Roll No: | | | | | | | | | | | | | |

BTECH
(SEM IV) THEORY EXAMINATION 2021-22
THEORY OF AUTOMATA AND FORMAL LANGUAGES

Time: 3 Hours ***Total Marks: 100***

Note: Attempt all Sections. If you require any missing data, then choose suitably.

SECTION A

1. Attempt *all* questions in brief. 2x10 = 20

Q.no	Questions	Marks	CO
(a)	Define Alphabet and String in Automata Theory. Solution: An alphabet is any finite set of symbols. Example − ∑ = {a, b, c, d} is an alphabet set where 'a', 'b', 'c', and'd' are symbols. Theory of computation is entirely based on symbols. These symbols are generally letters and digits. Alphabets are defined as a finite set of symbols. Examples: ∑ = {0, 1} is an alphabet of binary digits. A string over an alphabet is a finite sequence of letters from the alphabet. Examples. toc, money, c, and adedwxq are strings over the alphabet ∑ = {a, b, c, . . . , z}. 84029 is a string over the alphabet ∑ = {0, 1, 2, . . . , 9}. A string is a finite ordered sequence of symbols chosen form some set of alphabets or ∑. For example, 'aababbbbaa' is a valid string from the alphabet ∑ = {a, b}, similarly '001111000101' is a valid string form the alphabet ∑ = {0, 1}.	2	2
(b)	Give the definition of Deterministic Finite Automaton (DFA). Solution: Deterministic finite automata (or DFA) are finite state machines that accept or reject strings of characters by parsing them through a sequence that is uniquely determined by each string. The term "deterministic" refers to the fact that each string, and thus each state sequence, is unique. DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. The finite automata are called deterministic finite automata if the machine is read an input string one symbol at a time. DFA consists of 5 tuples {Q, Σ, q, F, δ}. Where, Q: Finite set of states. Σ: set of Input Symbols. q: Initial state. F: set of Final States. δ: Transition Function.	2	1
(c)	Explain in brief about the Kleen's Theorem. Solution: For any Regular Expression *r* that represents Language L(r),	2	2

	there is a Finite Automata that accepts same language. Kleene's theorem: The set of regular languages, the set of NFA-recognizable languages, and the set of DFA-recognizable languages are all the same. It states that any regular language is accepted by an FA and conversely that any language accepted by an FA is regular.		
(d)	Define Context Free Grammar (CFG). Solution: A context free grammar (CFG) is a forma grammar which is used to generate all the possible patterns of strings in a given formal language. CFG stands for context-free grammar. It is is a formal grammar which is used to generate all possible patterns of strings in a given formal language. Context-free grammar G can be defined by four tuples as: G = (V, T, P, S) Where, G is the grammar, which consists of a set of the production rule. It is used to generate the string of a language. T is the final set of a terminal symbol. It is denoted by lower case letters. V is the final set of a non-terminal symbol. It is denoted by capital letters. P is a set of production rules, which is used for replacing non-terminals symbols (on the left side of the production) in a string with other terminal or non-terminal symbols (on the right side of the production). **S** is the start symbol which is used to derive the string. We can derive the string by repeatedly replacing a non-terminal by the right-hand side of the production until all non-terminal have been replaced by terminal symbols.	2	1
(e)	Write the Context Free Grammar (CFG) for regular expression (0+1)* Solution: The CFG can be given by, Production rule (P): $S \rightarrow 0S \mid 1S$ $S \rightarrow \varepsilon$ The rules are in the combination of 0's and 1's with the start symbol. Since (0+1) * indicates {ε, 0, 1, 01, 10, 00, 11, ...}. In this set, ε is a string, so in the rule, we can set the rule $S \rightarrow \varepsilon$.	2	3
(f)	What is Right Linear grammar and Left Linear grammars? Solution: Right Linear Regular Grammar: In this type of regular grammar, all the non-terminals on the right-hand side exist at the rightmost place, i.e.; right ends. Examples: $A \dashrightarrow a$, $A \dashrightarrow aB$, $A \dashrightarrow \in$ where, A and B are non-terminals, a is terminal, and	2	3

	∈ is empty string S ⇢ 00B | 11S B ⇢ 0B | 1B | 0 | 1 where, S and B are non-terminals, and 0 and 1 are terminals Left Linear Regular Grammar: In this type of regular grammar, all the non-terminals on the right-hand side exist at the leftmost place, i.e; left ends. Examples: A ⇢ a, A ⇢ Ba, A ⇢ ∈ where, A and B are non-terminals, a is terminal, and ∈ is empty string S ⇢ B00 | S11 B ⇢ B0 | B1 | 0 | 1 where S and B are non-terminals, and 0 and 1 are terminals		
(g)	Discuss briefly about the Push Down Automata (PDA). Solution: Pushdown Automata is a finite automaton with extra memory called stack which helps Pushdown automata to recognize Context Free Languages. A Pushdown Automata (PDA) can be defined as: -Q is the set of states -∑ is the set of input symbols -Γ is the set of pushdown symbols (which can be pushed and popped from stack) -q0 is the initial state -Z is the initial pushdown symbol (which is initially present in stack) -F is the set of final states -δ is a transition function which maps Q x {Σ ∪ ∈} x Γ into Q x Γ*. In a given state, PDA will read input symbol and stack symbol (top of the stack) and move to a new state and change the symbol of stack.	2	4
(h)	What do you mean by Two stack Pushdown Automata? Solution: Two stacks push down automata (PDA) includes the following factors − -A Turing machine can accept languages that are not accepted by any PDA with one stack. -The strength of pushdown automata is increased by adding extra stacks. -A PDA with two stacks has the same computation power as for a Turing Machine. Two-Stack PDA is a computational model which is based on the generalization of Pushdown Automata (PDA) and Non-deterministic Two-Stack PDA which is equivalent to a deterministic Two-Stack PDA. The moves of the Two-Stack PDA are based on the following − -The state of finite control. -The input symbol that reads. -The top of the stack symbol on each of its stacks.	2	4

