA_1/bZA_1$$
$$Z \rightarrow aA_1\,A_1\,Z/bA_1Z/aA_1ZA_1Z/bZA_1Z$$

UNIT-3

Q.6. (a) Explain Turing machines as transducers?

Ans. A function of with domain D is said to be Turing-computable or just computable if there exists some Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ such that

$$q_0\, w \xleftarrow[M]{*} q_f\, f(w),\ q_f \varepsilon F, \forall\, w \varepsilon D.$$

We have had little reason so far to study transducers; in language theory, acceptors are quite adequate. But as we will shortly see, Turing machines are not only intersting as Language acceptors, they provide us with a simple abstract model for digital computers in general. Since the primary purpose of a computer is to transform input output, it acts as a transducer. If we want to model computers using Turing machines, we have to look at this aspect more closely.

The input for for a computation will be all the nonblank symbols on the tape at the computation. the output will be whatever is then on the tape. Thus, we can view a Turing machine transducer M as implementation of a function of defined by

$$\hat{w} = f(w),$$

provided that

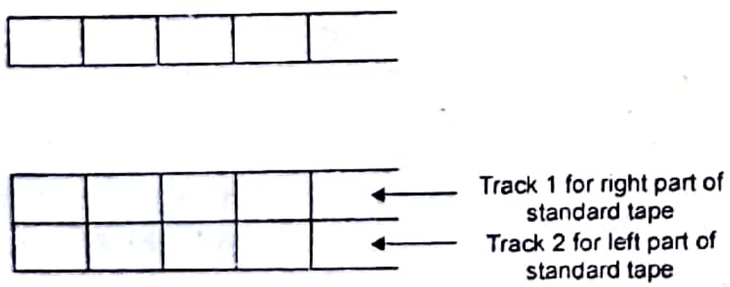
$$q_0 w \xleftarrow[M]{*} q_f \hat{w}$$

for some final state q_f.

Q.6. (b) Explain Turing machines with semi-infinite tape?

Ans. Many authors do not consider the model in figure 1 as standard, but use one with a tape that is unbounded only in one direction. We can visualize this as a tape that has a Left boundary. This Turing machine is otherwise identical to our standard model, except that no left move is permitted when the read-write head is at the boundary.

It is not difficult to see that this restriction does not affect the power of the machine. To simulate a standard Turing machine M by a machine $\hat{M}$ with a semi-infinite tape, we use the arrangement shown in Figure 2.

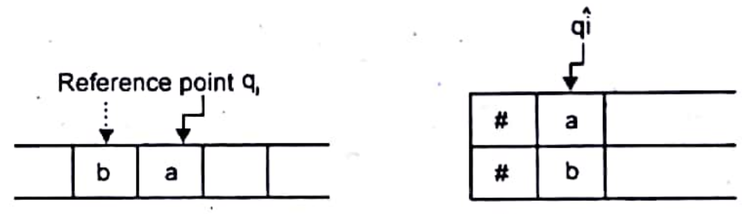


The simulating machine $\hat{M}$ has a tape with two tracks. On the upper one, we keep the information to the right of some reference point on M's tape. The reference point could be, for example, the position of the read write head at the start of the computation. The lower track contains the left part of M's tape in reverse order. $\hat{M}$ is programmed so that it will use information on the upper track only as long as M's read write head is to the right of the reference point, and work on the lower track as M moves into the left part of its tape. The distinction can be made by partitioning the state set of $\hat{M}$ into two parts, say Q_U and Q_L, the first to be used when working on the upper track, the second to be used on the lower are. Special end markers # are put on the left boundary of the tape to facilitate suitching from one track to other.

The simulating machine will first move via the transition.

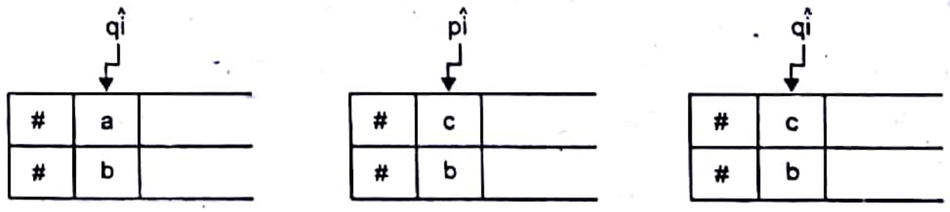
$$\hat{\delta}(\hat{q}_i, (a, b)) = (\hat{q}_j, (c, b), L)$$

Where $\hat{q}_i \in Q_U$. Because $\hat{q}_i$ belongs to Q_U, only information in the upper track is considered at this point. Now, the simulating machine sees (#, #)



in state $\hat{q}_j \varepsilon Q_U$. It next uses a transition

$$\hat{\delta}(\hat{q}_j, (\#, \#)) = (\hat{p}_j, (\#, \#), R)$$



Sequence of configuration in simulating
$s(q_i, a) = (q_j, c, L)$

Q.7. Explain a languages that are not recursively enumerable? (12.5)

Ans. We can establish the existence of languages that are not recursively enumerable in a variety of ways. One is very short and uses a very fundamental and elegant result of mathematics.

Theorem: Let S be an infinite countable set. Then its powerset 2^S not countable.

Proof: Let $S = (s_1, s_2, s_3 ...)$. Then any element t of 2^s can be represented by a sequence of 0's and 1's, with a 1 in position i if and only if si is in t. For example, the set (s_2, s_3, s_6)

is represented by 01100100..... while $\{s_1, s_2, s_3, ...\}$ is represented by 10101.... clearly, any element of 2^S can be represented by such a sequence, and any such sequence represents a unique element of 2^S. Suppose that 2^S were countable, then its element could be written in some order, say t_1, t_2,..... and we could enter these into a table, as shown in Figure. In this table, take the elements in the main diagonal, and complement each entry, that is, replace 0 with 1 and vice versa. In the example in Figure, the elements are 1100...., so we get 0011... as the result. The new sequence represents some element of 2^s say t_i for some i. But it cannot be t_1 because it differs from t_1 through s_1. For the same reason it cannot be t_2, t_3 or any other t_i. This contradiction creates a Logical impasse that can be removed only by throwing out the assumption that 2^s is countable.

t_1	(1)	0	0	0	0	.....
t_2	1	(1)	0	0	0	.....
t_3	1	1	(0)	1	0	.....
t_4	1	1	0	(0)	1	.....

Fig.

UNIT-IV

Q.8. (a) Explain Cook's Theorem? **(12.5)**

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 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

- α_0 is the initial ID of M with input w.
- $\alpha_0 \vdash \alpha_1 \vdash ... \vdash \alpha_k$, $k \le P(n)$
- α_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 α_i as a sequence of $p(n) + 1$ symbols (one symbol for the state and the remaining symbols for the tape symbols. So $a_i = x_{i_0}, x_{i_1} .... x_{ip(n)}$. If α_m is an accepting ID in the course of processing w then we write $\alpha_0 \vdash ... \vdash \alpha a \vdash \alpha_m = \alpha_{i, p(n)}$.

3. Representation of IDs in terms of Boolean Variables: We define a boolean variable y_{ijA} corresponding to (i, j)th entry in the ith 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 $w = a_1 a_2 ... a_n$ of length n and ending with b's. The corresponding boolean expression S is defined as:

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

(*ii*) Simulation of accepting ID: $\alpha_{p(n)}$ is the accepting ID. If $p_1, p_2, ... p_k$ 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

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

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

(*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.

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

Formulation of B_{ij}: When the state of α_i is none of $x_{i,j-1}, x_{ij}, x_{i,j+1}$, then the transition corresponding to α_{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

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

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

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.9. Explain Power of Quantum Computation? **(12.5)**

Ans. In classical complexity theory, the classes P and NP play a major role, but there are other classes of interset. Some of them are given below:

L: The class of all decision problems which may be decided by a TM running in logarithmic space.

PSPACE: The class of decision problems which may be decided on a TM using a polynomial number of working bits, with no limitation on the amount of time that may be used by machine.

Exp: The class of all decision problems which may be decided by a TM in exponential time, that is, $0(2^{n^k})$, k being a constant.

The hierarchy of these classes is given by

$L \leq P \leq NP \leq PSPACE \leq EXP.$

We also have two more classes.

BPP: The class of problems that can be solved using the randomized algorithm in polynomial time, if a bounded probability of error (say 1/10) is allowed in the solution of the problem.

BQP: The class of all computational problems which can be solved efficiently (in polynomial time) on a quantum computer where a bounded probability of error is allowed. It is easy to see that BPP ≤ BQP. The class BQP lies somewhere between P and PS PACE, but where exactly it lies with respect to P, NP and PSPACE is not known.

It is easy to give non-constructive proofs that many problems are in Exp, but it seems very hard to prove that a particular class of problems is in Exp. As for as quantum computation is concerned, two important classes are considered. One is BQP, which is analogous to BPP. The other is NPI (NP Intermediate) defined by.

NPI: The class of problems which are neither in P nor NP-complete.

Once again, no problem is shown to be in NPI. In that case P # NP is established. Two problems are likely to be in NPI, one being the factoring problem and the other being the graph isomorphism problems.

A quantum alogrithm for factoring has been discovered. Peter shor announced a quantum order finding algorithm and proved that factoring could be reduced to order-finding. This has motivated a search for a fast quantum algorithm for other problems suspected to be in NPI.

Grover developed an algorithm called the quantum search algorithm. A loose formulation of this means that a quantum computer can search a particular item in list of N items in $0(\sqrt{n})$ time and no further improvement is possible. If it were 0(log n), then a quantum computer can solve an NP-complete problem in an efficient way. Based on this, some researchers feel that the class BQP cannot contain the class of NP-complete Problems.

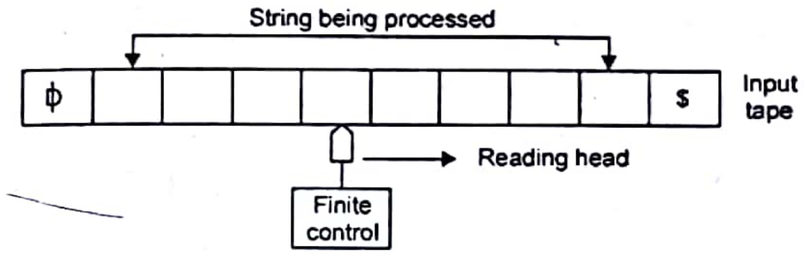
MODEL TEST PAPER-2
FIRST TERM EXAMINATION
FOURTH SEMESTER (B. TECH)
THEORY OF COMPUTATION [ETCS-206]

Time : 1.30 hrs. **M.M. : 30**

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

Q.1(*a*) Draw the block diagram of a finite automaton? **(2)**

Ans.



Block diagram of finite Automata

(*i*) Input Tape: Input tape is divided into squares, each square containing a single symbol from the I/P alphabet Σ. The left-to-right sequence of symbols b/w the two endmarkers is the input string to be processed.

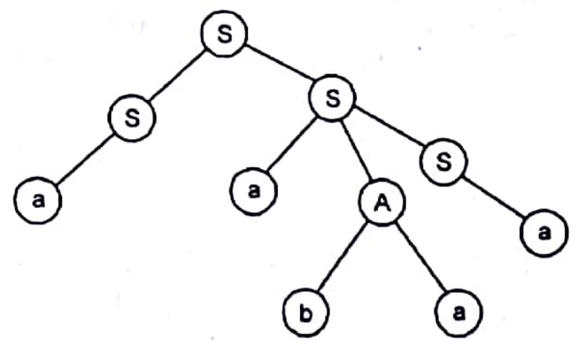
(*ii*) Reading Head: The head examines only one square at a time and can more one square either to the left or to the right. We restict the movement of the R-head only to right side.

(*iii*) Finite Control: The input to the finite control will usually be the symbol under the R-head, say a and present state of the m/c, say q, to give the next state of the FSM given by S(*q*, a).

Q.1(*b*) What do you mean by Derivation Tree? **(2)**

Ans. Any production can be represented in the form of tree is known as Derivation Tree *or* Syntax Tree *or* Parse Tree.

For example:



Q.1(*c*) What are the different closure properties of Regular Languages?(2)

Ans. Recall a closure properties is a statement that a certain operation on languages, when applied to languages in a class (*e.g.* the regular languages), produces a result that is also in that class.

Closure Properties of Regular Languages:

(i) UNION
(ii) Intersection
(iii) Difference
(iv) Concatenation
(v) Kleen Closure
(vi) Reversal
(vii) Homomorphism
(viii) Inverse Homomorphism

Q.1(d) What is regular expression? **(2)**

Ans. The regular expressions are useful for representing certain sets of strings in an algebric fashion. Actually these describe the languages accepted by finite state automata.

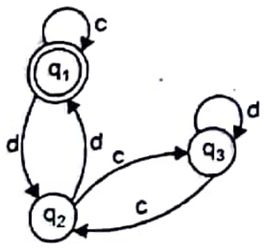
For example,

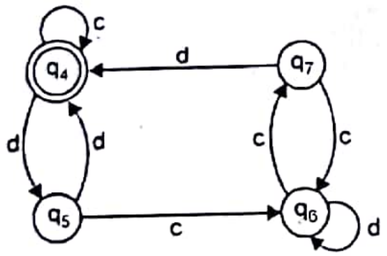
- Any terminal symbol, ^ and ϕ are regular expressions. When we view a in Σ as a regular expression, we denote it by a.
- If R is a regular expression, then (R) is also a regular expression.

Q.1(e) Is it true the language accepted by NFA is different from the regular language? Justify your answer? **(2)**

Ans. No, it is false. For every regular expression r there exists a NFA with ε-transition that accept $L(r)$.

Q.2(a) Consider the following two DFAs M and M′ over {0, 1} given in figure. Determine whether M and M′ are equivalent. **(5)**





Ans. The initial states in M and M' are q_1 and q_4 respectively. Hence the first element of the first column in the comparison table must be (q_1, q_4). The first element in the second column in (q_1, q_4) since both q_1 and q_4 are c-reachable from the respective initial states.

