GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)

Seshadri Rao Knowledge Village, Gudlavalleru – 521356.

Department of Computer Science and Engineering

HANDOUT

on

FORMAL LANGUAGES AND AUTOMATA THEORY

Vision

To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society.

Mission

- To impart quality education through well-designed curriculum in tune with the growing software needs of the industry.
- To serve our students by inculcating in them problem solving, leadership, teamwork skills and the value of commitment to quality, ethical behavior & respect for others.
- To foster industry-academia relationship for mutual benefit and growth.

Program Educational Objectives

PEO1: Identify, analyze, formulate and solve Computer Science and Engineering problems both independently and in a team environment by using the appropriate modern tools.

PEO2: Manage software projects with significant technical, legal, ethical, social, environmental and economic considerations.

PEO3: Demonstrate commitment and progress in lifelong learning, professional development, leadership and Communicate effectively with professional clients and the public.

HANDOUT ON FORMAL LANGUAGES AND AUTOMATA THEORY

Class & Sem.: II B.Tech – II Semester Year : 2018-19
Branch : CSE Credits : 3

==

1. Brief History and Scope of the Subject

Computer science has two major components:

1) the fundamental ideas and models underlying computing,

2) Engineering techniques for the design of computing systems, both hardware and software, especially the application of theory to design.

This subject is intended as an introduction to the first area, the fundamental ideas underlying computing.

Theoretical computer science had its beginnings in a number of diverse fields: biologists studying models for neuron nets, electrical engineers developing switching theory as a tool to hardware design, mathematicians working on the foundations of logic, and linguists investigating grammars for natural languages. Out of these studies came models that are central to theoretical computer science.

The notions of finite automata and regular expressions (Units 1, 2 and 3) were originally developed with neuron nets and switching circuits in mind. Recently, they have served as useful tools in the design of lexical analyzers, the part of a compiler that groups characters into tokens-indivisible units such as variable names and keywords. A number of compiler-writing systems automatically transform regular expressions into finite automata for use as lexical analyzers. A number of other uses for regular expressions and finite automata have been found in text editors, pattern matching, various text-processing and file-searching programs, and as mathematical concepts with application to other areas, such as logic.

The notion of a context-free grammar and the corresponding pushdown automaton (Units 4 and 5) has aided immensely the specification of programming languages and in the design of parsers-another key portion of a compiler. Formal specifications of programming languages have replaced extensive and often incomplete or ambiguous descriptions of languages. Understanding the capabilities of the pushdown automaton has greatly simplified parsing. In early compilers, parser design is a difficult problem, and many of the early parsers were quite inefficient and unnecessarily restrictive. Based on context-free-grammar-based techniques, parser design is no longer a problem, and parsing occupies only a few percent of the time spent in typical compilation.

In Unit 6, we deal with Turing machines and one of the fundamental problems of computer science; there are algorithms for computing functions. There are functions that are simply not computable; that is, there is no computer program that can ever be written.

2. Pre-Requisites

- Mathematical Foundation of Computer Science

3. Course Objectives:

- To introduce the classification of machines by their power to recognize languages and to solve problems in computing.
- To familiarize how to employ deterministic and non-deterministic machines.

Course Outcomes:

CO1: compare the automata based on their recognizing power.

CO2: design finite automata for regular languages.

CO3: reduce DFA by applying minimization algorithm.

CO4: write regular expressions for regular languages or for DFA by applying Arden’s theorem.

CO5: generate grammar for CFL’s.

CO6: use algorithm to simplify grammar.

CO7: design PDA‘s for context free languages.

CO8: design Turing Machine for the phrase-structured languages.

4. Program Outcomes:

Engineering Graduates will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

5. Mapping of Course Outcomes with Program Outcomes:

	1	2	3	4	5	6	7	8	9	10	11	12
CO1	H	H			H							
CO2		H	L		H							
CO3	H		L		H							
CO4	M	H										
CO5		H	M		M							
CO6	M											
CO7	M	H	L		M							
CO8	M	H	L		M							

6. Prescribed Text Books

1. John E.Hopcroft, Rajeev Motwani & Jeffrey D.Ullman J.D., "Introduction to Automata Theory Languages and Computation", 3rd edition, Pearson Education.
2. Lewis H.R., Papdimitriou, "Elements of Theory of Computation", 2nd edition, PHI.

7. Reference Books

1. Daniel I.A. Cohen, John Wiley, “Introduction to languages and the Theory of Computation”.
2. Sipser, Thomson, “Introduction to Theory of Computation”, 2nd edition.
3. Mishra and Chandrashekaran, “Theory of computer science - Automata, Languages, and Computation”, 2nd edition, PHI.
4. K.Krithivasan and R.Rama; Introduction to Formal Languages, Automata Theory and Computation; Pearson Education, 2009.

8. URLs and Other E-Learning Resources

1. Basis for a Mathematical TOC: http://www-formal.stanford.edu/jmc/basis1.pdf
2. Finite Automta: http://www.cs.odu.edu/~toida/nerzic/390teched/regular/fa/intr_2_fa.html
3. PDA: https://brilliant.org/wiki/pushdown-automata/
4. Turing Machine: http://plato.stanford.edu/entries/turing-machine

9. Digital Learning Materials:

- http://nptel.ac.in/courses/106104028/
- http://nptel.ac.in/courses/106104148/
- http://nptel.ac.in/courses/106106049/

10. Lecture Schedule / Lesson Plan

<table>
<tr><th rowspan="2">Topic</th><th colspan="2">No. of Periods</th></tr>
<tr><th>Theory</th><th>Tutorial</th></tr>
<tr><td colspan="3">UNIT –1: Fundamentals</td></tr>
<tr><td>Strings, Alphabet, Language, Operations on strings</td><td>1</td><td rowspan="3">1</td></tr>
<tr><td>Operations on languages, Finite State System</td><td>1</td></tr>
<tr><td>Finite Automaton Model</td><td>1</td></tr>
<tr><td>Acceptance of strings and languages</td><td>1</td><td rowspan="4">2</td></tr>
<tr><td>Deterministic finite automaton</td><td>2</td></tr>
<tr><td>Non deterministic finite automaton</td><td>2</td></tr>
<tr><td>Transition diagrams, language recognizers and applications of Finite Automata</td><td>2</td></tr>
</table>

Total	10+3(T)	
UNIT – 2: Finite Automata		
NFA with ε transitions – significance, acceptance of a language by a ε –NFA	1	1
Equivalence between NFA with and without ε transitions	2	
Minimization of FSM	2	
NFA to DFA conversion	1	
equivalence between two FSM‘s	1	
Finite automata with outputs - Moore machine, Mealy machines	1	1
Moore to Mealy Coversion-examples	1	
Mealy to Moore conversion-examples	1	
Total	10+2(T)	
UNIT – 3: Regular Languages		
Regular Sets, Identity Rules	1	1
Regular expressions	2	
Construction of finite Automata for a given regular expressions	1	1
Construction of regular expression for a given finite Automata	1	
Pumping lemma of regular sets	1	
Closure properties of regular sets, applications of regular languages.	1	
Total	7+2(T)	
UNIT – 4: Grammar Formalism		
Chomsky hierarchy of languages	1	1
Regular grammars - right linear and left linear grammars-examples	1	
Equivalence between regular linear grammar and FA	1	
Equivalence between FA and regular grammar	1	
Context free grammar-examples	2	

Derivation- Rightmost and leftmost derivation of strings, sentential forms, Derivation trees	2	1
Total	8+2(T)	
UNIT – 5: Context Free Grammars		
Ambiguity in context free grammars	1	1
Minimization of Context Free Grammars	1	
Chomsky normal form	1	
Greibach normal form	2	
Pumping Lemma for Context Free Languages	1	
Enumeration of Properties of CFL (proofs not required), applications of CFLs	1	
Push down automata, model of PDA	1	1
Design of PDA	2	
Applications of PDA	1	
Total	11+2(T)	
UNIT – 6: Turing Machine		
Turing Machine, model	1	1
Design of TM	2	
Types of Turing Machines	1	1
Computable functions	1	
Recursively enumerable languages, Recursive languages	1	
Decidability of problems	1	
Undecidability of posts correspondence problem	1	
Total	8+2(T)	
Total No.of Periods:	**54**	**13(T)**

FORMAL LANGUAGES AND AUTOMATA THEORY

UNIT-I

Objective:

- To introduce the classification of machines by their power to recognize languages and to solve problems in computing.
- To familiarize how to employ deterministic and non-deterministic finite automata.