(i)	What do you mean by basic Turing Machine Model? Solution: 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. Turing Machine was invented by Alan Turing in 1936 and it is used to accept Recursive Enumerable Languages (generated by Type-0 Grammar). A Turing machine consists of a tape of infinite length on which read and writes operation can be performed. The tape consists of infinite cells on which each cell either contains input symbol or a special symbol called blank. It also consists of a head pointer which points to cell currently being read and it can move in both directions. A TM is expressed as a 7-tuple (Q, T, B, ∑, δ, q0, F) where: -Q is a finite set of states -T is the tape alphabet (symbols which can be written on Tape) -B is blank symbol (every cell is filled with B except input alphabet initially) -∑ is the input alphabet (symbols which are part of input alphabet) -δ is a transition function which maps $Q \times T \rightarrow Q \times T \times \{L, R\}$. Depending on its present state and present tape alphabet (pointed by head pointer), it will move to new state, change the tape symbol (may or may not) and move head pointer to either left or right. -q0 is the initial state -F is the set of final states. If any state of F is reached, input string is accepted.	2	5
(j)	What do you understand by the Halting Problem? Solution: The halting problem is a decision problem about properties of computer programs on a fixed Turing-complete model of computation, i.e., all programs that can be written in some given programming language that is general enough to be equivalent to a Turing machine. unsolvable algorithmic problem is the halting problem, which states that no program can be written that can predict whether or not any other program halts after a finite number of steps. The unsolvability of the halting problem has immediate practical bearing on software development. Basically, halting means terminating. So, can we have an algorithm that will tell that the given program will halt or not. In terms of Turing machine, will it terminate when run on some machine with some particular given input string.	2	5

SECTION B

2. Attempt any *three* of the following: 10x3 = 30

Q.no	Questions	Marks	CO
(a)	Explain in detail about the Turing Church's Thesis and Recursively Enumerable languages. Solution: The Church-Turing thesis explains that a decision problem Q has a solution if and only if there is a Turing machine that determines the answer for every q ϵ Q. If no such Turing machine exists, the	10	5

	problem is said to be undecidable. The Church-Turing thesis (formerly commonly known simply as Church's thesis) says that any real-world computation can be translated into an equivalent computation involving a Turing machine. The fact there is a Church-Turing thesis is also the reason that mathematics and theory in general exists. While mathematicians love to introduce new symbols as they go, they can do so because they're all defined in terms of other simpler symbols down to a very small Turing-complete set. As Turing claimed, any process that can be naturally called an effective procedure is realized by a Turing machine. This is known as Turing's thesis. Namely, if someone built a device which (reliably) computed a function that cannot be computed by any Turing machine, that would disprove the Church-Turing thesis because it would establish existence of an effectively calculable function that is not computable by a Turing machine. The extended Church-Turing thesis is a foundational principle in computer science. It asserts that any" rea- sonable" model of computation can be efficiently simulated on a standard model such as a Turing Machine or a Random-Access Machine or a cellular automaton.		
(b)	Prove that the Compliment, Homomorphism, Inverse Homomorphism, and Closure of a Regular Language is also Regular. Solution: A regular language satisfies the following equivalent properties: it is the language of a regular expression (by the above definition) it is the language accepted by a nondeterministic finite automaton (NFA) it is the language accepted by a deterministic finite automaton (DFA) Regular Languages are closed under complementation, i.e., if L is regular then L = Σ∗ \ L is also regular. Proof. If L is regular, then there is a DFA M = (Q, Σ, δ, q0, F) such that L = L(M). Theorem If L is a regular language, its complement L is also regular. (i.e., switch accept and non-accept states) accepts L. The set of regular languages is closed under complementation. The complement of language L, written L, is all strings not in L but with the same alphabet. Closure properties on regular languages are defined as certain operations on regular language which are guaranteed to produce regular language. Closure refers to some operation on a language, resulting in a new language that is of same "type" as originally operated on i.e., regular. Regular languages are closed under inverse homomorphism, i.e., if L is regular and h is a homomorphism then h−1(L) is regular. A homomorphism on an alphabet is a function that gives a string for each symbol in that alphabet. Example: h(0) = ab; h(1) = . Extend to strings by h(a1…an) =h(a1)… h(an).	10	2
(c)	Give the Complete description about the Chomsky Hierarchy.	10	3

Solution: The Chomsky hierarchy is important in cognitive science

because the complexity of a grammar in the hierarchy can be used to evaluate (at the computational level) theoretical proposals within cognitive science.