(q, q')	(qc, qc')	(qd, qd')
(q_1, q_4)	(q_1, q_4)	(q_2, q_5)
(q_2, q_5)	(q_3, q_6)	(q_1, q_4)
(q_3, q_6)	(q_2, q_7)	(q_3, q_6)
(q_2, q_7)	(q_3, q_6)	(q_1, q_4)

As we do not get a pair (q, q') where q is a final state and q' is a non-final state (or vice versa) at every row, we proceed untill all the elements in the second and third columns are also in the first column. Therefore, M and M' are equivalent.

Q.2(b) Show that $L = \{a^p / P$ is a prime$\}$ is not regular? **(5)**

Ans. Step 1: We suppose L is regular. Let n be the no. of states in the finite automaton accepting L.

Setp 2: Let P be a prime number greater than n. Let $w = a^P$. By pumping Lemma, w can be written as $w = xyz$, with $|xy| \leq n$ and $|y| > 0$. x, y, z are simply strings of a's. So $y = a^m$ for some $m \geq 1$.

Step 3: Let $i = p + 1$. Then $|xy^iz| = |xyz| + |y^{i-1}| = P + (i-1)m = P + Pm$. By pumping Lemma, $xy^iz \in$ L. But $(xy^iz) = P + Pm = P(1 + m)$, and $P(1 + m)$ is not a prime. So $xy^iz \notin$ L. This is a contradiction. Thus L is not regular.

Q.3(a) For every NFDA, there exists a DFA which simulates the behaviour of NDFA, alternatively, if L is the set accepted by NDFA, then there exists a DFA which also accepts L? **(10)**

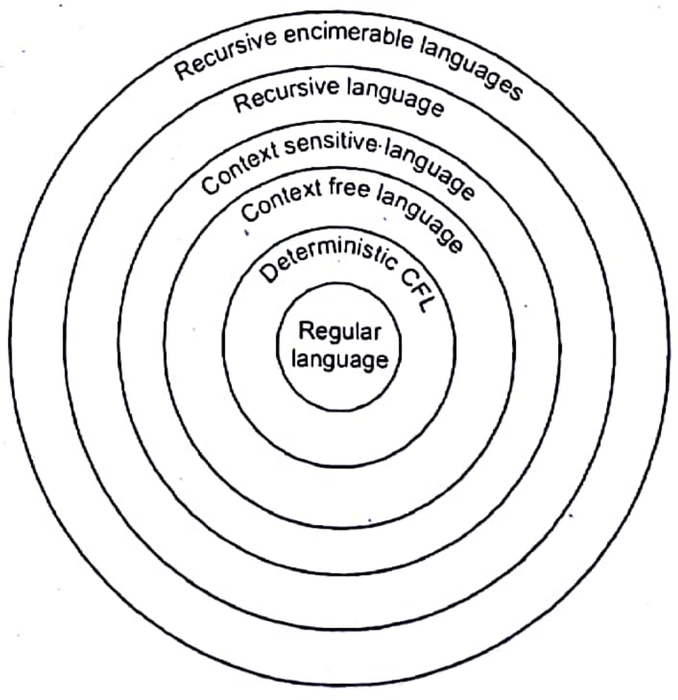
Ans. Refer to Model Test Paper-1, FTE in Q.No. 3(a)

Q.4(a) Explain Chowsky Hierarchy of Grammers? **(10)**

Ans. Chomsky Herarchy of Grammars: We can exhibit the relationship between grammars by the Chomsky Hierarchy. Noum Chomsky, a founder of formal Language theory, provided on initiql 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 α and β are arbitary string of grammar symbols with $\alpha \neq \in$. 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 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 α_1 and α_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.

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.

MODEL TEST PAPER-2
SECOND TERM EXAMINATION
FOURTH SEMESTER (B. TECH)
THEORY OF COMPUTATION [ETCS-206]

Time : 1.30 hrs. **M.M. : 30**

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

Q.1(*a*) Determine the type of grammar G? **(2)**

(*i*) $S \rightarrow aA, A \rightarrow aAB, B \rightarrow b, A \rightarrow a$

(*ii*) $S \rightarrow aAB, AB \rightarrow C, A \rightarrow b, B \rightarrow AB$

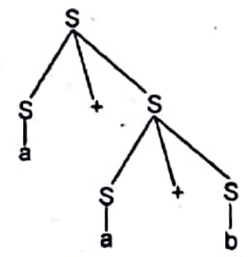
Ans. (*i*) This is type-2 grammar because every production on the left hand side has only one variable.

(*ii*) This is Type-0 grammar because in L.H.S. has at least one variable.

Q.1(*b*) Draw the parse tree for the string a + a • b i.e. for grammar? **(2)**

$$S \rightarrow S + S \mid S \bullet S \mid a \mid b$$

Ans.



Parse Tree for *a* + *a* * *b*

Q.1(*c*) What do you mean by Linear Bounded Automataon? **(2)**

Ans. The set of context-sensitive languages is accepted by the model and the infinite storage is restricted in size but not in accessibility to the storage in comparsion with the Turing Machine Model. It is called the Linear Bounded Automaton (LBA) because a Linear function is used to restict (to bound) the length of the tape. A linear Bounded automaton is a non-determinsitic turing m/c which has a single tape whose length is not infinite but bounded by a linear function of the length of the input string.

Q.1(*d*) What do you mean by CNF?

Ans. A context free grammar G is in chomsky normal form if every production is of the form $A \rightarrow a$ or $A \rightarrow BC$ and $S \rightarrow \wedge$ is in 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 $S \rightarrow AB/\wedge$, $A \rightarrow a$, $B \rightarrow b$. Then G is in chomsky normal form.

Q.1(*e*) What do you mean by Language Acceptability by turing machines? **(2)**

Ans. Let us consider the turing machine $M = (Q, \Sigma, \Gamma, S, q_0, b, F)$. A string ω in Σ^* is said to be accepted by M if low $q_0\omega \vdash^* \alpha_1 P \alpha_2$ for some $P \in F$ and $\alpha_1\,\alpha_2 \varepsilon \Gamma^*$.

M does not accept w if the machine M either halts in a non-accepting state or does not halt.

It may be noted that though there are other equivalent definations of acceptance by the Turning Machine.

Q.2. Show that L = $\{a^n b^n c^n \mid n \geq 1\}$ is not context-free but context sensitive? (10)

Ans. We have already constructed a context-sensitive grammar G generating L. We note that in every string of L, any symbol appears the same number of times as any other symbol. Also a cannot appear after b, and c cannot appear b and so on.

Step 1: Assume L is context-free. Let n be the natural number obtained by using the pumping Lemma.

Step 2: Let $Z = a^n b^n c^n$. Then $|Z| = 3n > n$. Write $z = uvwxy$, where $|vx| \geq 1$, i.e. at least one of v or x is not ^.

Step 3: $uvwxy = a^n b^n c^n$. As $1 \leq |vx| \leq n$, v or x cannot contain all the three symbols, a, b, c. So (i) v or x is of the form $a^i b^j$ (or $b^i c^j$) for same i, j such that $i + j \leq n$. Or (ii) v or x is a string formed by the repetition of only one symbol among a, b, c.

When v or x is of the form $a^i b^j$, $v^2 = a^i b^j a^i b^j$ (or $x^2 = a^i b^j a^i b^j$). As v^2 is a substring of uv^2wx^2y, we cannot have uv^2wx^2y of the form $a^m b^m c^m$. So, $uv^2wx^2y \notin L$.

When both v and x are formed by the repetition of a single symbol (e.g. $u = a^i$ and $v = b^j$ for some i and j, $i \leq n, j \leq n$), the string uwy will contain the remaining symbol, say a_1. Also, a_1^n will be a substring of uwy as a, does not occur in v or x. The number of occurences of one of the other two symbols in uwy is less than n (recall $uvwxy = a^n b^n c^n$), and n is the number of occurences of a_1. So $uv^0wx^0y = uwy \notin L$.

Thus for any choice of v or x, we get a contradiction. Therefore, L is not context-free.

Q.3(a) Explain the Partial Recursive function with example? (5)

Ans. Refer to Model Test Paper-1 in STE in Q.No. 4(a).

Q.3(b) Explain Primitive Recursive Function with example? (5)

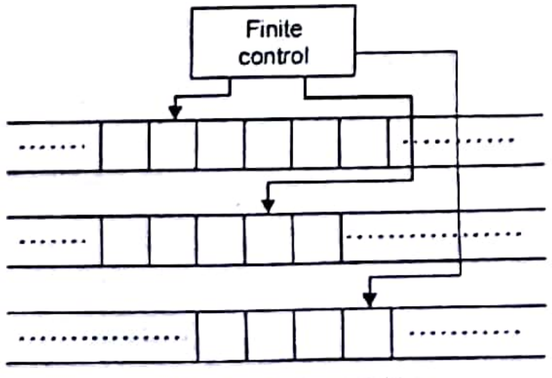
Ans. Refer to Model Test Paper-1 in STE in Q.No. 4(b).

Q.4(a) Explain Multitape Turing Machines? (5)

Ans. A multitape TM has a finite set Q of states, an initial state q_0, a subset F of Q called the set of final states, a set P of tape symbols, a new symbol b, not in P called the blank symbol.

There are K tapes, each divided into cells. The first tape holds the input string w. Initially, all the other tapes hold the blank symbol.

Initially the head of the first tape (input tape) is at the left and of the input w. All the other heads can be placed at any cell initially. δ is a partial function from $Q \times \Gamma^K$ into $Q \times \Gamma^K \times \{L, R, S\}^K$. We use implementation description to define δ. Figure represents a multitape TM. A move depends on the current state and K tape symbols under K tape heads.



Multitape turing machine

MODEL TEST PAPER-2
END TERM EXAMINATION
FOURTH SEMESTER (B. TECH)
THEORY OF COMPUTATION [ETCS-206]

Time : 3 hrs. **M.M. : 70**

Note: ***Attempt any one question from each unit Q.No. 1 is compulsory.***

Q.1(a) Give the formal statement of pumping Lemma for regular sets? (2)

Ans. Refer to Model Test Paper–1 in FTE in Q.No.1. (b)

Q.1. (b) What do you mean by ambiguous grammar?

Ans. Refer to Model Test Paper–1 in FTE in Q.No. 1. (e)

Q.1. (c) What is the difference between a recursive language and recursively enumerable language?

Ans. Refer to Model Test Paper-1 in STE in Q.No.1 (ii)

Q.1(d) Show that $L = \{a^i b^j c^k \mid k > i + j\}$ is not regular. (2.5)

Ans. We prove this by contradiction, assume $L = T(M)$ for some DFA with n states. Choose $w = a^n b^n c^{3n}$ in L. Using the pumping Lemma, we write $w = xyz$ with $|xy| \leq n$ and $|y| > 0$. As $w = a^n b^n c^{3n}$, $xy = a^i$ for some $i \leq n$. This means that $y = aj$ for some j, $1 \leq j \leq n$. Then $xy^{k+1}z = a^{n+jk} b^n c^{3n}$. Choosing K large enough so that $n + jk > 2n$, we can make $n + jk + n > 3n$. So, $xy^{k+1}z \notin L$. Hence L is not regular.

Q.1(e) Describe initial functions in primitive recursive functions? (2.5)

Ans. The initial functions over N:

- **Zero function:** Z defined by $Z(x) = 0$
- **Successor Function:** S defined by $S(x) = x + 1$
- **Projection Function :**

U_i^n defined by $U_i^n(x_1, \ldots\ldots x_n) = x_i$

Q.1(f) What is the relation between LBA and context-sensitive languages? (2.5)

Ans. The set of strings accepted by non-determistic LBA is the set of strings generated by the context-sensitive grammars, excluding the null strings. Now we give an important result:

If L is a context-sensitive language, then L is accepted by a Linear bounded automaton. The converse is also true.

Q.1(g) Define Arden's Theorem? (2.5)

Ans. Let P & Q be two regular expressions over Σ. If P does not contain ^, then the following equation in R, namely

$$\boxed{R = Q + RP}$$

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

Q.1(h) What is Moore and Melay machine? (2.5)

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.

Q.1(i) Whether NPDA and DPDA are equivalent? **(2.5)**

Ans. The languages accepted by NPDA and DPDA are not equivalent.

For example: WW^R is accepted by NPDA and not by any DPDA.

DPDA can not accept any ambiguous grammar and NPDA can accepted the ambiguous grammars.

Q.1(j) When a checking off symbols used in TM? **(2.5)**

Ans. Checking off symbols is useful method when a TM recognises a language with repeated strings $ab = nd$ also to compare the length of substrings.

Example: $[ww / w \in \Sigma^*]$ or $\left\{\frac{a^i b^i}{i} \geq 1\right\}$

This is implemented by using on extra track on the tape with symbols blank.

UNIT–I

Q.2(a) Design DFA for the Language **(6)**

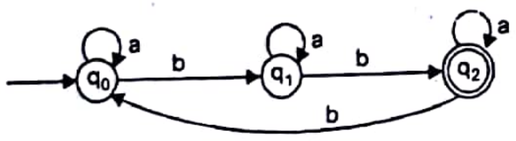
$$L = \{W \in (a, b)^* / n_b(w) \bmod 3 > 1\}$$

Ans. Let us first analyse the language L. $n_b(w)$: no. of b's in every string of long. $n_b(w)$ mod 3 : no. of b's in string is divided by three and remainder is result of $n_b(w)$ mod 3. $n_b(w)$ mod 3 > 1 means that remainder is only 2 since when any number is divided by 3 then remainder may be only 0, 1 and 2.