Syllabus:

Strings, alphabet, language, operations, finite state machine, finite automaton model, acceptance of strings and languages, deterministic finite automaton and non deterministic finite automaton, transition diagrams and language recognizers.

Learning Outcomes:

Students will be able to:

- Understand the basic definitions like alphabet, string, language and their operations.
- Understand the model of FA.
- Design DFA and NFA for the given regular language.
- Test the designed DFA and NFA for the set of strings that belongs to L and for the set of strings that doesn’t belongs to L.

Learning Material

Alphabet:

An alphabet is a finite, nonempty set of symbols. It is denoted by $\sum$.

Example:

$\sum = \{0, 1\}$ is binary alphabet consisting of the symbols 0 and 1.

$\sum = \{a, b, c \ldots z\}$ is lowercase English alphabet.

Powers of an Alphabet

If Σ is an alphabet, we can express the set of all strings of a certain length from that alphabet by using the exponential notation. It is denoted by Σ^k - the set of strings of length k.

Example:

$\Sigma^0 = \{\varepsilon\}$, regardless of what alphabet Σ is. ε is the only string of length 0.

If $\Sigma = \{0, 1\}$ then,

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

$\Sigma^2 = \{00, 01, 10, 11\}$

$\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

The set of all strings over an alphabet Σ is denoted by Σ^*. $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$

For example, $\{0, 1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$

The symbol $*$ is called ***Kleene star*** and is named after the mathematician and logician Stephen Cole Kleene.

The symbol **+** is called ***Positive closure*** i.e. $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \ldots$

$$\boxed{\Sigma^* = \Sigma^+ \cup \{\varepsilon\}}$$

String:

A string (or word) is a finite sequence of symbols chosen from some alphabet. The letters u, v, w, x, y and z are used to denote string.

Example:

If $\Sigma = \{a, b, c\}$ then abcb is a string formed from that alphabet.

- The ***length*** of a string w, denoted **|w|**, is the number of symbols composing the string.

Example:

The string abcb has length 4.

- The ***empty string*** denoted by **ε**, is the string consisting of zero symbols. Thus |ε| =0.

Operations on strings:

- ***Concatenation of strings***

  The concatenation of two strings is the string formed by writing the first, followed by the second, with no intervening space. Concatenation of strings is denoted by ◦.

  That is, if w and x are strings, then wx is the concatenation of these two strings.

  Example:

  The concatenation of dog and house is doghouse.

  Let x=0100101 and y= 1111 then x ◦ y=01001011111

- ***String Reversal***

  Reversing a string means writing the string backwards.

  It is denoted by w^R

  Example:

  Reverse of the string abcd is dcba.

  Note: If w= w^R, then that string is called palindrome.

- ***Substring***

  A substring is a part of a string.

  Example:

  If abcd is string then possible substrings are **ε**,a,b,c,d,ab,bc,cd,abc,bcd are proper substrings for the given string

  A ***prefix*** of a string is any number of leading symbols of that string.

  A ***suffix*** of a string is any number of trailing symbols.

  Example:

String abc has prefixes **ε**, a, ab, and abc; its suffixes are **ε**, c, bc, and abc.

A prefix or suffix of a string, other than the string itself, is called a ***proper prefix or suffix***.

Language:

A (formal) language is a set of strings of symbols from someone alphabet. It is denoted by L. We denote this language by ∑*.

- The empty set, Ø, and the set consisting of the empty string {**ε**} are languages.

Example:

If ∑= {a}, then ∑* = {**ε**, a, aa, aaa, ...}.

If ∑ = {0, 1}, then ∑* = {**ε**, 0, 1, 00, 01, 10, 11, 000,...}.

Operations on languages:

- ***Union***

  If L1 and L2 are two languages over an alphabet ∑.Then the union of L1 and L2 is denoted by L1 U L2.

  Example:

  L1={0,01,011} and L2={001}, then L1 U L2={0,01,011,001}

- ***Intersection***

  If L1 and L2 are two languages over an alphabet ∑.Then the intersection of L1 and L2 is denoted by L1 ∩ L2.

  Example:

  L1= {0, 01,011} and L2= {01}, then L1 ∩ L2= {01}

- ***Complementation***

  L is a language over an alphabet ∑, then the complement of L denoted by L', is the language consisting of those strings that are not in L over the alphabet.

  Example:

  If ∑={a,b} and L={a,b,aa}, then

  L'= ∑*-L = {ε,a,b,aa,bb,ab.........} - {a,b,aa} = {ε,bb,ab,ba.........}

- ***Concatenation***

  Concatenation of two languages L1 and L2 is the language L1 o L2 ,each element of which is a string formed by combining one string of L1 with another string of L2.

  Example:

  L1={bc,bcc,cc}and L2={cc,ccc},then L1oL2 = {bccc,bcccc,bcccccc,cccc,ccccc}

- ***Reversal***

  If L is language, then L^R is obtained by reversing the corresponding string in L. This operation is similar to the reversal of a string.

  $$\boxed{L^R = \{w^R \mid w \in L\}}$$

  Example:

  If L= {0, 011, 0111}, then L^R = {0, 110, 1110}

- ***Kleene Closure***

  The Kleene closure (or just closure) of L, denoted L*, is the set

  $$\boxed{L^* = \bigcup_{i=0}^{\infty} L^i}$$

- ***Positive Closure***

  The positive closure of L, denoted L^+, is the set

  $$\boxed{L^+ = \bigcup_{i=1}^{\infty} L^i}$$

That is, L* denotes words constructed by concatenating any number of words from L.

L+ is the same, where **ε**, is excluded.

Note: L+ contains **ε** if and only if L does.

Example:

Let L1 = {10, 1}

L * = L0 U L1 U L2................. = {ε, 1, 10, 11, 111, 1111 ,..........}

L ⁺ = L1 U L2 U L3..................= {1, 10, 11, 111, 1111..........}

Finite Automaton:

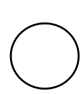
- A finite automaton (FA) consists of a finite set of states and a set of transitions from state to state that occur on input symbols chosen from an alphabet ∑.
- For each input symbol there is exactly one transition out of each state (possibly back to the state itself).
- One state, usually denoted q_0 is the initial state, in which the automaton starts. Some states are designated as final or accepting states.

Formally, a finite automaton is denoted by a 5-tuple **(Q, ∑, δ, q_o, F),** where

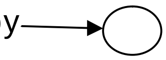
- Q is a finite set of states.
- ∑ is a finite input alphabet.
- **δ** is the transition function mapping Q x ∑ to Q i.e., **δ** (q,a) is a state for each state q and input symbol a.
- qo ∈ Q is the initial state.
- F ⊆ Q is the set of final states. It is assumed here that there may be
- more than one final state.

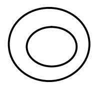
Transition Diagram:

- A transition diagram is a directed graph associated with an FA in which the vertices of the graph correspond to the states of the FA.
- If there is a transition from state q to state p on input a, then there is an arc labelled a from state q to state p in the transition diagram.

State is denoted by 

Transition is denoted by $\longrightarrow$

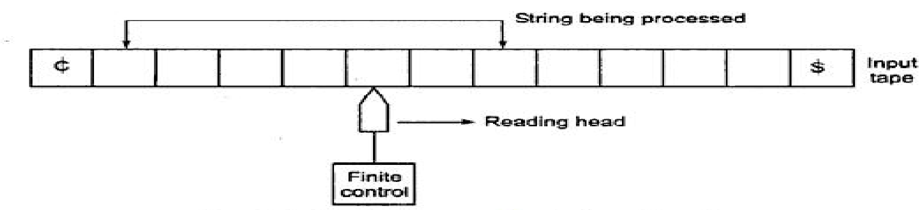
Initial state is denoted by 

Final state is denoted by 

Transition Table:

A tabular representation in which rows correspond to states, columns correspond to inputs and entries correspond to next states.