Chomsky Hierarchy represents the class of languages that are accepted by the different machine. The category of language in Chomsky's

Hierarchy is as given below:

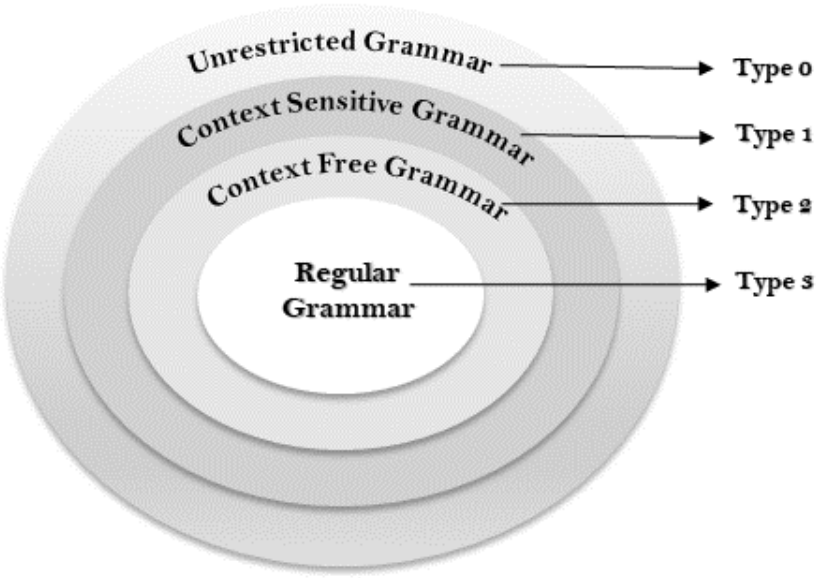


Fig: Chomsky Hierarchy

Type 0 known as Unrestricted Grammar.

Type 1 known as Context Sensitive Grammar.

Type 2 known as Context Free Grammar.

Type 3 Regular Grammar.

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

Type 0 Grammar:

Type 0 grammar is known as Unrestricted grammar. There is no restriction on the grammar rules of these types of languages. These languages can be efficiently modeled by Turing machines.

Example:

$bAa \rightarrow aa$

$S \rightarrow s$

Type 1 Grammar:

Type 1 grammar is known as Context Sensitive Grammar. The context

sensitive grammar is used to represent context sensitive language. The context sensitive grammar follows the following rules:

The context sensitive grammar may have more than one symbol on the left-hand side of their production rules.

The number of symbols on the left-hand side must not exceed the number of symbols on the right-hand side.

The rule of the form $A \rightarrow \varepsilon$ is not allowed unless A is a start symbol. It does not occur on the right-hand side of any rule.

The Type 1 grammar should be Type 0. In type 1, Production is in the form of $V \rightarrow T$

Where the count of symbol in V is less than or equal to T.

Example:

$S \rightarrow AT$

$T \rightarrow xy$

$A \rightarrow a$

Type 2 Grammar:

Type 2 Grammar is known as Context Free Grammar. Context free languages are the languages which can be represented by the context free grammar (CFG). Type 2 should be type 1. The production rule is of the form

$A \rightarrow \alpha$

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

Example:

$A \rightarrow aBb$

$A \rightarrow b$

$B \rightarrow a$

Type 3 Grammar:

Type 3 Grammar is known as Regular Grammar. Regular languages are those languages which can be described using regular expressions. These languages can be modeled by NFA or DFA.

Type 3 is most restricted form of grammar. The Type 3 grammar should be Type 2 and Type 1. Type 3 should be in the form of

$V \rightarrow T^*V / T^*$

Example: $A \rightarrow xy$

(d)	Convert the grammar S → aAA, A → a |aS| bS to a PDA that accepts the same language by Empty stack. Solution: The PDA P = (Q, Σ, Γ, δ, q0, Z0, F) is defined as Q = {q} Σ = {a, b} Γ = {a, b, S, A} q0 = q Z0 = S F = {} And the transition function is defined as: δ(q, ε , S) = {(q, aAA)} δ(q, ε , I) = {(q, aS),(q, bS),(q, a)} δ(q, a, a) = {(q, ε)} δ(q, b, b) = {(q, ε)} Here a PDA accepts a string when, after reading the entire string, the PDA has emptied its stack. For a PDA (Q, ∑, S, δ, q0, Z0, F), the language accepted by the empty stack is − L(PDA) = {w | (q0, w, I) ⊢* (q, ε, ε), q ∈ Q}	10	4
(e)	Grammar G is given with the production S->aSS A->b. Compute the string w= aababbb with the Left most and Right most derivation Tree. Solution:	10	1

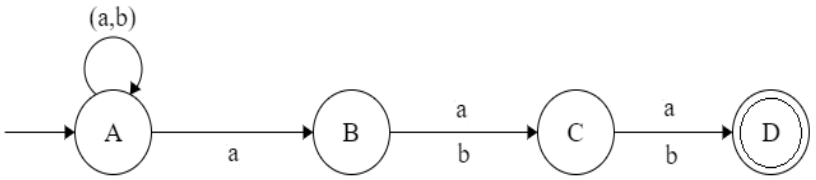
SECTION C

3. Attempt any *one* part of the following: 10x1 = 10

Q.no	Questions	Marks	CO
(a)	Write short notes on following. i) Turing Machine as Computer of Integer Functions ii) Universal Turing machine Solution: A basic Turing machine is a model for studying computation. Turing machines can solve decision problems and compute results based on inputs. When studying computation, we usually restrict our attention to integers. Since a real number has infinitely many fraction digits we cannot compute a real number in a finite time. Rational numbers a approximations to real numbers are equivalent and can be put in one-to-one correspondence with the integers. Programming a Turing machine is tedious and thus much work at higher levels of abstraction make the reasonable assumption that any completely defined algorithm or computer program could be implemented by a Turing machine. Turing Machine as Computer of Integer Functions: Turing machines can compute any function normally considered computable; in fact, it is quite reasonable to define computable to mean computable by a TM. In the analogy with a computer, the "tape" of the Turing machine is the computer memory, idealized to extend infinitely in each direction. The initial arrangement of colors of cells on the tape corresponds to the input	10	5