Let

$M = (Q, \Sigma, S, 20, F)$ be FA

$\Sigma = \{a, b\}$



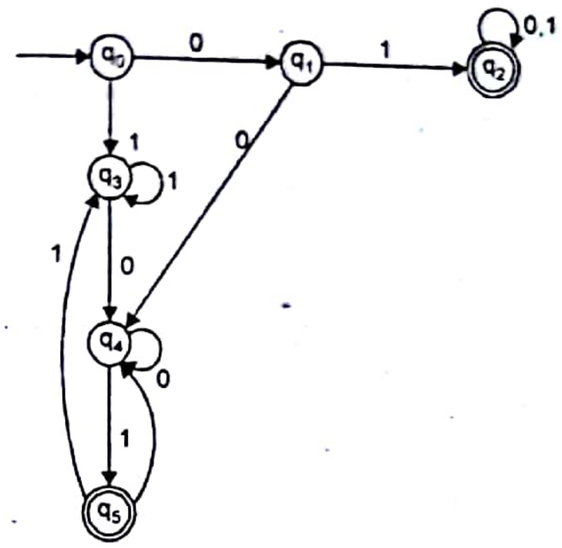
It will accept string, having 2b's, 5b's and 8b's so remainder is always two.

$Q = \{q_0, q_1, q_2\}$

$F = \{q_2\}$

Q.2(b) Design a FA over alphabet $\Sigma = \{0, 1\}$ which accept the set of strings either start with 01 or end with 01.

Ans. By the analysis of proble, it is clear that FA will accepts the strings such as 01, 011111, 01000.....



Let FA $M = \{Q, \Sigma, S, q_0, F\}$

$\Sigma = \{0, 1\}$ given

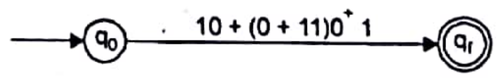
q_0 is initial state

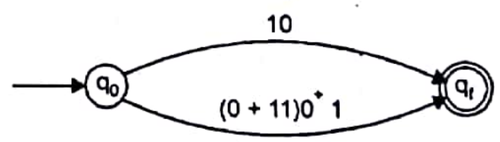
$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

$F = \{q_2, q_5\}$

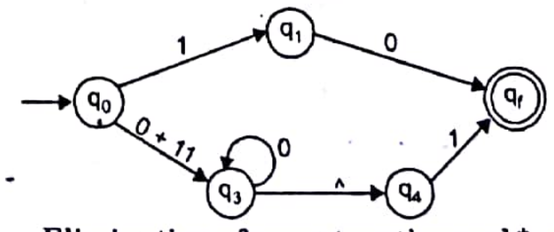
Q.3(a) Construct a DFA with reduced states equivalent to the r.e. 10 + (0 + 11) 0 * 1. **(12.5)**

Ans. Step 1: (Construction of NDFA) the NDFA is constructed by eliminating the operation +, concatenation and *, and the ^-moves in successive steps. The step-by-step construction is given

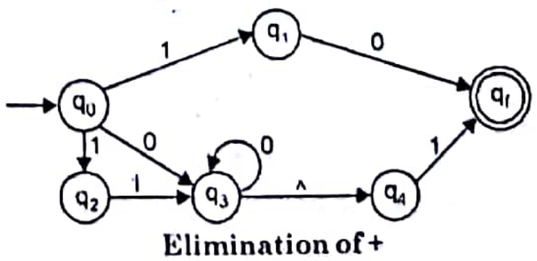




Elimination of +



Elimination of concatenation and *



Elimination of +

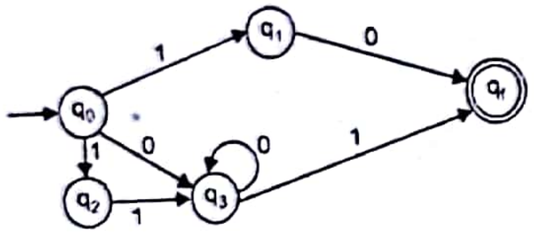


Fig. 1. Elimination of ∧-moves.

Step 2: (Construction of DFA) for the NDFA given in Fig. 1, the corresponding transition table is defined in Table 1.

Table 1 : Transition Table for Fig. 1.

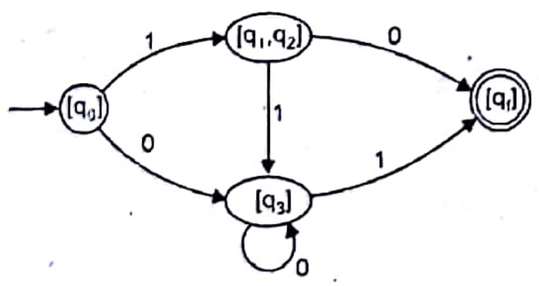
State/Σ	0	1
$\rightarrow q_0$	q_3	q_1, q_2
q_1	q_f	–
q_2	–	q_3
q_3	q_3	q_f
q_f	–	–

In Table 2 the columns corresponding to $[q_f]$ and ϕ are identical. So we can identify $[q_f]$ and ϕ.

Table 2 : Transition Table of DFA

Q	Q_0	Q_1
$\rightarrow [q_0]$	$[q_3]$	$[q_1, q_2]$
$[q_3]$	$[q_3]$	$[q_f]$
$[q_1, q_2]$	$[q_f]$	$[q_3]$
$[q_f]$	ϕ	ϕ
ϕ	ϕ	ϕ

The DFA with the reduced no. of states corresponding to Table 2 is defined by Fig. 2.



Reduced DFA

UNIT-II

Q.4. Find a GNF grammar equivalent to the following CFG.

S → AA/a

A → SS/b

Ans. Refer to Model Test Paper-1 in FTF in Q.No. 5.

Q.5. Consider the context Free Grammar G.

S → AB
A → a
B → C/b
C → D
D → E
E → a

Remove the unit Production?

Ans. Given CFG
S → AB
A → a
B → C/b
C → D
D → E
E → a

Contain three UNIT productions
B → C
C → D
D → 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 → AB
A → a
B → C/b
C → D
D → G
E → a

Now we can remove → D by using D → a, we get
S → AB
A → a
B → C/b
C → a
D → a
E → a

Similarly, we can remove B → C by using C → a, we obtain
S → AB
A → a
B → a/b
C → a
D → a

$$E \rightarrow a$$

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,

$$S \rightarrow AB$$
$$A \rightarrow a$$
$$B \rightarrow a/b$$

which is completely reduced grammar.

UNIT-III

Q.6. Explain these exists a recursively enumerable language whose complement is not recursively enumerable?

Ans. Let $\Sigma = \{a\}$, and conser the set of all Turing machines with this input alphabet.

This set is countable, so we can, assosiate an order $M_1, M_2, \ldots$ with its elements. For each Turning machine M_i, there is an associated recursively enumerable Language $L(M_i)$. Conversely, for each recursively enumerable language on Σ, there is some TM that accepts it.

We now consider a new language L defined as follows. For each $i \leq 1$, the string a^i in L if and only if $a^i \in L(M_i)$. It is clear that the language L is well defined, since the statement $a^i \in L(Mi)$ and hence $a^i \in L$, must either be true or false. Next, we consider the complement of L,

$$\bar{L} = \{a^i : a^i \notin L(Mi)\}$$

Consider the string a^K. Is it in L or in $\bar{L}$? Suppose that $a^K \in \bar{L}$. By this implies that

$$a^k \in L(M_K)$$

But now implies that

$$a^K \notin \bar{L}$$

Conversely, if we assume that a^k is in L, then $a^k \notin \bar{L}$ and implies that

$$a^k \notin L(M_K)$$

But then from, we get that

$$a^k \in \bar{L}$$

The contradiction is inescapable, and we must conclude that our assumption that $\bar{L}$ is recursively enumerable is false.

To complete the proof of theorem as stated, we must still show that L is recursively enumerable. We can use for this the known enumeration procedure for MM. Given a^i, we first find i by counting the number of a's, we first find i by counting the number of a's. We then use the enumeration procedure for Turing machines to find Mi. Finally, we give its description along with a^i to a universal. TM M_u that simulates the action of M an a^i. If a^i is in L, the computation carried out by M_u will eventually halt. The combined effect of this is a TM that accepts every $a^i \in L$. Therefore, by defination, L is recursively enumerable.

Q.7. Explain a languages that are not Recursively enumerable? (12.5)

Ans. Refer to Model Test Paper-1 in ETE in Q.No. 7.

UNIT-IV

Q.8(a) What is the importance of NP-complete problems?

Ans. We proved theorem regarding the properties of NP-complete problems. At the beginning of this concept we noted that the computer scientistis and mathematics strongly believe that P ≠ NP. At the same time, no problem in NP is proved to be in P. The entire complexity theory rests on the strong belief that P ≠ NP.

If P_1 is NP-complete and there is a polynomial time reduction of P_1 to P_2 then P_2 is NP-complete says that to extend the class of NP-complete problem, while If *some NP=complete problem is in P, then P = NP* asserts that the existence of one NP-complete problem admitting a polynomial-time algorithm will prove P = NP.

We will prove the existence of an NP-complete problem. We will give a list of NP-complete problem. Thousands of NP-complete problems in various branches such as Operations Research, Logic, Graph Theory, Combination etc. have been constructed so far. A polynomial time algorithm for any one of there problems will yield a proof of P = NP. But such multitude of NP-complete problems only strengthens the belief of the computer scientists that P ≠ NP.

Q.8(b) What do you mean by Polynomial time reduction and NP-completeness with example?

Ans. Polynomial Time Reduction: Let P_1 and P_2 be two problems. A reduction from P_1 to P_2 is an algorithm which converts an instance of P_1 an algorithm which converts an instance of P_1 to an algorithm which converts an instance of P_1 to an instance of P_2. If the time taken by the algorithm is a polynomial $P(n)$, n being the length of the input of P_1, then the reduction is called a polynomial reduction P_1 to P_2.

NP-completeness: Let L be a language or problem in NP. Then L is NP-complete if

(*i*) L is in NP.

(*ii*) For every language L' in NP there exists a polynomial time reduction of L' to L.

Q.9. Explain Cook's Theorem?

Ans. Refer to Model Test Paper-1 in ETE in Q.No. 8.

MODEL TEST PAPER-3
FIRST TERM EXAMINATION
FOURTH SEMESTER (B. TECH)
THEORY OF COMPUTATION [ETCS-206]

M.M. : 30

Time : 1.30 hrs.

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

Q.1.(a) Write any three applications of automat theory?

Ans. (*i*) It is base for formal language and these formal languages are useful of the programming language.

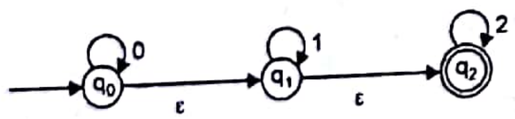
(*ii*) It plays an important role in compiler design.

(*iii*) To prove the correctness of the program automata theory is used.

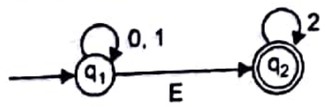
(*iv*) In switching theory and design and analysis of digital circuits automata theory is applied.

(*v*) It deals with design finite state machines.

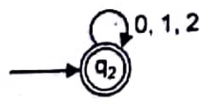
Q.1. (b) Obtain the NFA without ε transition to the following NFA with ε transition?



Ans. Remove ε-transitionfrom q_0 to q_1.



Now remove transition from q_0 to q_2. As q_0 to q_2 is transition q_0 will become start and final state both.



Q.1. (c) Write short notes on Minimization of DFA?

Ans. • Reducing the no. of states from given FA.

• First find out which two states are equivalent we than replace those two states by one representative state.

• For finding the equivalent states we will apply the following rule.

The two states s_1 and s_2 are equivalent if and only if both the states are signal or non-final state.

Q.1. (d) Show that $(r+s)^* \neq r^* + s^*$

Ans. L.H.S.

$$(r+s)^* = \{\varepsilon, r, s, rs, sr, ...\}$$

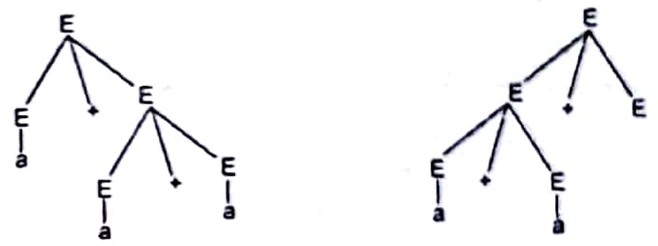
R.H.S $\quad r^* + s^* = \{\varepsilon, r, rr....\} \cup \{\varepsilon, s, ss, ...\}$

$= \{\varepsilon, r, rr, s, ss....\}$

Hence L.H.S. ≠ R.H.S. is proved.

Q.2. (a) How we can remove ambiguity from grammars?

Ans. 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 + 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 assosiativity it of + operator is not respected. Since, we assume the + operator is left assosiative 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.3. (a) Explain chomsky Hierarchy of grammers?

Ans. Model Test Paper-2 in FTE in Q. No. 4.

Q.4. Construct a DFA with reduced states equivalent to the r.e. 10 + (0 + 11) 0* 1.

Ans. Refer to Model paper-2 in ETE Q.No. 3.

MODEL TEST PAPER-3
SECOND TERM EXAMINATION
FOURTH SEMESTER (B. TECH)
THEORY OF COMPUTATION [ETCS-206]

Time : 1.30 hrs. **M.M. : 30**

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

Q.1.(a) Show that id + id * id can be generated by two district leftmost derivation in the grammar E → E + E/E * E/(E)/id.

Ans. (i)
$$E \to E + E$$
$$\to id + E$$
$$\to id + E * E$$
$$\to id + id * E$$
$$\to id + id * id$$

(ii)
$$E \to E * E$$
$$\to E + E * E$$
$$\to E + E * id$$
$$\to E + id * id$$
$$\to id + id * id$$

We showed that id + id * id can be generated by two distinct LMD.

Q.1. (b) What is the additional feature PDA has when compared with NFA? Is PDA superior Over NFA in the sense L acceptance? Justify your answer?

Ans. PDA is superior NFA by having the following additional features:

- stack which is used to store the necessary tape symbols and use the state to remember the conditions.
- Two ways of L acceptances, one by reaching its final state and another by emptying its stack.

Q.1. (c) Define Modifed PCP?