Finite Automata Model:



Block diagram of a finite automaton

The various components are explained as follows:

(i) ***Input tape:***

- The input tape is divided into squares, each square containing a single symbol from the input alphabet $\sum$.
- The end squares of the tape contain the endmarker ¢ at the left end and the endmarker $ at the right end.
- The absence of endmarkers indicates that the tape is of infinite length. The left-to-right sequence of symbols between the two endmarkers is the input string to be processed.

(ii) ***Reading head:***

- The head examines only one square at a time and can move one square either to the left or to the right.

- For further analysis, we restrict the movement of the R-head only to the right side.

(iii) ***Finite control:*** The input to the finite control will usually be the symbol under the R-head, say a, and the present state of the machine, say q, to give the following outputs:

- A motion of R-head along the tape to the next square (in some a null move, i.e. the R-head remaining to the same square is permitted)
- The next state of the finite state machine given by $\delta(q, a)$.

Acceptance of String by a Finite Automaton:

The FA accepts a string x if the sequence of transitions corresponding to the symbols of x leads from the start state to an accepting state and the entire string has to be consumed, i.e., a string x is accepted by a finite automaton M = $\mathbf{(Q, \sum, \delta, q_0, F)}$

$$\boxed{\text{if } \boldsymbol{\delta}(q_0, x) = q \text{ for some } q \in F.}$$

This is basically the acceptability of a string by the final state.

Note: A final state is also called an accepting state.

Transition function δ and for any two input strings x and y,

$$\boxed{\boldsymbol{\delta(q, xy) = \delta(\delta(q, x), y)}}$$

Example:

Consider the finite state machine whose transition function δ is given in the form of a transition table. Here $Q = \{q_0, q_1, q_2, q_3\}$, $\sum=\{0,1\}$, $F=\{q_0\}$. Give the entire sequence of states for the input string 110101.

State	Input	
	0	**1**
→q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

$$\delta(q_0, 110101) = \delta(q_1, 10101)$$
$$= \delta(q_0, 0101)$$
$$= \delta(q_2, 101)$$
$$= \delta(q_3, 01)$$
$$= \delta(q_1, 1) = q_0$$

q_0 is final state, therefore given string is accepted by finite automata.

Deterministic finite automaton:

Formally, a deterministic finite automaton can be represented by a 5-tuple M= **(Q, $\sum$, δ, q_o, F),**

where

- Q is a finite set of states.
- $\sum$ is a finite input alphabet.
- **δ** is the transition function mapping Q x $\sum$ to Q i.e., **δ** (q,a) is a state for each state q and input symbol a.
- $q_o \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final states. It is assumed here that there may be more than one final state.

Steps to design a DFA:

1. Understand the language for which the DFA has to be designed and write the language for the set of strings starting with minimum string that are accepted by FA.

2. Draw transition diagram for the minimum length string.
3. Obtain the possible transitions to be made for each state on each input symbol.
4. Draw the transition table.
5. Test DFA with few strings that are accepted and few strings that are rejected by the given language.
6. Represent DFA with tuples.

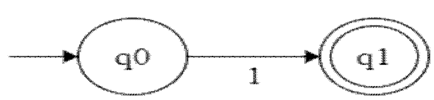
Examples

1. **Design DFA that accepts all strings which starts with '1' over the alphabet {0,1}**

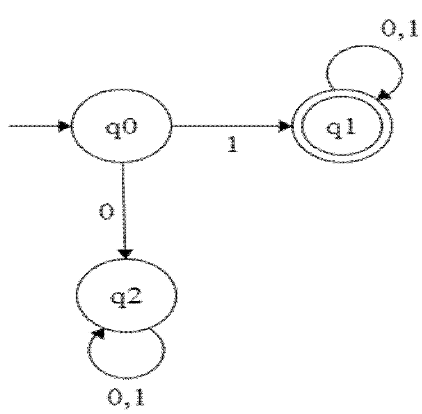
   Step 1: Understand the language for which the DFA has to be designed and write the language for the set of strings starting with minimum string that is accepted by FA.

   L = {1, 10, 11, 100, 110, 101, 111, ...}

   Step 2: Draw transition diagram for the minimum length string.



   Step 3: Obtain the possible transitions to be made for each state on each input symbol.



   Step 4: Draw the transition table.

State	Input	
	0	1
→q_0	q_2	q_1
q_1	q_1	q_1
q_2	q_2	q_2

Step 5: Test DFA with few strings that are accepted and few strings that are rejected by the given language.

Case i) Let w=1001 ∈ L

$\delta(q_0,1001) = \delta(q1,010)$

$= \delta(q_1,10) = \delta(q_1,0) = q_1$

q_1 is final state and the entire string has been consumed i.e., given string is accepted by DFA.

Case ii) Let w=0001 ∉ L

$\delta(q0,0001) = \delta(q2,001)$

$= \delta(q2,10)$

$= \delta(q2,0)$

$= q_2$

q_2 is not final state and the entire string has been consumed i.e., given string is rejected by DFA.

Step 6: Represent DFA with tuples.

DFA, M= **(Q, ∑, δ, qo, F)**

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

∑ **=** { 0,1 }

δ: $\delta(q_0,0)=q_2$

$\delta(q_0,1)=q_1$

$\delta(q_1,0)=q_1$

$\delta(q_1,1)=q_1$

$\delta(q_2,0)=q_2$

$\delta(q_2,1)=q_2$

q_0 – initial state

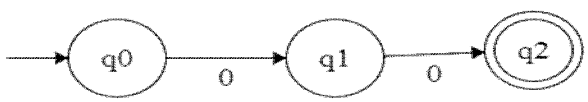
F – final state = { q_1}

2. Design DFA that accepts all strings which contains '00' as substring over the alphabet {0,1}

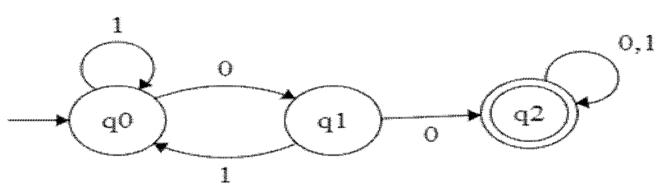
Step 1: Understand the language for which the DFA has to be designed and write the language for the set of strings starting with minimum string that is accepted by FA.

L={00,100,000,001,1100,1000,0100,1001,0001,11000,11100,................}

Step 2: Draw transition diagram for the minimum length string.



Step 3 : Obtain the possible transitions to be made for each state on each input symbol.



Step 4: Draw the transition table.

State	Input	
	0	1
→ q_0	q_1	q_0
$q1$	q_2	q_0
q_2	q_2	q_2

Step 5: Test DFA with few strings that are accepted and few strings that are rejected by the given language.

Case i) Let w = 1001 ∈ L

$$\delta(q_0,1001) = \delta(q_0,001)$$
$$= \delta(q_1,01)$$
$$= \delta(q_2,1)$$
$$= q_2$$

q2 is final state and the entire string has been consumed i.e., given string is accepted by DFA.

Case ii) Let w=1011 ∉ L

$$\delta(q_0,1011) = \delta(q_0,011)$$
$$= \delta(q_1,11)$$
$$= \delta(q_0,1)$$
$$= q_0$$

q_0 is not final state and the entire string has been consumed i.e., given string is rejected by DFA.

Step 6: Represent DFA with tuples.

DFA, M= **(Q, ∑, δ, qo, F)**

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

∑ **=** { 0,1 }

δ: $\delta(q_0,0)=q_1$

$\delta(q_0,1)=q_0$

$\delta(q_1,0)=q_2$

$\delta(q_1,1)=q_0$

$\delta(q_2,0)=q_2$

$\delta(q_2,1)=q_2$

q_0 – initial state

F – final state = { q_2 }

Nondeterministic finite automaton (NDFA/NFA):

A nondeterministic finite automaton is a 5-tuple (Q, ∑, δ, qo, F), where

- Q is a finite nonempty set of states;
- ∑ is a finite nonempty set of inputs;
- δ is the transition function mapping from Q x ∑ into 2^Q which is the power set of Q, the set of all subsets of Q;
- qo ∈ Q is the initial state; and
- F ⊆ Q is the set of final states

Steps to design a NFA:

1. Understand the language for which the NFA has to be designed and write the language for the set of strings starting with minimum string that is accepted by FA.
2. Draw transition diagram for the minimum length string.
3. Obtain the possible transitions to be made for each state on each input symbol.
4. Draw the transition table.
5. Test NFA with few strings that are accepted and few strings that are rejected by the given language.
6. Represent NFA with tuples.