	given to the computer. This input can contain both a "program" and "data". A Turing machine is a mathematical model of computation describing an abstract machine that manipulates symbols on a strip of tape according to a table of rules. Despite the model's simplicity, it is capable of implementing any computer algorithm. If you look at computational complexity, a Turing Machine is the most powerful machine - because it has unlimited memory, and no real machine has that. Any real machine cannot solve problems of arbitrary size; they cannot even read a problem, much less solve it.		
(b)	Explain in detail about the Pumping Lemma and application of Pumping Lemma for Regular Languages. Solution: 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. If L is regular, it satisfies Pumping Lemma. If L does not satisfy Pumping Lemma, it is non-regular. In simple terms, this means that if a string v is 'pumped', i.e., if v is inserted any number of times, the resultant string still remains in L. Pumping Lemma is used as a proof for irregularity of a language. Thus, if a language is regular, it always satisfies pumping lemma. The pumping lemma is often used to prove that a particular language is non-regular. Pumping lemma for regular language is generally used for proving a given grammar is not regular. Hence the correct answer is a given grammar is not regular. Pumping Lemma is to be applied to show that certain languages are not regular. 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. In the theory of formal languages, the pumping lemma may refer to: Pumping lemma for regular languages, the fact that all sufficiently long strings in such a language have a substring that can be repeated arbitrarily many times, usually used to prove that certain languages are not regular. There are two Pumping Lemmas, which are defined for 1. Regular Languages, and 2. Context – Free Languages **Pumping Lemma for Regular Languages** For any regular language L, there exists an integer n, such that for all x $\in$ L with $\|x\| \geq n$, there exists u, v, w $\in \Sigma*$, such that x = uvw, and (1) $\|uv\| \leq n$ (2) $\|v\| \geq 1$ (3) for all $i \geq 0$: $uv^iw \in L$ In simple terms, this means that if a string v is 'pumped', i.e., if v is inserted any number of times, the resultant string still remains in L. Pumping Lemma is used as a proof for irregularity of a language. Thus, if a language is regular, it always satisfies pumping lemma. If there exists at least one string made from pumping which is not in L, then L is surely not regular. The opposite of this may not always be true. That is, if Pumping Lemma holds, it does not mean that the language is regular.	10	2

4. Attempt any *one* part of the following: 10x1 = 10

Q.no	Questions	Marks	CO
(a)	Construct a Non-Deterministic Finite Automation (NFA) for the language L which accepts all the strings in which the third symbol From right end is always 'a' over Σ = {a, b}. Solution: We have to draw a Non-deterministic finite automata (NFA) of the language containing the set of all strings over {a, b} in which 3rd symbol from RHS is 'a'. The strings in which 3rd last symbol is "a" are: aaa, aab, aaab, aaaa, aabbaaa, bbbaba etc The regular expression: L = (a+b)*.a.(a+b).(a+b) The NFA of the language containing all the strings in which 3rd symbol from the RHS is "a" is shown as below.  NFA	10	1
(b)	Explain in detail about the Myhill-Nerode theorem using suitable example. Solution: Myhill Nerode Theorem: A language is regular if and only if ≡L partitions ∑* into finitely many equivalence classes. If ≡L partitions ∑* into n equivalence classes, then a minimal DFA recognizing L has exactly n states. Example: To prove that L = $\{a^n b^n \mid n \geq 0\}$ is not regular. We can show that L has infinitely many equivalence classes by showing that a^k and a^i are distinguishable by L whenever $k \neq i$. Thus, for $x = a^k$ and $y = a^i$ we let $z = b^k$. Then $xz = a^k b^k$ is in the language but $yz = a^i b^k$ is not. Thus, each equivalence class of L can contain at most one string of the form ai so there must be infinitely many equivalence classes. That means L is not regular by the Myhill Nerode theorem. Note: To prove whether or not a language L is regular is also done using Pumping Lemma, the distinction between this and Myhill Nerode theorem is that, there are some non-regular language satisfying the Pumping Lemma but no such non-regular language is there which satisfies Myhill Nerode theorem.	10	3