Ans. Given lists A and B, of K strings each from Σ^*, say.

$$A = w_1, w_2, w_3 \ldots w_k$$
$$B = x_1, x_2, x_3, \ldots, x_k$$

Does there exist a sequence of integers $i_1, i_2, \ldots i_r$ such that $w_{i_1}, w_{i_2}, w_{i_3} \ldots w_{im} = x_{i1}, x_{i2} \ldots x_{ir}$.

The sequence of $i_1, i_2, \ldots i_m$ is a solution to this instance of PCP.

Q.1. (d) What are the properties of recursive and recursively enumerable language?

Ans.
- The complement of a recursive language is recursive.
- The union of two recursive language are recursive. The union of two RE languages are RE.

- If a language L and complement L are both RE. then L is recursive.

Q.1. (e) What do you mean by linear Bound automation?

Ans. Refer to Model Test Paper-2 in STE in Q.No. 1. (c)

Q.2. Reduce the following grammar G into CNF:

$$S \to aAD$$
$$A \to aB/bAB$$
$$B \to b$$
$$D \to d$$

Ans. Step 1: As there are no null productions or unit productions, we can proceed to step 2.

Step 2: Let $G_1 = (V', \{a, b, d\}, P_1, S)$, where P_1 and V are constructed as follows:

(i) $B \to b$, $D \to d$ are included in P_1.

(ii) $S \to aAD$ gives rise to $S \to CaAD$ and $Ca \to a$.

$A \to aB$ gives rise to $A \to CaB$.

$A \to bAB$ gives rise to $A \to C_bAB$ and $C_b \to b$

$$V' = \{S, A, B, D, Ca, Cb\}$$

Step 3: P_1 consists of $S \to CaAD$, $A \to CaB/C_bAB$, $B \to b$, $D \to d$, $Ca \to a$, $C_b \to b$.

$A \to C_aB$, $B \to b$, $D \to d$, $Ca \to a$, $C_b \to b$ are added to P_2.

$S \to CaAD$ is replaced by $S \to CaC_1$ and $C_1 \to AD$.

$A \to C_bAB$ is replaced by $A \to C_bC_2$ and $C_2 \to AB$.

Let $G_2 = (\{S_1 A, B, D, C_a, C_b, C_1, C_2\} \{a, b, d\}, P_2, S)$

where P_2 consists of $S \to C_a C_1$, $A \to C_a B/C_b C_2$,

$C_1 \to AD$, $C_2 \to AB$, $B \to b$, $D \to d$, $Ca \to a$,

$C_b \to b$. G_2 is in CNF and equivalent to G.

Q.3. Construct a PDA A equivalent to the following CFG S → OBB, B → 0S/1S/0. Test whether 010^4 is in N(A).

Ans. Define PDA A as follows:

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

δ 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

$\vdash (q_1\ 10^4, 1SB)$ 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

$\vdash$ (q, 0^3, 000) by Rule R_2

$\vdash$ (q, ^, ^) by Rule R_3

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

Q.4. Show that $L = \{a^n b^n c^n / n \geq 1\}$ is not context Free but context sensitive?

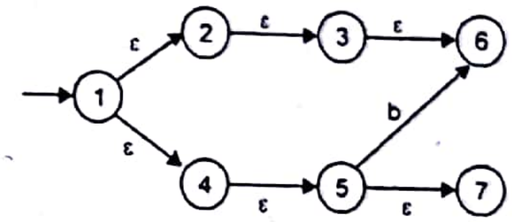
Ans. Refer to Model Test Paper-2 in STE in Q.No. 2.

MODEL TEST PAPER-3
END TERM EXAMINATION
FOURTH SEMESTER (B. TECH)
THEORY OF COMPUTATION [ETCS-206]

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

Note: *Attempt any one question from each unit and Question No. 1 is compulsory.*

Q.1.(a) Find the ε-closure of the states 1, 2, 4 in the following transition diagram?



ε-Closure (1) = {1, 2, 3, 4, 6}

ε-Closure (2) = {2, 3, 6}

ε-Closure (4) = {4, 5, 6}

Q.1. (b) What are the applications of pumping Lemma?

Ans. Pumping Lemma is used to check if a language is regular or not.

- Assume that the language (L) is regular.
- Select a constant 'n'.
- Select a string (z) in 2, such that |z| > n.
- Split the word z into u, v and w such that |uv| <= n and |v| >= 1.
- You achieve a concatenation to pumping lemma that there exists an 'i' such that $uv^i w$ is not in 2, Then 2 is not a regular language.

Q.1. (c) What is the procedure conversion from automata to regular expression using arden's theorem?

Ans. Arden's theorem is used to find regular expression from the DFA. Using this theorem if the equation is of the form R = Q + RP, we can write this as

$$R = QP^*$$

- Write the equations for all the states.
- Apply Arden's theorem and eliminate all the states.
- Find the equation of the final state with only the input symbols.
- Mode the simplification if possible.
- The equation obtained is the required regular expression.

Q.1. (d) Find $L(G)$, where G {{S}, {0, 1} {$S \rightarrow OS1, S \rightarrow \varepsilon$}, S)

Ans. From the given grammar G

- $S \rightarrow 0S1$
- $S \rightarrow \varepsilon$

$S \rightarrow 0S1$

$\rightarrow 00S11\ (S \rightarrow 0S1)$

$\rightarrow 0011\ (S \rightarrow \varepsilon)$

$S \rightarrow 0S1$

$\rightarrow 01\ (S \rightarrow \varepsilon)$

$\rightarrow 0^n\ 1^n$

Hence $L(G) = \{0^n\ 1^n\ 1^n / n \geq 0\}$

Q.1. (e) What are the uses of CFG?

Ans. • Construction of compilers.

• Simplified the defination of programming.

• Describes the arithmetic expressions with arbitrary nesting of balanced parenthesis.

• Describes block structure in programming languages.

• Model neural nets.

Q.1. (f) Compare NPDA and DPDA?

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.

• The PDA is deterministic in the scnse, that at most one move is possible from any ID.

Q.1. (g) When a recursively enumerable language is said to be recursive. Is it true that the language accepted by a NDTM is different from recursively enumerable language?

Ans. • A language L is recursively enumerable if there is a TM that accepts L, and recursive if there is a TM that recognises L-Turning acceptable or Turing decidable language.

• No, the language accepted by NDTM is same as recursively enumerable language.

Q.1. (h) What are the actions take place in a Turing machine in one more?

Ans. In one more, the machine examines the present symbol under the R/W head on the tape and the present state of automata to determine.

(*i*) A new symbol to be written on the tape in the cell under the R/W head.

(*ii*) A motion of the R/W head along the tape either the head moves one cell. left (L) or one cell right (R).

(*iii*) The next state of automation.

(*iv*) Whether to halt or not.

Q.1. (i) What are the different types of a TM?

Ans. • Two way finite tape TM.

• Multi tape TM.

• Non deterministic TM.

• Multi-dimensional TM.

• Multi-head TM.

Q.1. (j) When we say a problem is decidable? Give an example of undecidable problem?

Ans. A problem whose language is recursive, is said to be decidable, otherwise it is undecidable. Decidable problems have an algorithm that takes the input, as as instance of the problem and determines whether the answer to that instance is "yes" or "no". For example, halting problems are undecidable problems.

Unit–I

Q.2. Q. Prove that for any transition function δ and for any two input strings *x* and *y*.

$$\delta(q, xy) = \delta(\delta(q, x), y) \qquad \text{...(1)}$$

Ans. By the method of induction on $|y|$ i.e. Length of y.

Basis: When $|y| = 1, y = a \varepsilon z$

L.H.S. of (1) $= \delta(q, xa)$

$= \delta(\delta(q, x), a)$ by property

$=$ R.H.S.

Assume the result i.e. for all strings *x* and strings y with $|y| = n$. Let y be a string of Length n + 1. Write $y = y_1 a$ where $|y_1| = n$.

L.H.S. of (1) $= \delta(q, xy, a) = \delta(q, x_1, a), x_1 = xy_1,$

$= \delta(\delta(q, x), a)$ by property

$= \delta(\delta(q, xy), a)$

$= \delta(\delta(\delta(q, x), y_1), a)$ by induction hypothesis

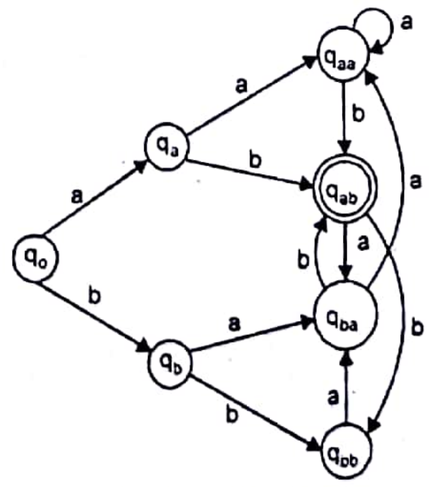
R.H.S. of (1) $= \delta(\delta(q, x), y_1, a)$

$= \delta(\delta(\delta(q, x), y_1), a)$ by property

Hence L.H.S. = R.H.S. This proves (1) for any string y of Length n + 1. By the principle of induction, (1) is true for all strings.

Q.3. Construct a DFA accepting all strings over {a, b} ending in ab.

Ans. We require two transitions for accepting the string ab. If the symbol b is processed after aa or ba, then also we end in ab. So we can have states for remembering aa, ab, ba, bb. The state corresponding to ab can be the final state in our DFA. Keeping these in mind we construct the required DFA. Its transition diagram is described as:



DFA for ending in ab

Q.4. Consider the CFG G.
$S \to AB, A \to a, B \to C/b, C \to D, D \to E, E \to a$ Remove the Unit production?

Ans. Refer to Model Test Paper-2 in ETE in Q.no. (5)

Q.5. Is it possible for a regular grammar to be ambiguous?

Ans. Let $G = (V_N, \Sigma, P, S)$ be regular. Then every production is of the form $A \to aB$ or $A \to b$. Let $\omega \in L(G)$. Let $S \stackrel{*}{\Rightarrow} \omega$ be a left most derivation. We prove that any leftmost derivation $A \stackrel{*}{\Rightarrow} w$, for every $A \in V_N$ is unique by induction on $|w|$. If $|w| = 1$, then $w = a \in T$. The only production is $A \to a$. Hence there is basis for induction. Assume that any leftmost derivation of the form $A \stackrel{*}{\Rightarrow} w$ is unique then $|w| = n - 1$. Let $|w| = n$ and $A \stackrel{*}{\Rightarrow} w$ be a Leftmost derivation.

Take $w = aw$, $a \in T$. Then the first step of $A \stackrel{*}{\Rightarrow} w$ has to be $A \Rightarrow aB$ for some $B \in V_N$. Hence the LMD $A \stackrel{*}{\Rightarrow} w$ can be shift into $A \Rightarrow aB \stackrel{*}{\Rightarrow} aw_1$. So, we get a leftmost derivation $B \stackrel{*}{\Rightarrow} w_1$. By induction hypothesis, $B \stackrel{*}{\Rightarrow} w_1$ is unique. So, we get a unique left most derivation of w. Hence a regular grammar cannot be ambiguous.

UNIT–III

Q.6. Design a TM that accepts

$$L = \{0^n 1^n / n \geq 1\}$$

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.

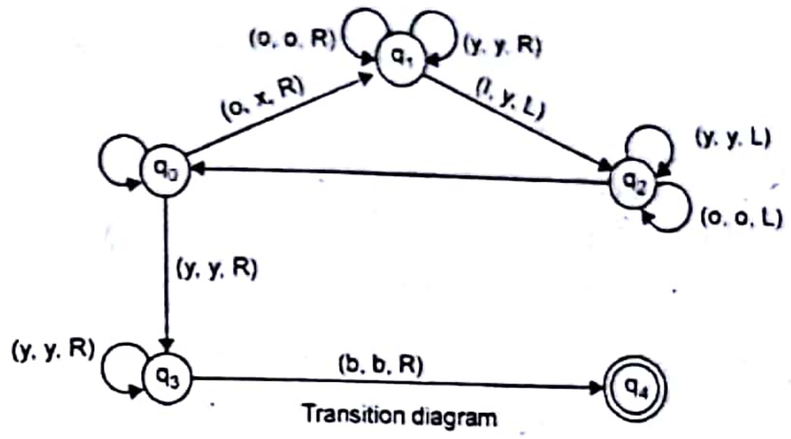
(*b*) Repeat (a) with the leftmost 0. If we move back and forth and no 0 or 1 remains, move to a final state.

(*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:–

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

where
$$Q = \{q_0, q_1, q_2, q_3, q_f\}$$
$$F = \{q_f\}$$
$$\Sigma = \{0, 1\}$$
$$\Gamma = \{0, 1, x, y, b\}$$

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

$q0011 \vdash xq_1, 001 \vdash x0\, q_1\, 11 \vdash xq_2\, 0y1$
$\vdash q_2\, x0y_1 \vdash xq_0\, 0y1 \vdash xx\, q_1\, y1 \vdash xxyq_1\, 1$
$\vdash xx\, q_2\, yy \vdash xq_2\, xyy \vdash xx\, q_0\, yy \vdash xx\, y\, q_3\, y$
$\vdash xx\, yy\, q_3 \vdash xx\, yy\, q_3\, b \vdash xx\, yy\, b\, q_4\, b$

Hence 0011 is accepted by M.

$q_0\, 010 \vdash xq_1\, 10 \vdash q_2\, xy0 \vdash x\, q_0\, y0 \vdash xy\, q_3 0$

As $\delta(q_3, 0)$ is not defined, M habts. So 010 is not accepted by M.

Q.7. Explain there exists a recursively enumerable language whose complement is not recursively enumerable?

Ans. Refer to Model Test Paper-2 in ETE in Q.No.6.

UNIT–IV

Q.8. Explain CHURCH-TURING Thesis?

Ans. Since 1970s many techniques for controlling the single quantum systems have been developed but with only modest success. But an experimental prototype for performing quantum cryptography, even at the initial level may be useful for some real-world applications.