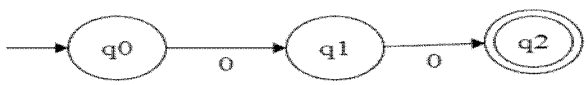
Examples:

1. **Design NFA that accepts all strings which contains ‘00’ as substring over the alphabet {0,1}**

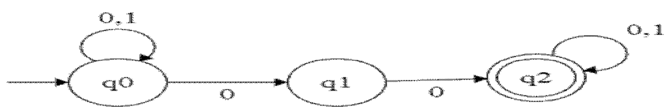
Step 1: Understand the language for which the NFA has to be designed and write the language for the set of strings starting with minimum string that is accepted by FA

L={00,100,000,001,0100,1100,1000,1001,0001,11000,11100,.............}

Step 2: Draw transition diagram for the minimum length string.



Step 3: Obtain the possible transitions to be made for each state on each input symbol. **0, 1**



Step 4: Draw the transition table.

State	Input	
	0	**1**
→ q_0	$\{q_0,q_1\}$	q_0
q1	q_2	-
q_2	q_2	q_2

Step 5: Test NFA with few strings that are accepted and few strings that are rejected by the given language.

Case i) Let w=0100 ∈ L

$\delta(q_0,0100) = \delta(\{q_0,q_1\},100)$

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

$= \delta\ (\{q_0,\ q1\},0)$

$= \{q_0,\ q1,\ q2\}$

q_2 is final state and the entire string has been consumed i.e., given string is accepted by NFA.

Case ii) Let w=1011 ∉ L

$\delta(q_0,1011) = \delta(q_0,011)$

$= \delta(\{q_0,q_1\},11)$

$= \delta(q_0,1)$

$= q_0$

q_0 is not final state and the entire string has been consumed i.e., given string is rejected by NFA.

Step 6: Represent NFA with tuples.

NFA, M= **(Q, ∑, δ, qo, F)**

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

∑ = { 0,1 }

δ: $\delta(q_0,0)=\{q_0,q_1\}$

$\delta(q_0,1) = q_0$

$\delta(q_1,0) = q_2$

$\delta(q_1,1) = \emptyset$

$\delta(q_2,0) = q_2$

$\delta(q_2,1) = q_2$

q_0 – initial state

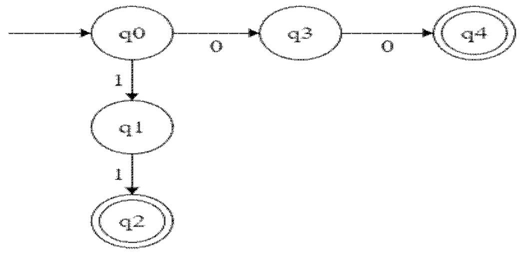
F – final state = $\{ q_2 \}$

2. Design NFA that accepts strings which contains either two consecutive 0's or two consecutive 1's.

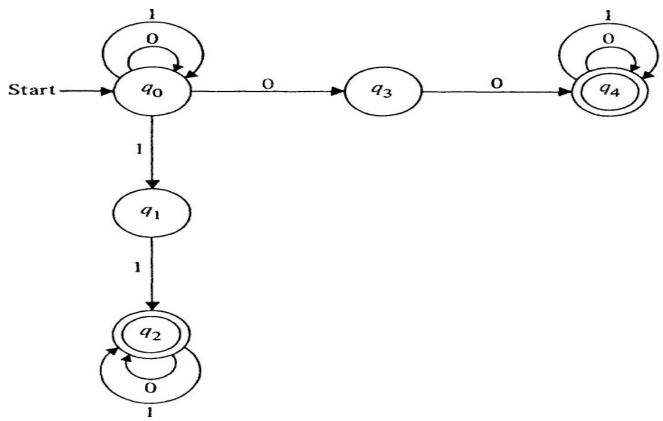
Step 1: Understand the language for which the NFA has to be designed and write the language for the set of strings starting with minimum string that is accepted by FA.

L={00,11,100,001,110,011,111,000,0100,1011,..............}

Step 2: Draw transition diagram for the minimum length string.



Step 3: Obtain the possible transitions to be made for each state on each input symbol.



Step 4: Draw the transition table.

State	Input	
	0	**1**
→ q_0	$\{q_0,q_3\}$	$\{q_0,q_1\}$
q_1	-	q_2
q_2	q_2	q_2
q_3	q_4	-
q_4	q_4	q_4

Step 5: Test NFA with few strings that are accepted and few strings that are rejected by the given language.

Case i) Let the input, w = 01001 ∈ L

$\delta(q_0,0) = \{q_0,q_3\}$

$\delta(q_0,01) = \delta(\delta(q_0,0),1) = \delta(\{q_0,q_3\},1) = \delta(q_0,1) \cup \delta(q_3,1) = \{q_0,q_1\}$

Similarly, we compute

$\delta(q_0,010) = \{q_0,q_3\}$, $\delta(q_0,0100) = \{q_0,q_3,q_4\}$

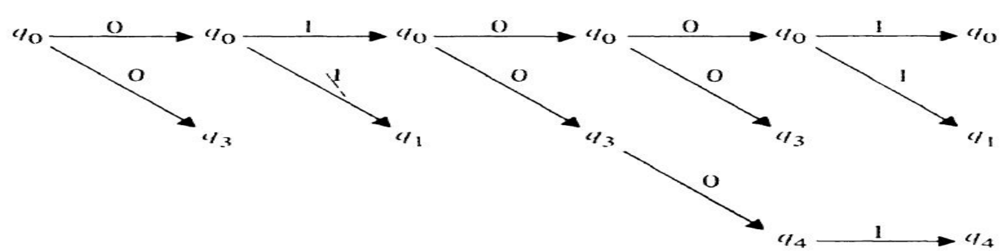
and

$\delta(q_0,01001) = \{q_0,q_1,q_4\}$

→ final state

After the entire string is consumed, the FA is in the state q_4.

As q_4 is the final state, the string is a accepted by FA



Case ii) Let w = 010 ∉ L

$\delta(q_0,010) = \delta(\{q_0,q_3\},10)$

$= \delta(\{q_0,q_1\},0)$

$= \{q_0,q_3\}$

There is no path to the final state after the entire string is consumed. So the string is rejected by FA.

Step 6: Represent NFA with tuples.

NFA, M= **(Q, ∑, δ, qo, F)**

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

∑ **=** { 0,1 }

δ: $\delta(q_0,0)=\{q_0,q_3\}$

$\delta(q_0,1)= \{q_0,q_1\}$

$\delta(q_1,0)= \varnothing$

$\delta(q_1,1)=q_2$

$\delta(q_2,0)=q_2$

$\delta(q_2,1)=q_2$

$\delta(q_3,0)=q_4$

$\delta(q_3,1)=\varnothing$

$\delta(q_4,0)=q_4$

$\delta(q_4,1)=q_4$

q_0 – initial state

F – final state = { q_2,q_4 }

Note: The minimal state DFA, accepting all strings over the alphabet {0, 1} where the nth symbol in every string from the right end is a 1, has 2^n states.

Language recognizers:

A language recognizer is a device that accepts valid strings produced in a given language. Finite state automata are formalized types of language recognizers. The language accepted by Finite Automata M designated L(M) is the set {x | $\delta(q0,x)$ is in F}.

Applications of FA:

- Used in Lexical analysis phase of a compiler to recognize tokens.
- Used in text editors for string matching.
- Software for designing and checking the behavior of digital circuits.

Limitations of FA:

- FA's will have finite amount of memory.
- The class of languages recognized by FA s is strictly the regular set. There are certain languages which are non regular i.e. cannot be recognized by any FA.