Roll No: | | | | | | | | | | | | | |

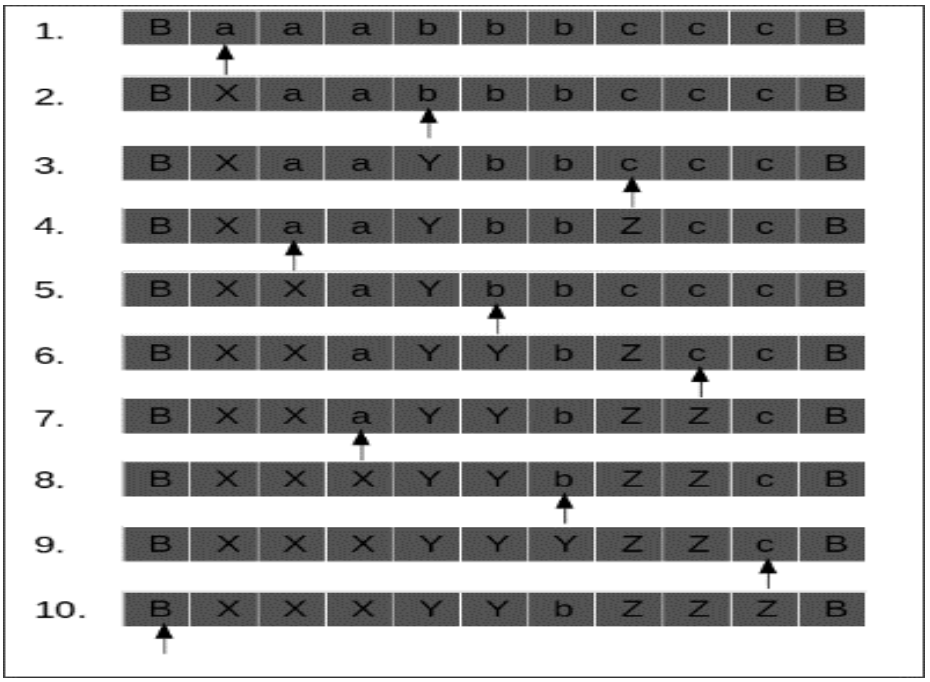
BTECH
(SEM IV) THEORY EXAMINATION 2021-22
THEORY OF AUTOMATA AND FORMAL LANGUAGES

5. Attempt any *one* part of the following: 10x1 = 10

Q.no	Questions	Marks	CO
(a)	Prove that the following Language L = {$a^n b^n$: n>=0} is not a regular language. Solution: **$\{a^n b^n\}$ Is Not Regular** 1 Proof is by contradiction using the pumping lemma for regular languages. Assume that $L = \{a^n b^n\}$ is regular, so the pumping lemma holds for L. Let k be as given by the pumping lemma. 2 Choose x, y, and z as follows: $x = a^k$ $y = b^k$ $z = \varepsilon$ Now $xyz = a^k b^k \in L$ and $\|y\| \geq k$ as required. 3 Let u, v, and w be as given by the pumping lemma, so that $uvw = y$, $\|v\| > 0$, and for all $i \geq 0$, $xuv^i wz \in L$. 4 Choose $i = 2$. Since v contains at least one b and nothing but bs, uv^2w has more bs than uvw. So xuv^2wz has more bs than as, and so $xuv^2wz \notin L$. 5 By contradiction, $L = \{a^n b^n\}$ is not regular. **The Pattern** 1 Proof is by contradiction using the pumping lemma for regular languages. Assume that $L = \{a^n b^n\}$ is regular, so the pumping lemma holds for L. Let k be as given by the pumping lemma. 2 Here, you choose xyz and show that they meet the requirements, $xyz \in L$ and $\|y\| \geq k$. Choose them so that pumping in the y part will lead to a contradiction, a string $\notin L$. 3 Let u, v, and w be as given by the pumping lemma, so that $uvw = y$, $\|v\| > 0$, and for all $i \geq 0$, $xuv^i wz \in L$. 4 Here, you choose i, the number of times to pump, and show that you have a contradiction: $xuv^i wz \notin L$. 5 By contradiction, $L = \{a^n b^n\}$ is not regular.	10	4
(b)	Design a Turing Machine for the language L. Where, L={$a^n b^n c^n$| n≥1} Solution: Approach for $a^n b^n c^n \mid n \geq 1$ 1. Mark 'a' then move right.	10	5

2. Mark 'b' then move right
3. Mark 'c' then move left
4. Come to far left till we get 'X'
5. Repeat above steps till all 'a', 'b' and 'c' are marked
6. At last if everything is marked that means string is accepted.

TAPE movement for string "aaabbbccc":



Explanation of TAPE movement:

Step 1-2

1. Input is given as "aaabbbccc" (scan string from left to right)
2. Mark 'a' as 'X' and move one step right
3. Reach unmarked 'b' and pass every 'a' and 'Y'(in case) on the way to 'b'

Step 3-4

4. Mark 'b' as 'Y' and move one step right
5. Reach unmarked 'c' and pass every 'b' and 'Z'(in case) on the way to 'c'
6. Mark 'c' as 'Z' and move one step left cause now we have to repeat process
7. Reach unmarked 'X' and pass every 'Z', 'b', 'Y', 'a' on the way to 'X'
8. Move to 'a' and repeat the process

Step 5-9

9. Step 1-4 are repeated

Step 10

10. When TAPE header reaches 'X' and next symbol is 'Y' that means

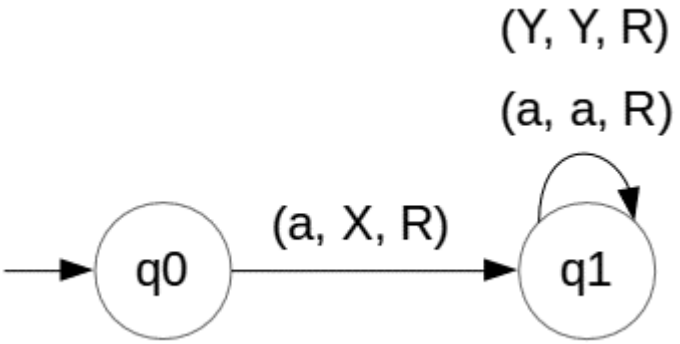
'a's are finished

11. Check for 'b's and 'c's by going till BLANK(in right) and passing 'Y' and 'Z' on the way
12. If no 'b' and 'c' are not found that means string is accepted