Recall the church-Turing Thesis which asserts that any algorithm that can be performed on any computing machine can be performed on a TM as well.

Miniaturization of chips has increased the power of the computer. The growth of computer power is now described by moore's law, which states that the computer power will double for constant cost once in every two years. Now it is felt that a limit to this doubling power will be reached in two or three decades, since the quantum effects will begin to interface in the functioning of electronic devices as they are made smaller and smaller. So, efforts are on to provide a theory of quantum computation which will compensate for the possible failure of the Moore's law.

As an algorithm requiring polynomial time was considered as an efficient algorithm, a strengthened version of the church-Turing thesis was enunciated.

Any algorithm process can be simulated efficiently by a TM. But a challenge to the strong church-Turing thesis arose from analog computation. Certain types of analog computers solved some problems efficiently whereas these problems had no efficient

solution on a Turing machine. But when the presence of noise was taken into account, the power of the analog computers disappeared.

Q.9. (a) What are the importance of NP-Computer Problems?

Ans. Refer to Model-Test Paper-2 in ETE in Q.No. 8(*a*).

Q.9. (b) What do you mean by Polynomial time reduction and NP-completeness with example?

Ans. Refer to Model Test Paper-2 in ETE in Q.No. 8 (*b*).

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 Σ^*	(*i*) It is denoted by Σ^+.
(*ii*)	The set of strings, include the empty string over an alphabet Σ.	(*ii*) The set of string, exclude The empty string over on alphabet Σ.
(*iii*)	**For exapmple:-** $\Sigma = \{0\}$ $\{o\}^* = \{\varepsilon, 0, 00, 000...\}$	(*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: ϕ, {} and {*a*} for any symbol a belongs to (ε) 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:

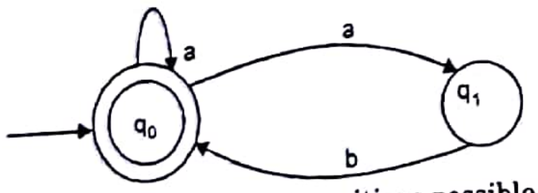
- a for some $a \in \Sigma$.
- ϵ if $R = \epsilon$,
- ϕ if $R = \phi$,
- (reverse (R_1) U reverse (R_2), if $R = R_1 U R_2$
- (reverse (R_2) o reverse (R_1), if $R = R_1 o R_2$ or
- (reverse $(R_1{}^*)$, if $R = (R_1{}^*)$

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_0' for M', and from q_0', 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.

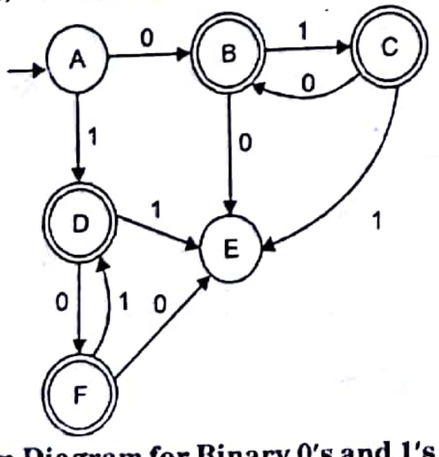
$$(01)^*0 + (10)^* 1$$

(*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:

δ (A, 0101)
— δ (B, 0101)
— δ (C, 01)
— δ (B, 1)
— C [Final State]

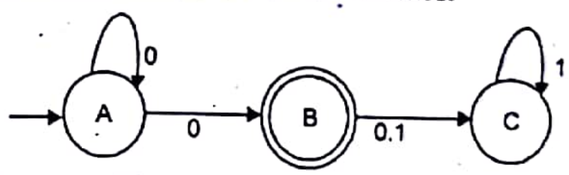
So, 0101 is acceptable string for designing automata.

Non-acceptable string 011:

δ (A, 011)
— δ (B, 11)
— δ (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 [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 $i \geq 0$.

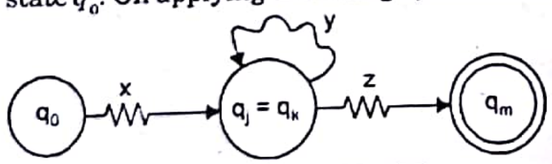
Proof: Let

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

That is, Q_1 is the sequence of states in the path with path value $w = a_1 a_2 \ldots a_m$. 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 $a_1 a_2 \ldots a_j$, $a_{j+1} \ldots a_k$ and $a_{k+1} \ldots a_m$. Let x, y, z denote these strings $a_1 a_2 \ldots a_{j+1} \ldots a_k, a_{k+1} \ldots a_m$, 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, $y \neq \wedge$.

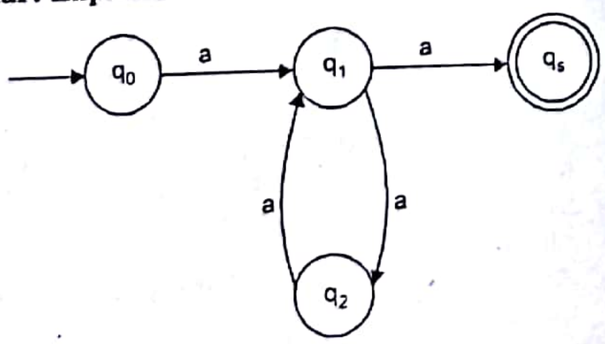


String accepted by M

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

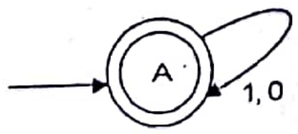
Ans. We can write a^{2n} as $a\,(a^2)^i\,a$, where $i \geq 0$.

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 ^- 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

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

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

$$\alpha \to \beta$$

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

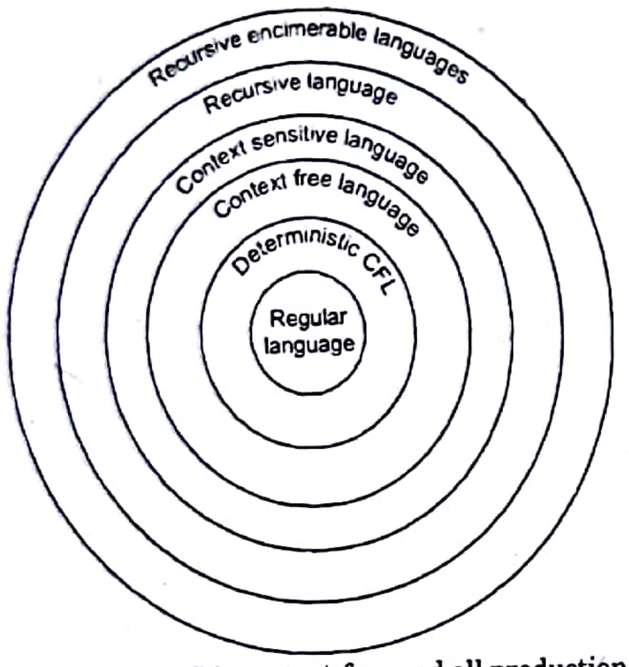
Context Sensitive Grammar: P is the set of rules called productions defined as defined as

$$\alpha \to \beta$$

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 $\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2$, with $\beta \neq \epsilon$. Replacement of variable A by string β is permitted in the "context" of α_1 and α_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 Σ. 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: $Q + (QP^*)P = Q(\wedge + P^*P) = QP^*$

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

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

From Eq (1)

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

We now show that any solution of Eq...(1) is equivalent to QP^*. Suppose R satisfies *Eq*...(1) is it satisfies Eq-2. Let w be a string of Length i in the set R. The w belongs to the set $Q(\wedge + p + p^2 + ...+p^i) + RP^{i+1}$. 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 $Q(\wedge + P+P^2 + ...+ P^i)$ 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 $Q(\wedge + P + P^2 +.....+ P^k)$. 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 $\alpha \in (V_N U \Sigma)^*$.

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.

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

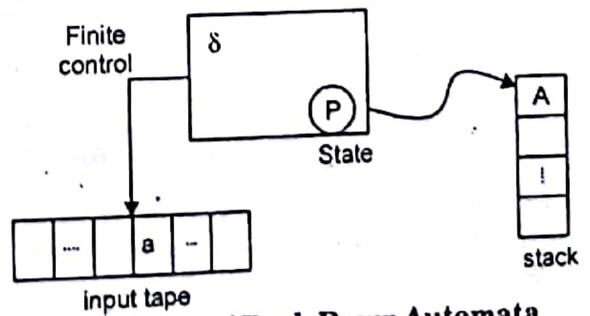
Q.1. (b) Differentiate between finite Automator and pushdown Automata.

Ans.

Finite Automata	Push Down Automata
1. A DFA is a "state" machine. 2. DFA and NFAs cannot do that stuff, but any DFA or NFA can also be represented as a push down automata.	1. A PDA is a "stack" machine 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) τ is a finite nonempty set of tape symbols

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

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

(v) δ 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

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

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

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

Solution:

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

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

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

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

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

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

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

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: $\delta(q_1, \wedge, zo) = \{(q_f, zo)\}$

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

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–

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

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

We will simulate this PDA for the following string:

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

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.

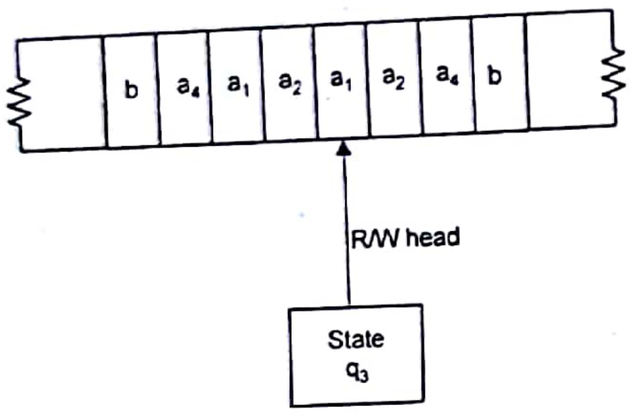
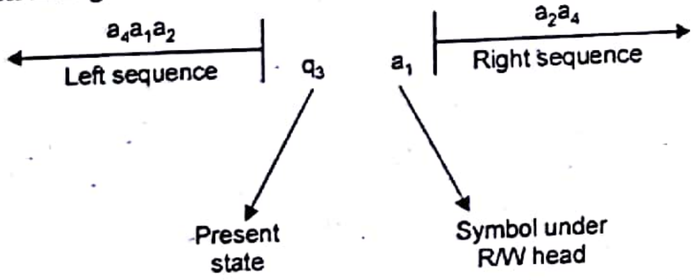


Fig. Snapshot of Turing Machine

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.

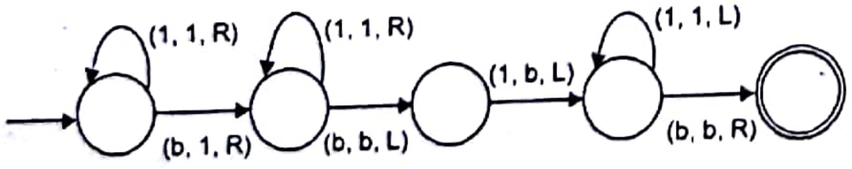
(*ii*) We observe that the blank symbol may occur as par of the ceft or right substring.

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.

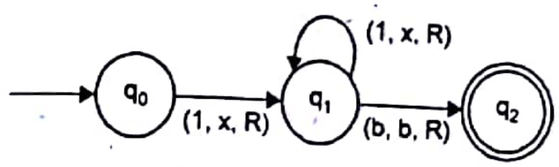
Ans.



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

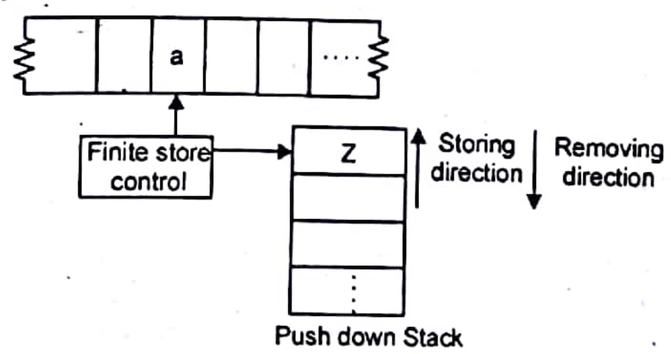
Present State	Input-tape 1	symbols b
$\rightarrow qo$	$x\,r\,q_1$	–
q_1	$xR\,q_1$	$bR\,q_2$
q_2	–	–

Transition Table for 11*

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

Q.(*i*) Pushdown stack

Ans.



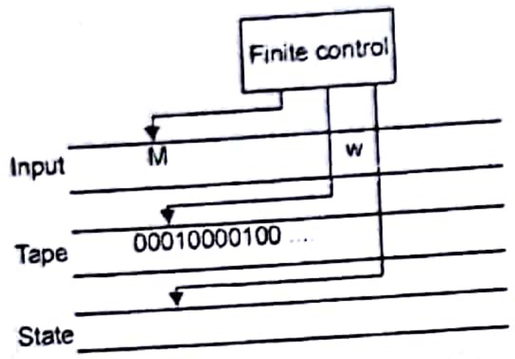
Model of push down Automata

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.



• 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:

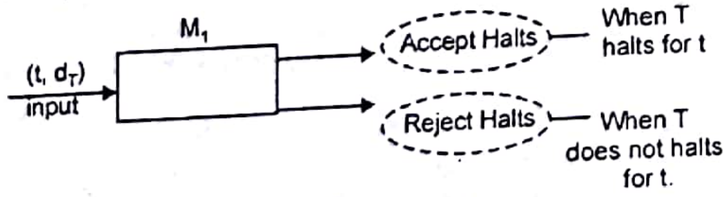
Ans. Halting Problem: To state halting problem we will consider the given configuration of a turing machine.