Differences between NFA and DFA:

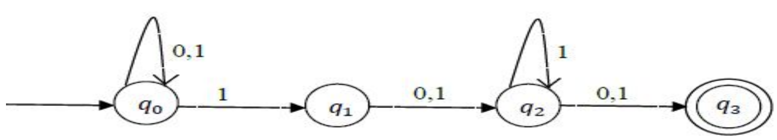
S.No	NFA	DFA
1	A nondeterministic finite automaton is a 5-tuple	A deterministic finite automaton can be represented by a 5-tuple

	M= (Q, ∑, δ**, q_o, F)**, where **δ**: Q x ∑ into 2^Q.	**M= (Q, ∑, δ, q_o, F),** where **δ**: Q x ∑ to Q.
2	NFA is the one in which there exists many paths for a specific input from current state to next state.	DFA is a FA in which there is only one path for a specific input from current state to next state.
3	NFA is easier to construct.	DFA is more difficult to construct.
4	NFA requires less space.	DFA requires more space.
5	Time required for executing an input string is more.	Time required for executing an input string is less.

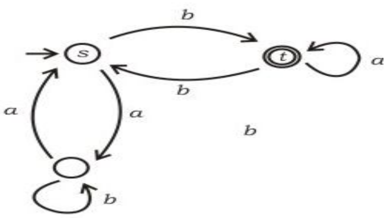
UNIT-I
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. The prefix of abc is [ ]
A) c B) bc C) b D) ε
2. $\sum^* = \sum^+ \cup \varepsilon$ [True | False]
3. Alphabet is _________________. [ ]
A) Finite collection of strings. B) Finite collection of symbols.
C) Finite collection of languages. D) All the above.
4. A _______ of a string is any number of leading symbols of that string.
5. __________ is a directed graph associated with an FA in which the vertices of the graph correspond to the states of the FA.
6. The transition function for NFA is a mapping function given as ____________.
7. The transition function for DFA is a mapping function given as _____________.
8. A = {a, b, c}. Power set of A= _____________.
9. FA has [ ]
A) Unlimited memory B) no memory at all
C) Limited memory D) none of the above.
10. Number of states requires to accept string ends with 10. [ ]
A) 3 B) 2 C) 1 D) can't be represented.
11. Consider the finite automaton in the following figure



What is the set of reachable states for the input string 0011? [ ]
A) {q_0, q_1, q_2} B) {q0, q1} C) {q0, q1, q2, q3} (D) {q3}
12. Given the language L = {ab, aa, baa}, which of the following strings are in L*?
1) abaabaaabaa 2) aaaabaaaa 3) baaaaabaaaab 4) baaaaabaa[ ]
A) 1,2and3 **B)** 2,3and4 **C)** 1,2and4 **D)** 1, 3 and 4
13. In the automaton below, s is the start state and t is the only final state.



Consider the strings u = abbaba, v = bab, and w = aabb. Which of the following statements is true? [ ]

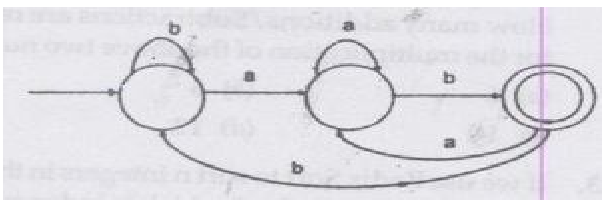
A) The automaton accepts u and v but not w

B) The automaton accepts each of u,v and w

C) The automaton rejects each of u,v and w

D) The automaton accepts u but rejects v and w

14. If the final states and non-final states in the DFA below are interchanged, then which of the following languages over the alphabet {a,b} will be accepted by the new DFA?

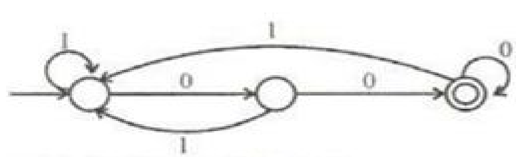


A) Set of all strings that do not end with ab [ ]
B) Set of all strings that begin with either an a or a b
C**)** Set of all strings that do not contain the substring ab,
D) All the above

15. What is the minimum number of states in the NFA accepting the language {a, ab}?
A) 3 B) 2 C)1 D) 4 [ ]

16. The smallest finite automation which accepts the language {x | length of x is divisible by 3} has [ ]
A) 2 states B) 3 states C) 4 states D) 5 states

17. The below DFA accepts the set of all strings over {0,1} that [ ]



a) begin either with 0 or 1 b) end with 0
c) end with 00 d) contain the substring 00

18. Consider a DFA over ∑={a,b} accepting allstrings which have number of a's divisible by 6 and number of b's divisible by 8. What is the number of states that the DFA will have? [ ]

A) 8 B) 14 C) 15 D) 48

SECTION-B

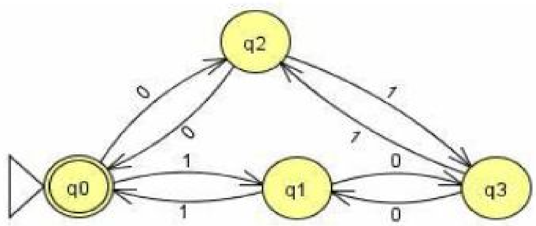
SUBJECTIVE QUESTIONS

1. Define string and alphabet.
2. Explain operations on strings and languages.
3. Define Positive Closure and Kleene Closure.
4. Define (i) Finite Automaton(FA) (ii)Transition diagram
5. Explain the model of FA.
6. Write the differences between NFA and DFA.
7. What is the difference between empty language and null string?
8. Which of the following Finite Automaton is having ambiguity and why?

   i) NFA ii) DFA
9. Draw the Finite state machine for accepting the languages □ and Ø.
10. From the given transition table. Check whether the following strings are accepted or not.
    i) 101101
    ii) 000000

Q/∑	0	1
→q0	q2	q1
q1	q3	q0
q2	q0	q3
q3	q1	q2

11. Construct DFA accepting the set of all strings beginning with 101.
12. Design a DFA for a language which contains strings of a's & b's and each string ends with aab.
13. Describe the words w in the language L accepted by the automaton in

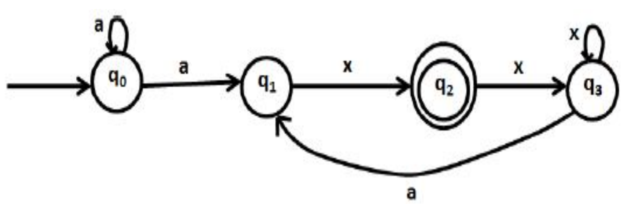


14. Design DFA accepting the set of all strings that begin with 01 and end with 11.
15. a) Design a DFA to accept the following language. L = {w: |w| mod 3 = 0} on Σ = {a}

    b) Design DFA accepting the language whose binary interpretation is divisible by 5 over the alphabet {0, 1}.
16. Design a DFA to accept strings of a's and b's having even number of a's and b's.
17. Design a DFA that accepts all strings over Σ={0,1} that do not contain 101 as a substring.
18. Design NFA that accepts the language of strings over Σ ={0,1} such that some two 0's are separated by a string whose length is 4i, for some i ≥0.
19. Design a NFA to accept strings of 0's & 1's such that each string ends with 00.
20. For the NFA given below;

    i. Check whether the string axxaxxa is accepted or not

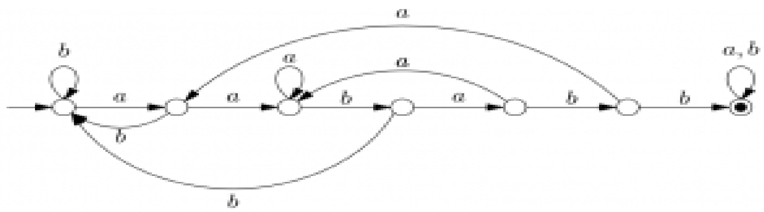
    ii. Give atleast two transition paths