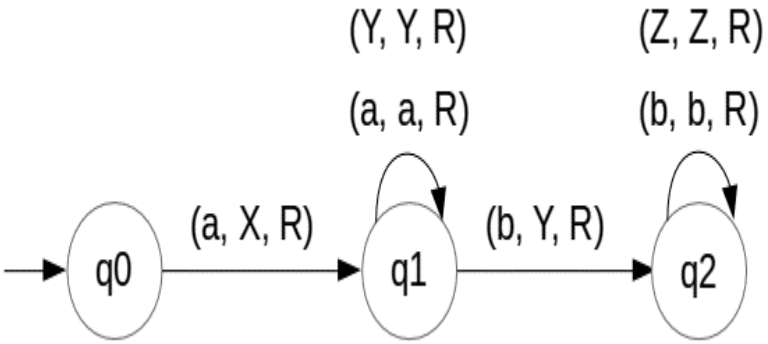
State Transition Diagram

We have designed state transition diagram for $a^n b^n c^n \mid n \geq 1$ as follows:

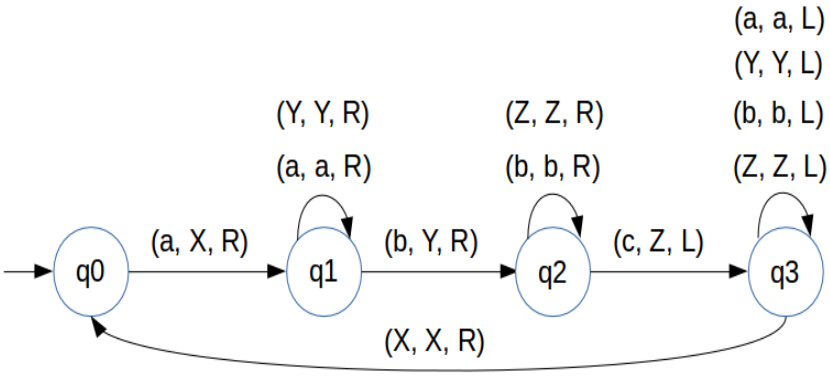
1. Following Steps:
   a. Mark 'a' with 'X' and move towards unmarked 'b'
   b. Move towards unmarked 'b' by passing all 'a's
   c. To move towards unmarked 'b' also pass all 'Y's if exist



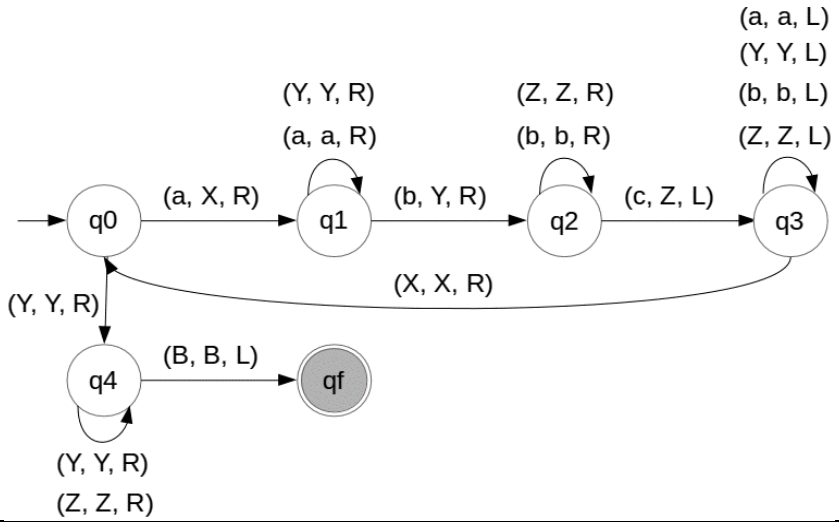
2.Following Steps:
a. Mark 'b' with 'Y' and move towards unmarked 'c'
b. Move towards unmarked 'c' by passing all 'b's
c. To move towards unmarked 'c' also pass all 'Z's if exist



3. Following Steps:
a. Mark 'c' with 'Z' and move towards first 'X' (in left)
b. Move towards first 'X' by passing all 'Z's, 'b's, 'Y's and 'a's
c. When 'X' is reached just move one step right by doing nothing.



4. To check all the 'a's, 'b's and 'c's are over add loops for checking 'Y' and 'Z' after "we get 'X' followed by 'Y'"

	To reach final state (qf) just replace BLANK with BLANK and move either direction.  		