The output of TM can be:

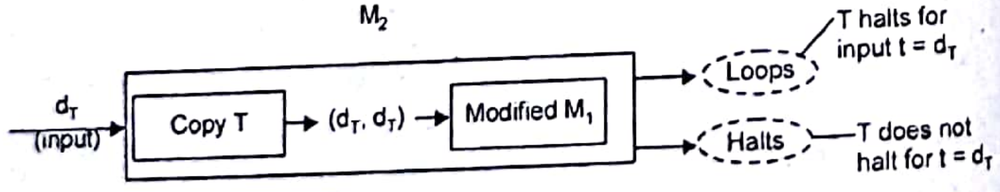
• **Halt:** The machine starting at this configuration will halt after a finite number of states.

• **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 $\beta \neq \epsilon$. Replacement of variable A by string β is permitted in the "context" of α_1 and α_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 represenles 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:

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

δ 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

$\vdash (q, 0^3, 000)$ by Rule R_2

$\vdash (q, \wedge, \wedge)$ by Rule R_3

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

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 $G_2 \in L_2$. The G_1 can be given as $G_1 = \{V_1, \Sigma, P, S,\}$ where P_1 can be given as

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

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

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

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

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.

$$S \to AB$$
$$A \to 0A / 0$$
$$B \to 1\,B2 / 12$$

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 $\in$ and single characters from Σ. 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.

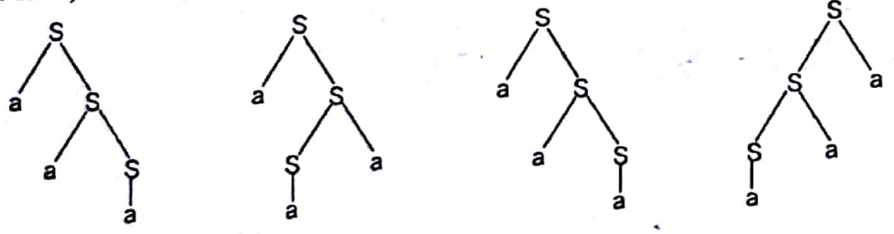
Q.3. (b) Define Ambiguity in Grammer and discuss the procedure to remove the ambiguity from the grammer. **(6.25)**

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:

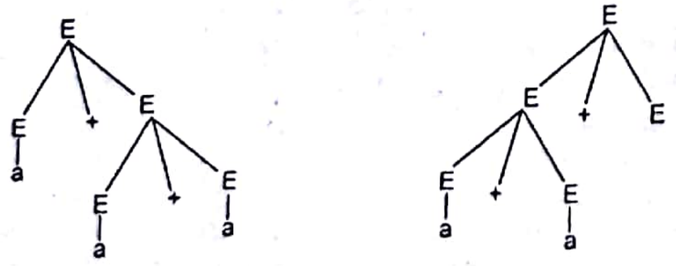
$S \rightarrow aS/Sa/a$

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



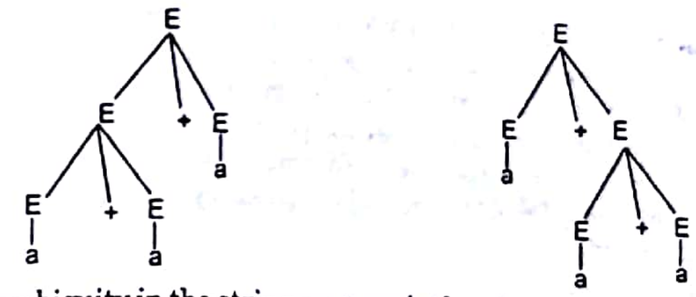
The CFG is therefore ambiguous.

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 asscciative 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

R.E = (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.

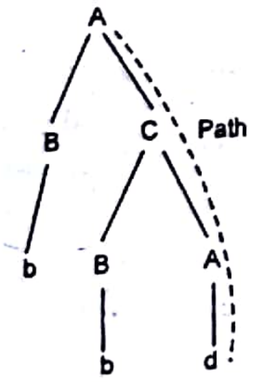
Basis: If $i = 1$

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

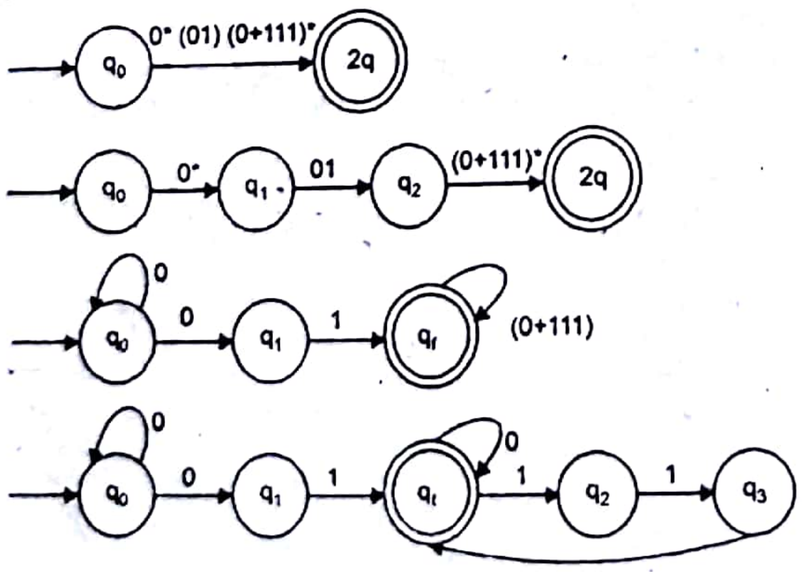
i.e. Path Length $i = 3$

$|w| \leq 2^{i-1}$ *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 ∈ . 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
→ 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
→ $[q_0]$	$[q_0, q_1]$	$[\Phi]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_f]$
(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
→ $[q_0, q_1]$	$[q_0, q_1]$	$[q_f]$
(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 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

- α_0 is the initial ID of M with input w.
- $\alpha_0 \vdash \alpha_1 \vdash \ldots \vdash \alpha_k$, $k \le P(n)$
- α_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 α_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 α_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 ith 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:

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

(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 α_i is none of $x_{i,j-1}$, x_{ij}, $x_{i,j+1}$, then the transition corresponding to α_{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}	$x_{i,j+1}$
$x_{j+1,j-1}$	$x_{i+1,j}$	$x_{i+1,j+1}$

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 $0(p^2(n))$ 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:-

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

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

$$(q_o, abab, z_o) \vdash (q_1, bab, az_o)$$
$$\vdash (q_2, ab, z_o)$$
$$\vdash (q_3, b, az_o)$$
$$\vdash (q_2, \epsilon, z_o)$$
$$\vdash (q_f, \epsilon)$$

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 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):
    k = =0
if
    s = =t
return
    k = = 1
if
    s = = t or (s, t)
return
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 clas containing the language that are complete for NL, the class of decision problems tha can be solved by non-determinstic Turing machine using a logarithmic amount of memor

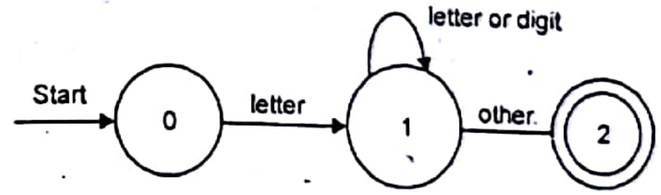
space. The NL-complete language are the most "difficult" or "expressive" probelms in NL. If a method exist for solving any one of the NL- complete problems in logarithmic memory space, then NL = L

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

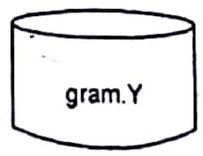
Output: A parser file. tab. c (bison) or y. tab. c (yacc).

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

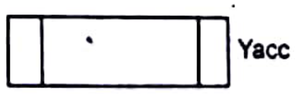
(2) A file file. tab. h containing declaration.

(3) The parser called yy parse ().

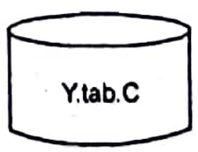
(4) Parser expects to use a function called yylex () to get token.



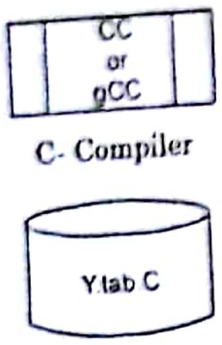
file containing desired grammer in YACC format



YACC program



C source program created by YACC



C- Compiler

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 $S \to \wedge$ i 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 an 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 proof 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 $a \in \Sigma$ (α 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 Σ, only if $L \subseteq \Sigma^*$ this is because Σ^* is the set of all strings (of all possible length including 0) over the given alphabet Σ.

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 Σ 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. –

$(\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to a, B \to b\})$

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

a and b are Terminal symbols.

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

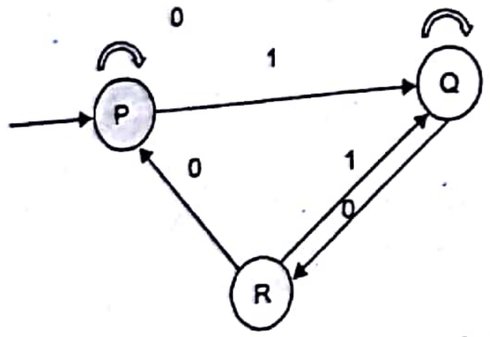
Q.2.(a)Differentiate between Deterministic Finite Automata and Non-Deterministic Finite Automata.

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:

$P = \epsilon + P.0 + R.0$ ----------(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

$P = \epsilon + P.0 + Q.0.0$ ----------(5)

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

$P = \epsilon + P.0 + 0.0.P.1(1 + 0.1)^*$

$P = \epsilon + P((0 + 0.0.1(1 + 0.1)^*)$

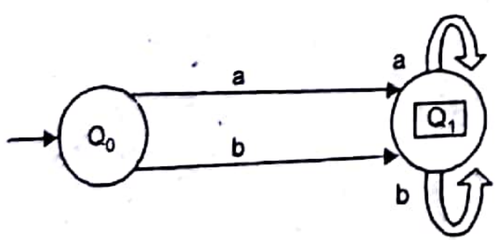
Apply the theorem result ($Q + RP=QP^*$)

$P = \epsilon((0 + 0.0.1(1 + 0.1)^*)^*$

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

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 $(Q, \Sigma, O, \delta, X, q_0)$ where–

Q is a finite set of states.

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

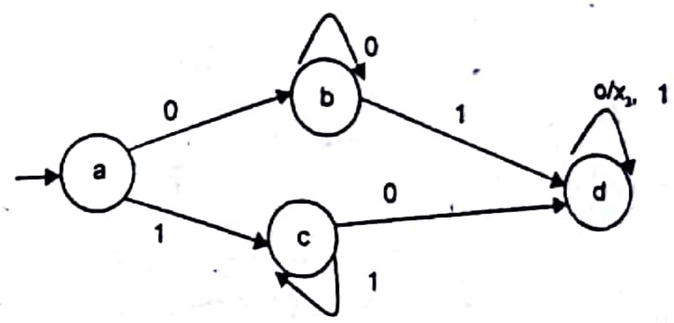
O is a finite set of symbols called the output alphabet.

δ is the input transition function where $\delta: Q \times \Sigma \rightarrow Q$

X is the output transition function where $X: Q \rightarrow O$

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 $(Q, \Sigma, O, \delta, X, q_0)$ 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 $\delta: Q \times \Sigma \rightarrow Q$

X is the output transition function where $X: Q \times \Sigma \rightarrow O$

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.

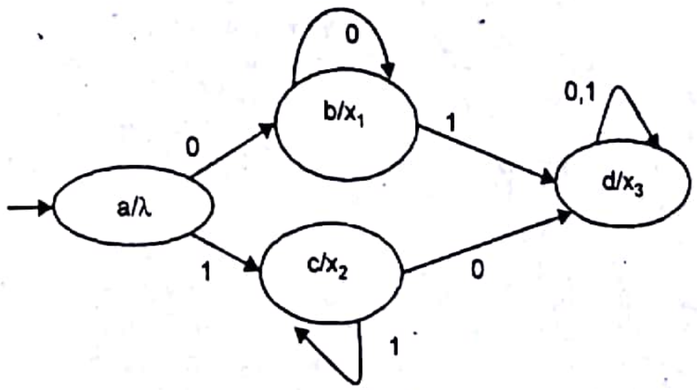


Table format for both machines:

Moore Machine

Present state	Next state		Output
	a = 0	a = 1	
$\rightarrow q_0$	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
$\rightarrow q_0$	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 $L(r) \cup L(s)$.
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.

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

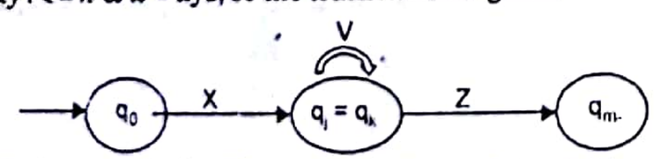
$Q_1 = |q_0 q_1 q_2$_____$q_m|$

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_1must 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.

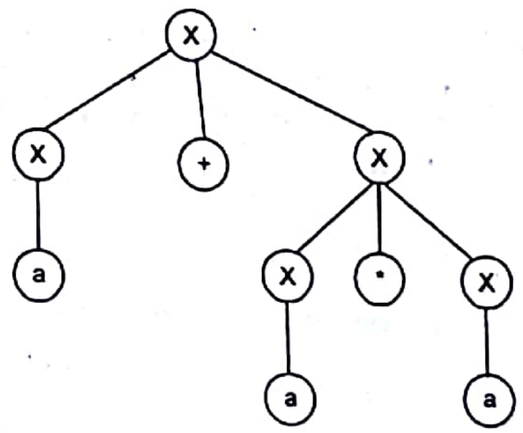
Solution.