SECTION-C

QUESTIONS AT THE LEVEL OF GATE

1. Consider the following Deterministic Finite Automata



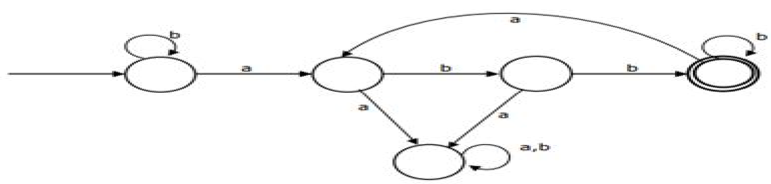
Which of the following is true? [ ]

A) It only accepts strings with prefix as “aababb”
B) It only accepts strings with substring as “aababb”
C) It only accepts strings with suffix as “aababb”
D) None of the above

2. The possible number of states of a deterministic finite automaton that accepts a regular language
$L = \{w_1aw_2 \mid w_1,w_2 \in \{a,b\}^*, |w_1| = 2, w_2>=3\}$ is _____
[GATE 2017 set-2]

3. Let w be any string of length n in {0, 1}*. Let L be the set of all substrings of w. What is the number of states in a non-deterministic finite automaton that accepts L? [ ]
A)n-1 B)n C)n+1 D)2n-1
[GATE2010]

4. Consider the machine M:

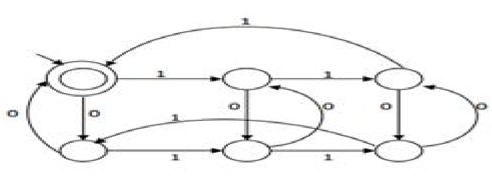


[GATE 2005]

The language recognized by M is: [ ]

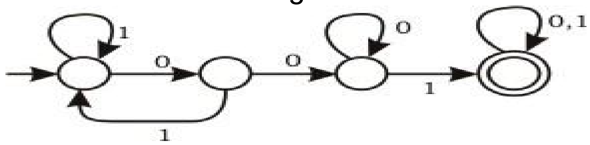
a) {w Є {a,b}* | every a in w is followed by exactly two b’s}
b) {w Є {a,b}* | every a in w is followed by at least two b’s}
c) {w Є {a,b}* | w contains the substring ‘abb’}
d) {w Є {a,b}* | w does not contain ‘aa’ as a substring}

5. The following finite state machine accepts all those binary strings in which the number of 1’s and 0’s are respectively [ ]
[GATE 2004]



a) divisible by 3 and 2
b) odd and even
c) even and odd
d) divisible by 2 and 3

6. Consider the following deterministic finite state automaton M.

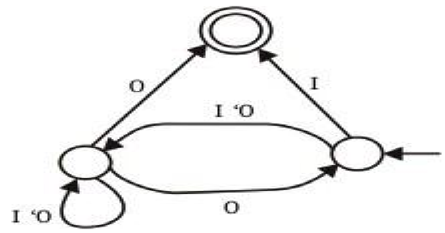


[GATE 2003]

Let S denote the set of seven bit binary strings in which the first, the fourth, and the last bits are 1. The number of strings in S that are accepted by M is [ ]

(A) 1 **(B)** 5 **(C)** 7 **(D)** 8

7. Consider the NFA M shown below.



Let the language accepted by M be L. Let L1 be the language accepted by the NFA M1, obtained by changing the accepting state of M to a non-accepting state and by changing the non-accepting state of M to accepting states. Which of the following Statements is true? [ ]

A) L1 = {0,1}* – L **B)** L1={0,1}*
C) L1 ⊆ L **D)** L1 = L

8. Construct a finite state machine that accepts the language, over {0,1} of all strings that contain neither the substring 00 nor the substring 11.

[Gate 2000]

9. What can be said about a regular language L over {a} whose minimal finite state automaton has two states? [ ]

[Gate 2000]

A) L must be $\{a^n \mid n \text{ is odd}\}$
B) L must be $\{a^n \mid n \text{ is even}\}$
C) L must be $\{a^n \mid {>=}0\}$
D) Either L must be $\{a^n \mid n \text{ is odd}\}$, or L must be $\{a^n \mid n \text{ is even}\}$

UNIT – II

Objective:
To familiarize how to employ non-deterministic finite automata with ε transitions and finite automata with outputs.

Syllabus:
Finite Automata:
NFA with ε transitions - significance, acceptance of languages, equivalence between NFA with and without ε transitions, NFA to DFA conversion, minimization of FSM, equivalence between two FSM‘s, finite automata with output-Moore and Mealy machines, applications of FA.

Learning Outcomes:
Students will be able to:

- Convert NFA to DFA and NFA with epsilon transitions to NFA without Epsilon transitions.
- Minimize the given DFA.
- Test whether the two DFA's are equivalent or not.
- Design Moore and Mealy Machines

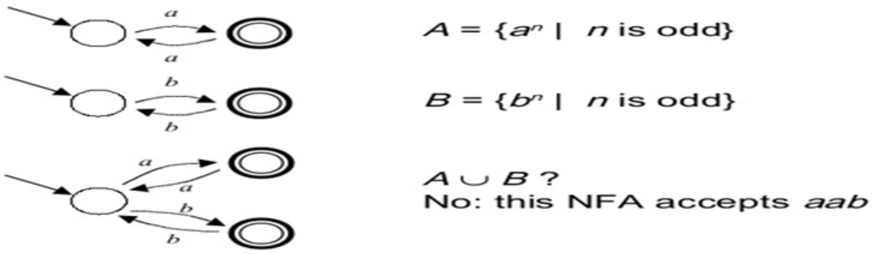
NFA with ε transitions:
An ε -NFA is a tuple (Q, Σ, δ, qo, F)
where

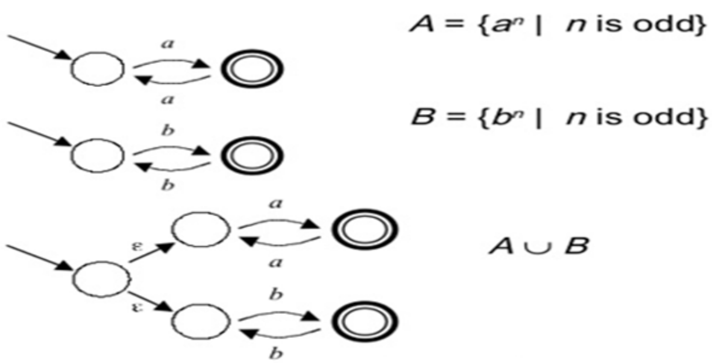
- Q is a set of states,
- Σ is the alphabet,
- δ is the transition function that maps each pair consisting of a state and a symbol in Σ ∪ { ε } to a subset of Q,
- q_0 is the initial state,
- F ⊂ Q is the set of final (or accepting) states.

Significance of ε-NFA:

It becomes very difficult or many times it seems to be impossible to draw directly NFA or DFA.

Example:





String acceptance by ε –NFA

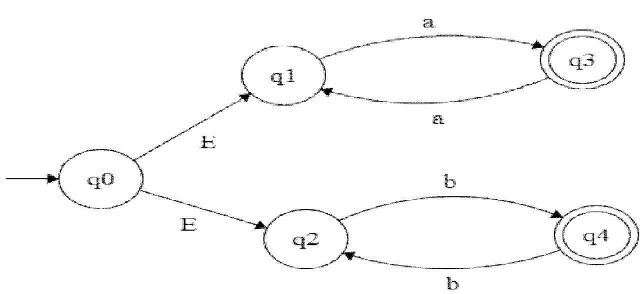


Fig:1

Transition Table:

Q/∑	a	b	ε
q_0	-	-	$\{q_1,q_2\}$
q_1	q_3	-	-
q_2	-	q_4	-
q_3	q_1	-	-
q_4	-	q_2	-

Example:

Check whether the string 'bbb' is accepted or not for the above automaton.

$$q_0 \xrightarrow{\varepsilon} q_2 \xrightarrow{b} q_4 \xrightarrow{b} q_2 \xrightarrow{b} q_4$$

$$q_0 \xrightarrow{\varepsilon} q_1$$

As q4 is the final state, the given string is accepted by the given ε –NFA.