6. Attempt any *one* part of the following: 10x1 = 10

Q.no	Questions	Marks	CO
(a)	"Discuss in detail about the Pigeonhole Principle with the help of suitable example." Solution: If there are n people who can shake hands with one another (where n > 1), the pigeonhole principle shows that there is always a pair of people who will shake hands with the same number of people. In this application of the principle, the 'hole' to which a person is assigned is the number of hands shaken by that person. Theorem (The Pigeonhole Principle): If m. objects are distributed into n bins and m > n, then at. least one bin will contain at least two objects. Theorem (The Pigeonhole Principle): If m. objects are distributed into n bins and m > n, then at least one bin will contain at least two objects. In mathematics, the pigeonhole principle states that if n items are put into m containers, with n > m, then at least one container must contain more than one item. For example, if one has three gloves (and none is ambidextrous/reversible), then there must be at least two right-handed gloves, or at least two left-handed gloves, because there are three objects, but only two categories of handedness to put them into. This seemingly obvious statement, a type of counting argument, can be used to demonstrate possibly unexpected results. For example, given that the population of London is greater than the maximum number of hairs that can be present on a human's head, then the pigeonhole principle requires that there must be at least two people in London who have the same number of hairs on their heads.	10	2

(b)	Minimize the given DFA shown below (Figure A).	10	1

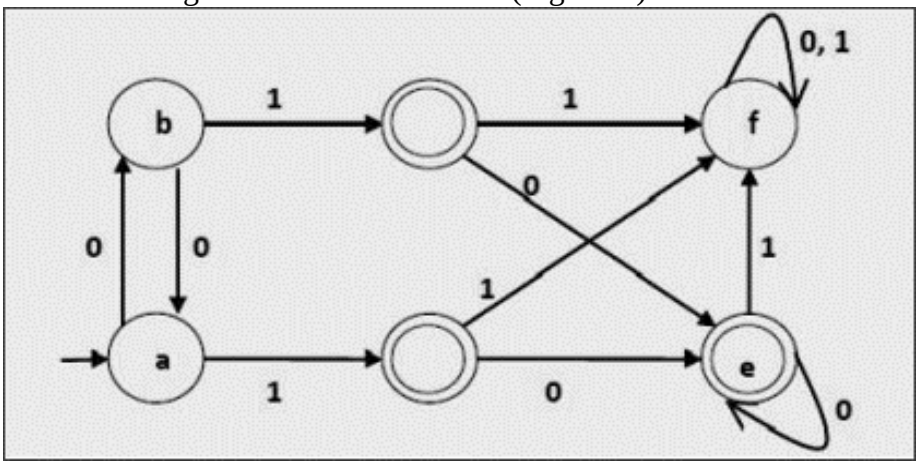
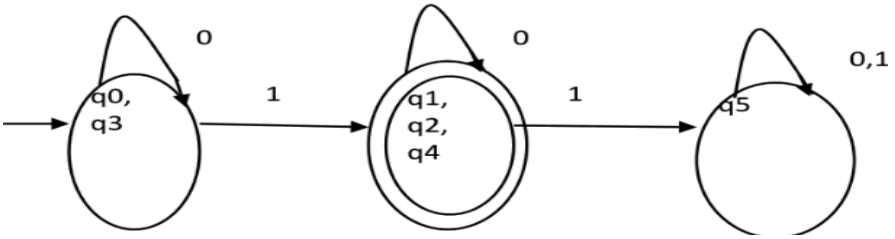


Figure A

Solution:

Step 1. P0 will have two sets of states. One set will contain q1, q2, q4 which are final states of DFA and another set will contain remaining states. So P0 = {{q1, q2, q4}, {q0, q3, q5}}.
Step 2. To calculate P1, we will check whether sets of partition P0 can be partitioned or not:
i) For set { q1, q2, q4 } :
δ (q1, 0) = δ (q2, 0) = q2 and δ (q1, 1) = δ (q2, 1) = q5, So q1 and q2 are not distinguishable.
Similarly, δ (q1, 0) = δ (q4, 0) = q2 and δ (q1, 1) = δ (q4, 1) = q5, So q1 and q4 are not distinguishable.

Since, q1 and q2 are not distinguishable and q1 and q4 are also not distinguishable, So q2 and q4 are not distinguishable. So, {q1, q2, q4} set will not be partitioned in P1.
ii) For set { q0, q3, q5 } :
δ (q0, 0) = q3 and δ (q3, 0) = q0
δ (q0, 1) = q1 and δ(q3, 1) = q4
Moves of q0 and q3 on input symbol 0 are q3 and q0 respectively which are in same set in partition P0. Similarly, Moves of q0 and q3 on input symbol 1 are q1 and q4 which are in same set in partition P0. So, q0 and q3 are not distinguishable.
δ (q0, 0) = q3 and δ (q5, 0) = q5 and δ (q0, 1) = q1 and δ (q5, 1) = q5
Moves of q0 and q5 on input symbol 1 are q1 and q5 respectively which are in different set in partition P0. So, q0 and q5 are distinguishable. So, set { q0, q3, q5 } will be partitioned into { q0, q3 } and { q5 }. So,
P1 = { { q1, q2, q4 }, { q0, q3}, { q5 } }
To calculate P2, we will check whether sets of partition P1 can be partitioned or not:
iii)For set { q1, q2, q4 } :
δ (q1, 0) = δ (q2, 0) = q2 and δ (q1, 1) = δ (q2, 1) = q5, So q1 and q2 are not distinguishable.
Similarly, δ (q1, 0) = δ (q4, 0) = q2 and δ (q1, 1) = δ (q4, 1) = q5, So q1 and q4 are not distinguishable.
Since, q1 and q2 are not distinguishable and q1 and q4 are also not distinguishable, So q2 and q4 are not distinguishable. So, { q1, q2, q4 } set will not be partitioned in P2.
iv)For set { q0, q3 } :
δ (q0, 0) = q3 and δ (q3, 0) = q0
δ (q0, 1) = q1 and δ (q3, 1) = q4
Moves of q0 and q3 on input symbol 0 are q3 and q0 respectively which are in same set in partition P1. Similarly, Moves of q0 and q3 on input symbol 1 are q1 and q4 which are in same set in partition P1. So, q0 and