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

Derivation 1 –

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

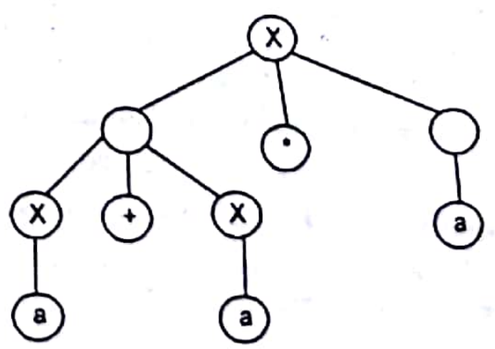
Parse tree 1–



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 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 $L1 \cap L2$, $L1\,L2$, L^*, $L1 \cup L2$ and $L1 - L2$.

2. If L, L 1 and L 2 are recursively enumerable languages, then so are $L1 \cup L2$, $L1\,L2$, L^* and $L1 \cap L2$.

3. If Σ is an alphabet, $L \subseteq \Sigma^*$ is a recursively enumerable language, and $\Sigma^* - L$ is recursively enumerable, then L is recursive.

4. If Σ is an alphabet, $L \subseteq \Sigma^*$ is a recursively enumerable language, and $\Sigma^* - L$ 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., $L \leq TH$).

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.

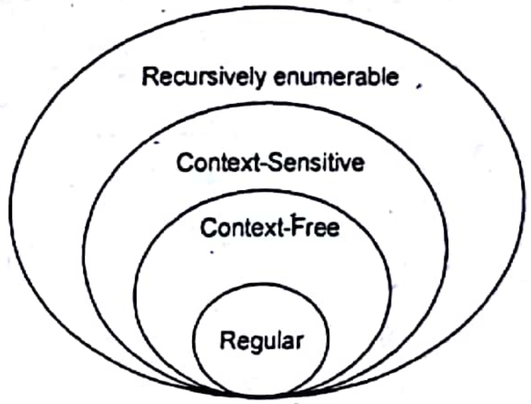
Q.1. (e) Differentiate between Moore and Mealy machine?

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 $X \to a$ 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

$S \to X\,a$

$X \to a$

$X \to aX$

$X \to abc$

$X \to \epsilon$

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 $A \in N$ (Non-terminal) and $\alpha, \beta, \gamma \in (T \cup N)^*$ (Strings of terminals and non-terminals)

The strings α and β may be empty, but γ 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

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

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.

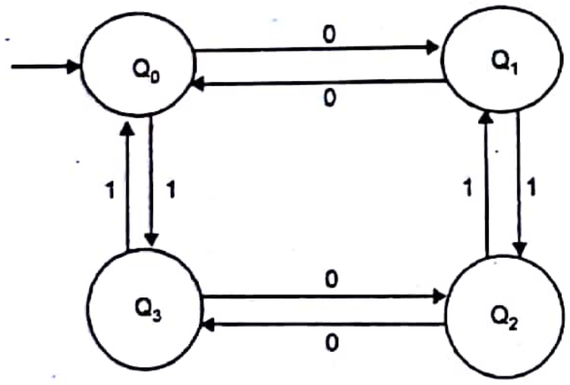
The productions can be in the form of $\alpha \rightarrow \beta$ where α is a string of terminals and non-terminals with at least one non-terminal and α cannot be null. β is a string of terminals and non-terminals.

Example

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

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

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

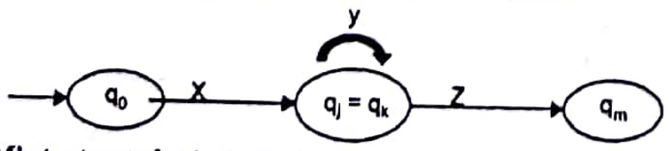
Q_1 = {q_0 q_1 q_2——q_m}

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_1must 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 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 → bB/aA

A → b/bS/aAA

B → a/aS/bBB for the string bbaababa. Find

(*i*) Left Most Derivation

(*ii*) Right Most Derivation

(*iii*) Parse Tree

Ans: Left most derivation:

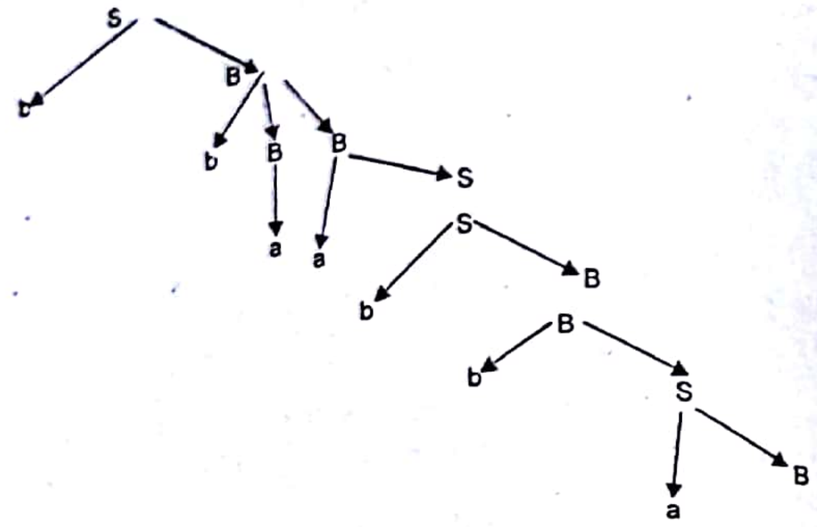
S = bB
= bbBB
= bbaB
= bbaaS
= bbaabB
= bbaabaS
= bbaababB
= bbaababa

Right most derivation:

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

10-2016

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

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

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

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

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

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

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

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

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 $L_2 = \{c^m d^m, n \neq 0\}$. 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 $L = \{a^n b^n, n \geq 0\}$. 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.

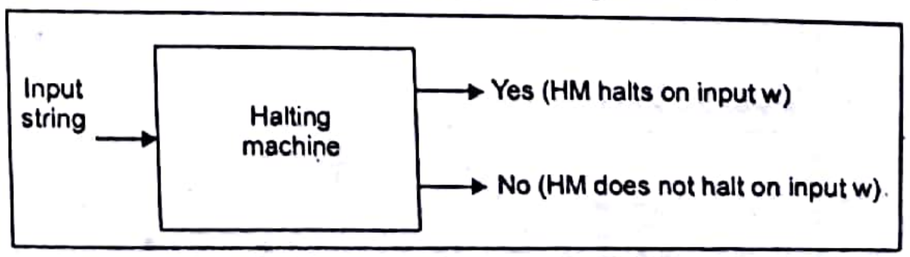
Hence, L is not a context-free language.

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.

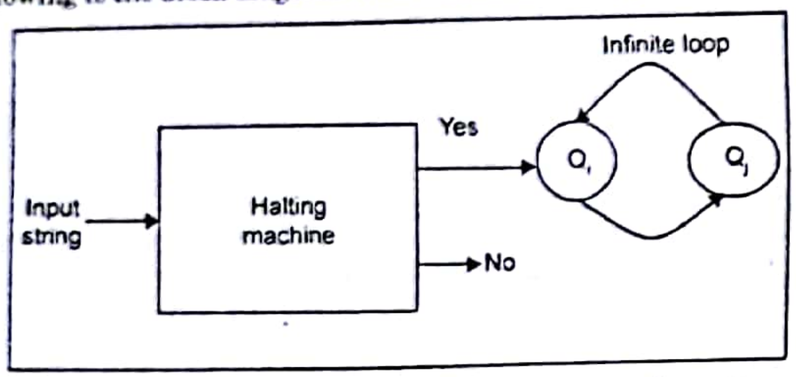
Proof: At first, we will assume that such a Turing machine exists to solve this problem and then we will show it is contradicting itself. We will call this Turing machine as a **Halting machine** that produces a 'yes' or 'no' in a finite amount of time. If the halting machine finishes in a finite amount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

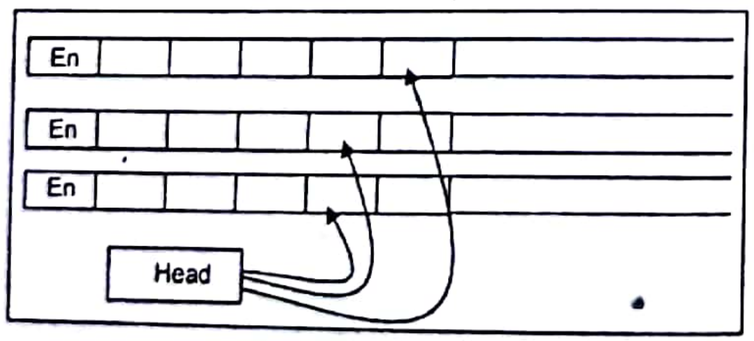
Ans: A Turing Machine (TM) is a mathematical model which consists of an infinite length tape divided into cells on which input is given. It consists of a head which reads the input tape. A state register stores the state of the Turing machine. After reading an input symbol, it is replaced with another 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; $\delta : Q \times X \rightarrow Q \times X \times \{Left\_shift, Right\_shift\}$.
- 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
  $\delta: Q \times X^k \rightarrow Q \times (X \times \{Left\_shift, Right\_shift, No\_shift\})^k$
  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
  $\delta(Q_i, [a_1, a_2, a_3, ...]) = (Q_j, [b_1, b_2, b_3, ...], Left\_shift \text{ or } Right\_shift)$
- 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;
  $\delta : Q \times X' \rightarrow P(Q \times X \times \{Left\_shift, Right\_shift\})$.
- 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 determinstic and non-deterministic space complexity. It states that for any function.

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

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 valuc 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

- α_0 is the initial ID of M with input w.
- $\alpha_0 \vdash \alpha_1 \vdash \ldots \vdash \alpha_k$, $k \le P(n)$
- α_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 α_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 α_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 ith 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:

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

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

(iii) Simulation of Intermediate Moves: We have to simulate valid moves $\alpha_i \vdash \alpha_{j+1}$, 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 α_i is none of $x_{i,j-1}, x_{ij}, x_{i,j+1}$, then the transition corresponding to α_{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 ānd 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 $A \rightarrow B_1C$ 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)

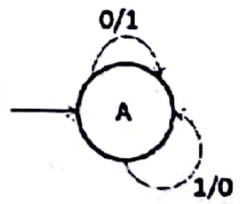
Deterministic Finite Automata	Non-Deterministic Finite Automata
1. Transition function ($\delta : Q \times \Sigma \rightarrow Q$)	Transition relation $\Delta : Q \times \Sigma \rightarrow P(Q)$.
2. DFA cannot use empty string transition	NFA can use empty string transition
3. DFA requires more space	NFA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

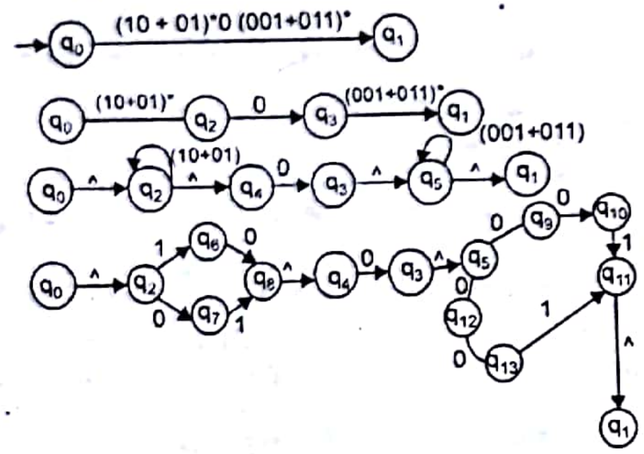
Ans. 1(11)*

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_3
q_1	q_2	q_5
q_2	q_3	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

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

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

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

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

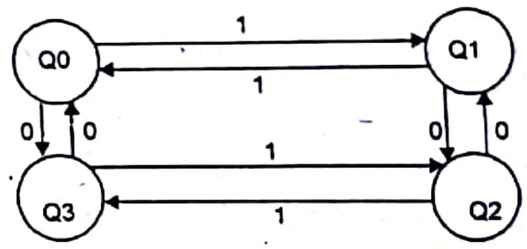
$\prod_5 = (q_0), (q_1), (q_2), (q_3), (q_4), (q_5), (q_6)$

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

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

$= q_1 1 (1 + o11)^*$

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

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 $|xy| \leq m$, 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
$S \rightarrow C_0AD$
$C_0 \rightarrow 0$
A → 0B
$A \rightarrow C_0b$
A → 1AB
$A \rightarrow C_1AB$
$C_1 \rightarrow 1$
$S \rightarrow C_0AD$
$S \rightarrow C_0C_1$
$C_1 \rightarrow AD$
$A \rightarrow C_1AB$
$A \rightarrow C_1C2$
$C_2 \rightarrow AB$

Q.4. (b) Remove the null Production from the given grammar-{ S →aS/AB,A→ ϵ B → ϵ ,D → b} here S,A,B & D are the variables , S is the Start Variable,a,b are the terminls and ϵ 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 ⇒ ^.
A variable A in a context-free grammar is nullable if $A \rightarrow A_1 \rightarrow \wedge$ is a production in G}

$W_1 = \{A_1 \in V_n : A_1 \rightarrow \wedge$ is a production in G}
$= \{A, B\}$
$W_2 = W_1 \cup \{A \in V_N$: there exists a production $A \rightarrow \alpha$ 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 $A \rightarrow X_1 X_2 \ldots X_k$ 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]

M.M. : 75

Time : 3 Hrs.

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 !=ϵ and xy^iz belongs to L for each i>= 0.

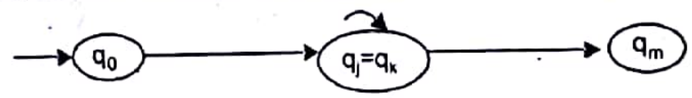
Proof: Find a non-empty string y^i not too far from the beginning of w that can be pumped i.e. repeating y^iany no of times,keeps the resulting string in the language.

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^iit 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 $S' \rightarrow S$.