ε –NFA to NFA Conversion:

Step 1: Find the ε-closure for all states in the given ε-NFA.

$$\hat{\delta}(q, \epsilon) = \epsilon\text{-CLOSURE}(q)$$

ε-closure (q) denotes the set of all states p such that there is a path from q to p labelled ε.

Step 2: Find the extended transition function for all states on all input symbols for the given ε-NFA.

$$\boxed{\delta'(q,a) = \varepsilon\text{-closure}(\delta(\delta'(q, \varepsilon),a))}$$

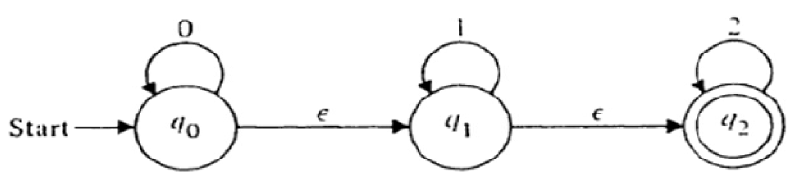
Step 3: Draw the transition table or diagram from the extended transition function (NFA)

Step 4: F is the set of final states of NFA, whose ε -closure contains the final state of ε -NFA.

Step 5: To check the equivalence of ε -NFA and NFA, the string accepted by **ε** -NFA should be accepted by NFA.

Example:

1. Convert NFA with ε-moves into an equivalent NFA without ε-moves.



Finite automaton with ϵ-moves.

Step 1: Find the ε-closure for all states in the given ε-NFA.

ε -CLOSURE (q_0) = {q_0, q_1, q_2}

ε -CLOSURE (q_1) = {q1, q_2}

ε -CLOSURE (q_2) = {q2}

Step 2: Find the extended transition function for all states on all input symbols for the given ε-NFA.

δ' (q_0,0) = ε-closure(δ (δ'(q_0, ε),0))

= ε-closure(δ {q_0, q_1, q_2},0)

= ε-closure(δ(q_0, 0) U δ (q_1,0) Uδ (q_2,0))

= ε-closure(q_0 U Ø U Ø)

= {q_0, q_1, q_2}

δ' (q_0,1) = ε-closure(δ (δ'(q_0, ε),1))

= ε-closure(δ {q_0,q_1, q_2},1)

= ε-closure(δ (q_0,1) U δ(q_1,1) U δ(q_2,1))

= ε-closure(Ø U q_1 U Ø)

={q_1,q_2}

δ' (q_0,2) = ε-closure(δ (δ'(q_0, ε),2))

= ε-closure(δ { q_0,q_1, q_2},2)

= ε-closure(δ (q_0,2) U δ (q_1,2) Uδ (q_2,2))

= ε-closure(q_2 U Ø)

={q_2}

δ' (q_1,0) = ε-closure(δ (δ'(q_1, ε),0))

= ε-closure(δ {q_1, q_2},0)

= ε-closure(δ (q_1,0) Uδ (q_2,0))

= ε-closure(Ø)

={ Ø }

δ' (q_1,1) = ε-closure(δ (δ'(q_1, ε),1))

= ε-closure(δ {q_1, q_2},1)

= ε-closure(δ (q_1,1) Uδ (q_2,1))

= ε-closure(q_1)

={q_1, q_2 }

δ (q_1,2) = ε-closure(δ (δ'(q1, ε),2))

= ε-closure(δ {q1, q2},2)

$= \varepsilon\text{-closure}(\delta (q1,2) \cup \delta (q2,2))$

$= \varepsilon\text{-closure}(q2)$

$=\{q2\}$

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

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

$= \varepsilon\text{-closure}(\emptyset)$

$=\{ \emptyset\}$

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

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

$= \varepsilon\text{-closure}(\emptyset)$

$=\{ \emptyset \}$

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

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

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

$=\{ q_2\}$

Step 3: Draw the transition table or diagram from the extended transition function (NFA)

State	Inputs		
	0	**1**	**2**
→ q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	q_2
q_1	$\emptyset$	$\{q_1, q_2\}$	q_2
$*q_2$	$\emptyset$	$\emptyset$	q_2

Step 4: F is the set of final states of NFA, whose ε -closure contains the final state of ε -NFA.

State	Inputs		
	0	**1**	**2**
→ q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	q_2
q1	$\emptyset$	$\{q_1, q_2\}$	q_2
q2	$\emptyset$	$\emptyset$	q_2

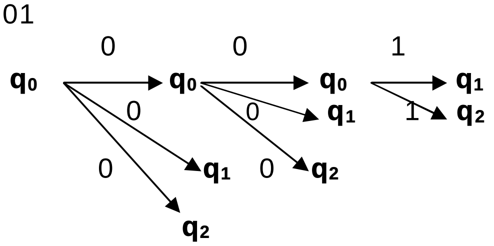
Step 5: To check the equivalence of ε -NFA and NFA, the string accepted by **ε** -NFA should be accepted by NFA.

String acceptance by ε-NFA:
Let w=001

$$q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \xrightarrow{\varepsilon} q_1 \xrightarrow{1} q_1 \xrightarrow{\varepsilon} q_2$$

As q2 is the final state, the string is accepted by the given ε-NFA.

String acceptance by NFA:
If w=001



As q1 and q2 are final states, the string is accepted by the NFA.

NFA to DFA Conversion:

Step 1: First take the starting state of NFA as the starting state of DFA.

Step 2: Apply the inputs on initial state and represent the corresponding states in the transition table.

Step 3: For each newly generated state, apply the inputs and represent the corresponding states in the transition table.

Step 4: Repeat step 3 until no more new states are generated.

Step 5: The states which contain any of the final states of the NFA are the final states of the equivalent DFA.
Step 6: Represent the transition diagram from the constructed table.
Step7: To check the equivalence of NFA and DFA, the string accepted by NFA should be accepted by DFA.
Step 8: Write the tuple representation for the obtained DFA.

Note: If the NFA has *n* states, the resulting DFA may have up to 2^n states, an exponentially larger number, which sometimes makes the construction impractical for large NFAs.

Example:

1. Construct DFA equivalent to the NFA M=($\{q_0,q_1\}$,{0,1}, δ,q_0,$\{q_1\}$)
where $\delta(q_0,0) = \{q_0,q_1\}$ $\delta(q_0,1) = \{q_1\}$ $\delta(q_1,0) = \emptyset$ $\delta(q_1,1) = \{q_0,q_1\}$

Step 1: First take the starting state of NFA as the starting state of DFA

Q/Σ	0	1
→ $[q_0]$		

Step 2: Apply the inputs on initial state and represent the corresponding states in the transition table.

Q/Σ	0	1
→ $[q_0]$	$[q_0,q_1]$	$[q_1]$

Step 3: For each newly generated state, apply the inputs and represent the corresponding states in the transition table.

Q/Σ	0	1
→ $[q_0]$	$[q_0,q_1]$	$[q_1]$
$[q_0,q_1]$	$[q_0,q_1]$	$[q_0,q_1]$
$[q_1]$	Ø	$[q_0,q_1]$

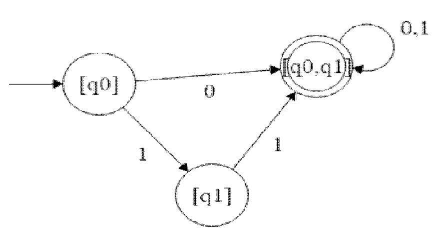
Step 4: Stop the procedure as there are no more new states being generated.

Step 5: The states which contain any of the final states of the NFA are the final states of the equivalent DFA.

q_1 is the final state in NFA. q_1 is included in the state $[q_0,q_1]$ and $[q_1]$. So $[q_0,q_1]$ and $[q_1]$ are the final states of the DFA.

Q/Σ	0	1
→ $[q_0]$	$[q_0,q_1]$	$[q_1]$
$[q_0,q_1]$	$[q_0,q_1]$	$[q_0,q_1]$
$[q_1]$	Ø	$[q_0,q_1]$

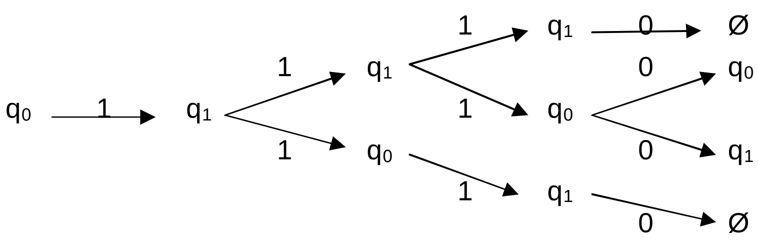
Step 6: Represent the transition diagram from the constructed table.