	q3 are not distinguishable. v) For set {q5}: Since we have only one state in this set, it can't be further partitioned. So, P2 = {{q1, q2, q4 }, { q0, q3 }, { q5 } } Since, P1=P2. So, this is the final partition. Partition P2 means that q1, q2 and q4 states are merged into one. Similarly, q0 and q3 are merged into one. Minimized DFA  		

7. Attempt any *one* part of the following: 10x1 = 10

Q.no	Questions	Marks	CO
(a)	Explain in detail about the following. i) Closure properties of Regular Languages ii) Decidability- Decision properties of Regular Languages Solution: Closure properties of Regular Languages: In an automata theory, there are different closure properties for regular languages. They are as follows − -Union -Intersection -concatenation -Kleene closure -Complement Let see one by one with an example Union If L1 and If L2 are two regular languages, their union L1 U L2 will also be regular. Example L1 = $\{a^n \mid n > O\}$ and L2 = $\{b^n \mid n > O\}$ L3 = L1 U L2 = $\{a^n$ U $b^n \mid n > O\}$ is also regular. Intersection: If L1 and If L2 are two regular languages, their intersection L1 ∩ L2 will also be regular. Example L1= $\{a^m b^n \mid n > 0$ and $m > O\}$ and L2= $\{a^m b^n$ U bn am $\mid n > 0$ and $m > O\}$ L3 = L1 ∩ L2 = $\{a^m b^n \mid n > 0$ and $m > O\}$ are also regular. Concatenation: If L1 and If L2 are two regular languages, their concatenation L1.L2 will also be regular.	10	4

	Example L1 = $\{a^n \mid n > 0\}$ and L2 = $\{b^n \mid n > O\}$ L3 = L1.L2 = $\{a^m.b^n \mid m > 0$ and $n > O\}$ is also regular. Kleene Closure: If L1 is a regular language, its Kleene closure L1* will also be regular. Example L1 = (a U b) L1* = (aU b) * Complement: If L(G) is a regular language, its complement L'(G) will also be regular. Complement of a language can be found by subtracting strings which are in L(G) from all possible strings. Closure properties on regular languages are defined as certain operations on regular language which are guaranteed to produce regular language. Closure refers to some operation on a language, resulting in a new language that is of same "type" as originally operated on i.e., regular. -Regular Languages are closed under intersection, i.e., if L1 and L2 are regular then L1 ∩ L2 is also regular. -Closed under Union Operation. -Closed under Concatenation. -Closed under Star operation. ii)Decidability- Decision properties of Regular Languages: A decision property for a class of languages is an algorithm that takes a formal description of a language (e.g., a DFA) and tells whether or not some property holds. Example: Is language L empty? When we talked about protocols represented as DFA's, we noted that important properties of a good protocol were related to the language of the DFA. Example: "Does the protocol terminate?" = "Is the language finite?" Example: "Can the protocol fail?" = "Is the language nonempty? We might want a "smallest" representation for a language, e.g., a minimum-state DFA or a shortest RE. υIf you can't decide "Are these two languages the same?" ω I.e., do two DFA's define the same language? You can't find a "smallest. A closure property of a language class says that given languages in the class, an operator (e.g., union) produces another language in the same class. Example: the regular languages are obviously closed under union, concatenation, and (Kleene) closure. ω Use the RE representation of languages		
(b)	Check whether the grammar is ambiguous or not. R-> R+R/ R-R/ R*R/ a / b / c. Obtain the string w = a+b*c Solution:	10	3

Given

$R \rightarrow R+R \mid R-R \mid R*R \mid a \mid b \mid c$

and string $w = a+b*c$

We have to show that given grammar is ambiguous or not?

Now, we are going to show two possible leftmost derivation trees for same string $w = a+b*c$ in order to prove that given grammar is ambiguous Grammar.

Leftmost derivation 1:

$R \rightarrow R+R \rightarrow a+R \rightarrow a+R*R$

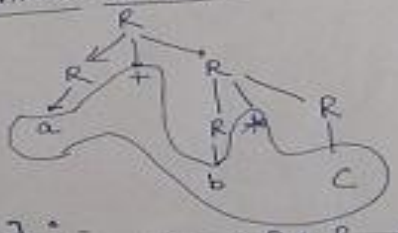
$\downarrow$

$a+b*R$

$\downarrow$

$a+b*c$

Leftmost derivation tree 1:



Leftmost derivation 2:-

$R \rightarrow R*R \rightarrow R+R*R$

$\downarrow$

$a+R*R$

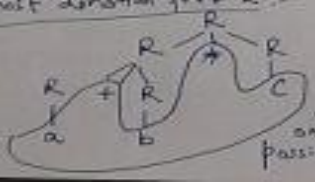
$\downarrow$

$a+b*R$

$\downarrow$

$a+b*c$

Leftmost derivation tree 2:-



Since more than one Leftmost derivation trees are possible Hence it is ambiguous Grammar.