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

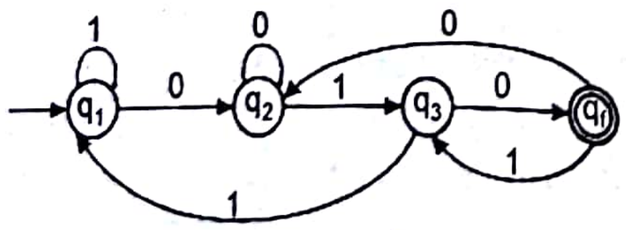
Step 3 – Remove unit productions. (Using the Unit production removal algorithm discussed earlier)

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

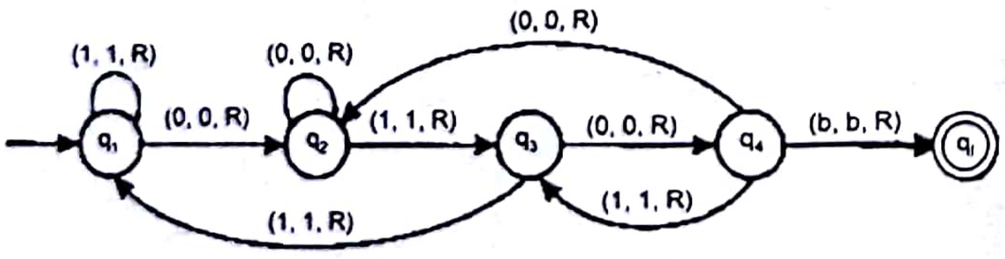
Step 5 – Do proper substitutions of productions to convert it into the proper form of GNF.

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.

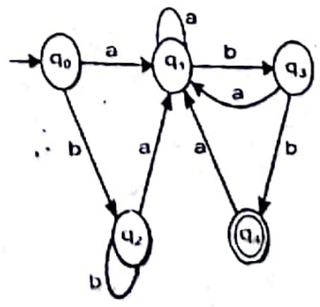
Proof: There is a proof of the theorem that is constructive; is demonstrates an algorithm for STCON, the problem of determining whether there is a path between two vertices in a directed graph, which runs in $0(\log n)^2$).

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
→ 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

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

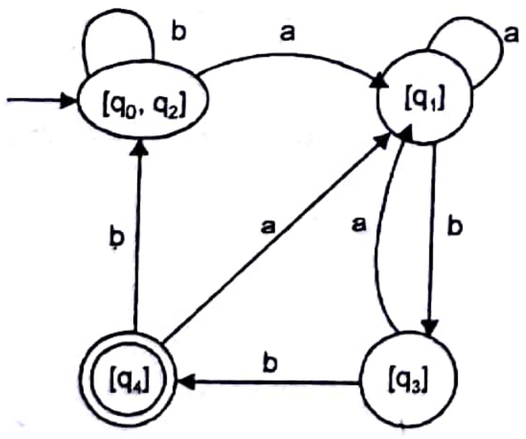
Now, the states q_0 and q_1 are not 1-equivalent because under b-column of q_0 and q_1 we get q_2 and q_3 where q_2 and q_3 do not lie in the same class.

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 π_2. So get the minimum automata as

State/Σ	a	b
→ $[q_0, q_2]$	$[q_1]$	$[q_0, q_2]$
$[q_1]$	$[q_1]$	$[q_3]$
$[q_3]$	$[q_1]$	$[q_4]$
((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) $\text{L.H.S.} = (1 + 00^*1)(\wedge + (0 + 10^*1))^*(0 + 10^*1))$

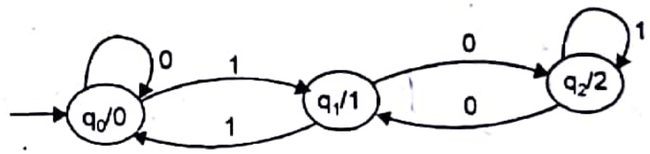
$= (1 + 00^*1)(0 + 10^*1)^*$

$= 1(\wedge + 00^*)(0 + 10^*1)^*$ $[\wedge + R^*R = R^*]$

$= 0^*1(0 + 10^*1)^* = \text{R.H.S.}$

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.

$\lambda'(q_0, 1) = 1$

$\lambda'(q_1, 0) = \lambda(\delta(q_1, 0))$

$= \lambda(q_2)$

$\lambda'(q_1, 0) = 2.$

$\lambda'(q_1, 1) = \lambda(\delta(q_1, 1))$

$= \lambda(q_0)$

$\lambda'(q_1, 1) = 0$

$\lambda'(q_2, 0) = \lambda(\delta(q_2, 0))$

$= \lambda(q_1)$

$\lambda'(q_2, 0) = 1$

$\lambda'(q_2, 1) = \lambda(\delta(q_2, 1))$

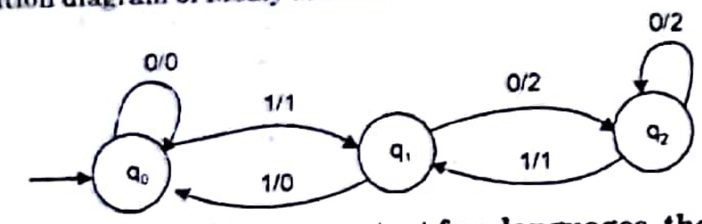
$= \lambda(q_2)$

$\lambda'(q_2, 1) = 2.$

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: $G = (V1 \cup V2 \cup \{S\}, T1 \cup T2, P1 \cup P2 \cup \{S \rightarrow S1 \mid S2\}, S)$.

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: $S \rightarrow aSa \mid bSb \mid a \mid b \mid \wedge$

• Let L_2 be $\{a^n b^n \mid n \geq 0\}$ defined by: $S \rightarrow aSb \mid \wedge$

• Then the union language is defined by: $S \rightarrow S1 \mid S2$ $S1 \rightarrow aS1a \mid bS1b \mid a \mid b \mid \wedge$ $S2 \rightarrow aS2b \mid \wedge$

Q.4. (a) Convert the following grammar G to Chomsky Normal Form: $S \rightarrow ABCa$ $A \rightarrow aAbb, A \rightarrow e, B \rightarrow bB, B \rightarrow b, B \rightarrow AC, C \rightarrow aCa, C \rightarrow e$ (6)

Ans. $A \rightarrow e, B \rightarrow b, C \rightarrow e$ are in CNF

$S \rightarrow ABCa, F \rightarrow b$ therefore $B \rightarrow FB$

Now we have to convert $S \rightarrow ABCa$

$A \rightarrow aAbb$, $C \rightarrow aCa$ now $M \rightarrow a$

Therefore

$S \rightarrow ABCM$

$A \rightarrow MAFF$

$C \rightarrow MCM$

$G \rightarrow CM$

Therefore

$S \rightarrow ABG$,$H \rightarrow BG$,$S \rightarrow AH$

$C \rightarrow MG$

$L \rightarrow FF$

So $A \rightarrow MAL$,$V \rightarrow AL$,$A \rightarrow MV$ 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 $A \rightarrow BC$ in P, for every $p, q, r \in Q$ we have the production $[p, A, r] \rightarrow [p, B, q][q, C, r]$ 2 in P'.

3. For every production $A \rightarrow \sigma$ in P, for every $(p, \sigma, q) \in Q \times \Sigma \times Q$ such that $\delta(p, \sigma) = q$ we have the production $[p, A, q] \rightarrow \sigma$ 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.

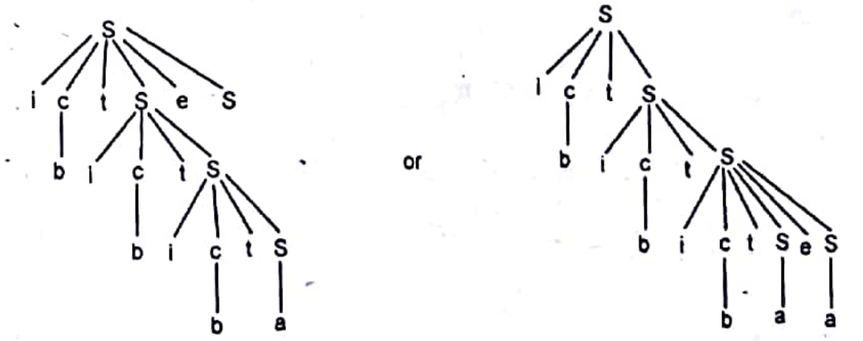
Proof, Let L_1 and L_2 are two CFLs. We will assume that complement of a context-free language is a CFL itself. Hence L'_1 and L'_2 both are CFLs. We can also state that $(L'_1 \cup L'_2)$ is context free (since CFLs are closed under union). But $(L'_1 \cup L'_2) = [L'_1 \cup L_2$'i.e., $L = L_1 \cap L_2$ 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.



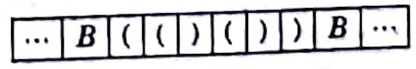
Thus, we have got more than two parse trees. Hence the given grammer is ambiguous

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.



δ 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. $\delta(q_1, X) = Xq_1 L$
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)**

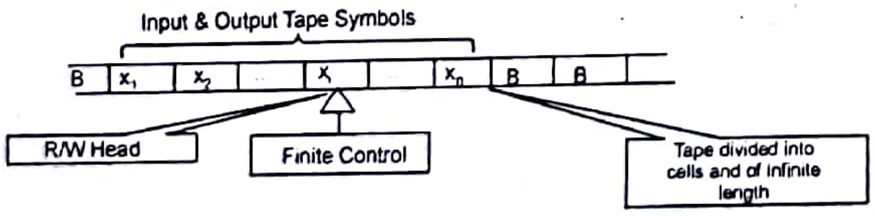
Ans. A Turing Machine (TM) is a mathematical model which consists of an infinite length tape divided into cells on which input is given. It consists of a head which reads the input tape. A state register stores the state of the Turing machine. After reading an input symbol, it is replaced with another 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
- Σ is the input alphabet
- δ 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.

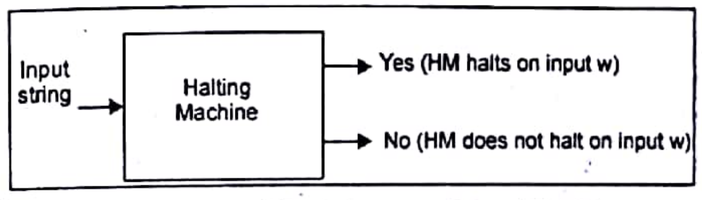
These differences are unimportant.

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.

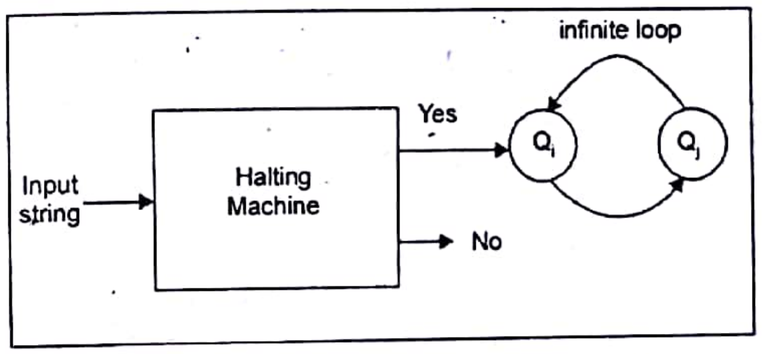
Proof - At first, we will assume that such a Turing machine exists to solve this problem and then we will show it is contradicting itself. We will call this Turing machine as a **Halting machine** that produces a 'yes' or 'no' in a finite amount of time. If the halting machine finishes in a finite amount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

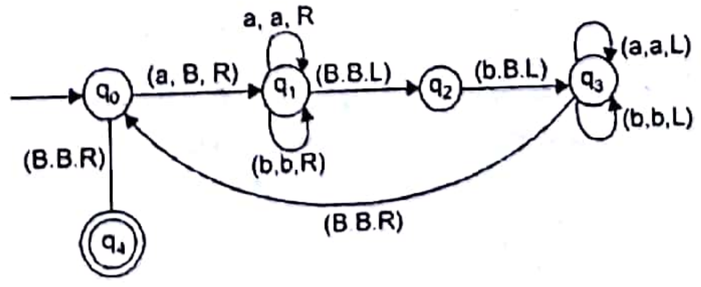
Here, we have got a contradiction. Hence, the halting problem is undecidable.

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 $\vdash$ Bq_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

$\vdash$ Baabbaq_1B $\vdash$ Baabbq_2bB

$\vdash$ Baabq_3bBB $\vdash$ Baa$q_3$bbBB $\vdash$ Ba$q_3$abbBB

$\vdash$ B$q_3$aabbBB $\vdash$ q_3BaabbBB

$\vdash$ B$q_0$aabbBB $\vdash$ BB$q_1$abbBB $\vdash$ BBa$q_1$bbBB

$\vdash$ BBabq_1BB $\vdash$ BBabbq_1BB $\vdash$ BBabq_2bBB

$\vdash$ BBaq_3bBBB $\vdash$ BB$q_3$abBBB $\vdash$ Bq_3BabBBB

$\vdash$ BB$q_0$abBBB $\vdash$ BBBq_1bBBB $\vdash$ BBBbq_1 BBB

$\vdash$ BBBq_2bBBB $\vdash$ BBq_3BBBBB $\vdash$ BBBq_0BBBB $\vdash$BBBBq_4BBB

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: $\Sigma^* \rightarrow \Sigma$ Σ^* 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

- α_0 is the initial ID of M with input w.
- $\alpha_0 \vdash \alpha_1 \vdash ....\vdash \alpha_k$, $k \leq P(n)$
- α_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 α_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 α_m is an accepting ID in the course of processing w then we write $\alpha_0 \vdash \therefore \vdash \alpha_a \vdash a_m = \alpha_{i, p(n)}$.

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

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

(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 α_i is none of $x_{i,j-1}$, x_{ij}, $x_{i,j+1}$, then the transition corresponding to α_{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

$N_i = (A_{i0} \vee B_{i0}) \wedge (A_{i1} \vee B_{i1}) \wedge \ldots \wedge (A_i, p(n) \vee Bi, p(n))$

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

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.