Step 7: To check the equivalence of NFA and DFA, the string accepted by NFA should be accepted by DFA.

Let **w=1110** be the string accepted by NFA.

Acceptability by NFA:



Acceptability by DFA:

$\delta([q_0],1110) = \delta([q_1],110)$ $\quad [q0] \xrightarrow{1} [q1] \xrightarrow{1} [q0,q1] \xrightarrow{1} [q0,q1] \xrightarrow{0} [q0,q1]$

$= \delta([q_0,q_1],10)$

$= \delta([q_0,q_1],0)$

$= [q_0,q_1]$ □ F

Step 8: Write the tuple representation from the obtained DFA.

DFA M' = $(Q,\sum, \delta,q_0,F)$
where Q = $\{[q_0], [q_0,q_1], [q_1] \}$

$\sum = \{0, 1\}$

δ - transition function

$[q_0]$ - initial state
F = $\{[q_0], [q_0,q_1]\}$

Minimization of Finite Automata:

Two states ql and q*2* are equivalent (denoted by q1 ≡ *q2)* if both δ*(q1, x)* and δ*(q2, x)* are final states. or both of them are nonfinal states for all *x* □ $\sum$*.

Two states q1 and *q2* are k-equivalent *(k* ≥ 0) if both δ*(q1, x)* and δ*(q2, x)* are final states or both nonfinal states for all strings *x* of length *k* or less. In particular, any two final states are 0-equivalent and any two nonfinal states are also 0-equivalent.

Construction of Minimum Automaton:

Step 1: (Construction of π_0)· By definition of 0-equivalence, $\pi_0 = \{Q_1^0, Q_2^0\}$ where Q_1^0 is the set of all final states and $Q_2^0 = Q - Q_1^0$.

Step 2: (Construction of π_{k+1} from π_k).

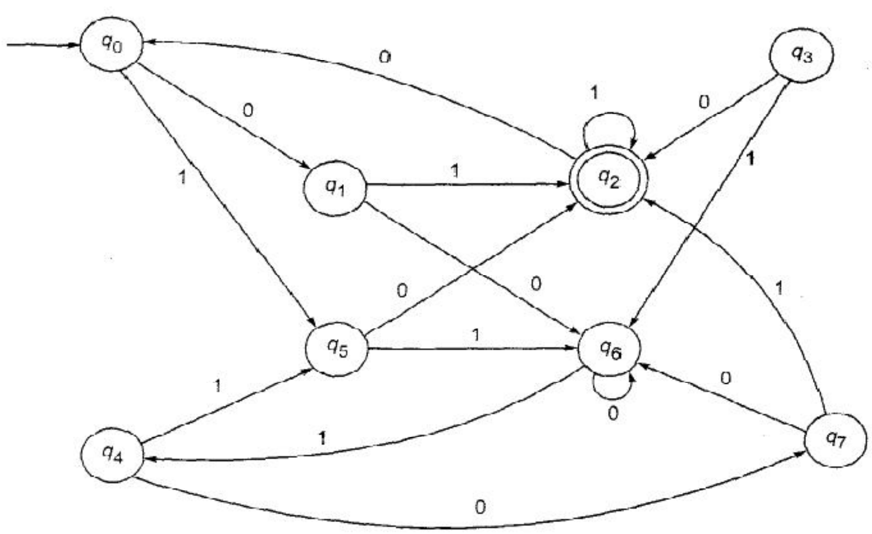
- Let Q_i^k be any subset in π_k . If q_1 and q_2 are in Q_i^k , they are (k + 1)-equivalent provided δ (q1,a) and δ(q2,a) are k-equivalent.
- Find out whether δ (q1, a) and δ (q2, a) are in the same equivalence class in π_k for every a ∑. If so q1 and q2 are (k + 1)-equivalent.
- In this way, Q_i^k is further divided into (k + 1)-equivalence classes. Repeat this for every Q_i^k in π_k to get all the elements of π_{k+1}.

Step 3: Construct π_n for n = 1, 2, ... until $\pi_n = \pi_{n+1}$.

Step 4: (Construction of minimum automaton). For the required minimum state automaton, the states are the equivalence classes obtained in step 3. i.e. the elements of π_n The state table is obtained by replacing a state q by the corresponding equivalence class [q].

Example:

Construct a minimum state automaton equivalent to the finite automaton.



Solution:

It will be easier if we construct the transition table.

State/Σ	0	1
$\rightarrow q_0$	q_1	q_5
q_1	q_6	q_2
q_2	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

Step 1: Construction of $\boldsymbol{\pi_0}$

$\boldsymbol{\pi_0} = \{Q_1^0,\ Q_2^0\}$

where $Q_1^0 = F = \{q2\}$ $\qquad$ $Q_2^0 = Q - Q_1^0$

$\therefore \boldsymbol{\pi_0} = \{\{q2\}, \{q0,q1,q3,q4,q5,q6,q7\}\}$

Step 2: The {q2} in $\boldsymbol{\pi_0}$ cannot be further partitioned. So, $Q_1^1 = \{q2\}$.

Compare q_0 with q_1, q_3,q_4,q_5,q_6 and q_7.

Consider qo and q1 □ Q_2^0.

- The entries under the 0- column corresponding to qo and q1 are q1 and q6; they lie in Q_2^0.
- The entries under the 1-column are q_5 and q_2. q2 □ Q_1^0 and q5 □ Q_2^0. Therefore qo and q1 are not 1- equivalent.

Q/∑	**0**	**1**
$\mathbf{q_0}$	$\mathbf{q_1}$	$\mathbf{q_5}$
$\mathbf{q_1}$	$\mathbf{q_6}$	$\mathbf{q_2}$

Consider q0 and q3

Q/∑	**0**	**1**
$\mathbf{q_0}$	$\mathbf{q_1}$	$\mathbf{q_5}$
$\mathbf{q_3}$	$\mathbf{q_2}$	$\mathbf{q_6}$

The entries under the 0- column corresponding to qo and q3 are q1 and q2; q1 □ Q_2^0 and q2 □ Q_1^0. The entries under the 1-column are q5 and q6; they lie in Q_2^0. Therefore qo and q3 are not 1- equivalent

Similarly, qo is not 1-equivalent to q5 and q7.

Consider q0 and q4

Q / ∑	**0**	**1**
q0	**q1**	**q5**
q4	**q7**	**q5**

- The entries under the 0- column corresponding to qo and q4 are q1 and q7; they lie in Q_2^0.
- The entries under the 1-column are q5 and q5; they lie in Q_2^0. Therefore qo and q1 are 1- equivalent.

  Similarly, qo is 1-equivalent to q6.

  {qo. q4, q6} is a subset in π_1.
  So, Q_2^1 = {q0,q4,q6}

- Repeat the construction by considering q1 and anyone of the state's q3, q5, q7. Now, q1 is not 1-equivalent to q3 or q5 but 1-equivalent to q7.
  Hence, Q_3^1 = {q1,q7}.
- The elements left over in $Q_2^0$are q3 and q5. By considering the entries under the 0-column and the 1-column, we see that q3 and q5 are 1-equivalent.
  So Q_4^1 = {q3, q5}.
  Therefore, π_1 = {{q2}. {qo, q4, q6}. {q1, q7}, {q3, q5}}

Step 3: Construct π_n for n = 1, 2, ... until $\pi_n = \pi_{n+1}$.

Calculate 2-equivalent, π_2.

π_2 = {{q2}, {qo,q4}, {q6}, {q1,q7}, {q3,q5}}

Similarly calculate 3-equivalent, π_3.

π_3 = {{q2}, {qo,q4}, {q6}, {q1,q7},{q3,q5}}

As $\pi_2 = \pi_3$, π_2 gives us the equivalence classes.

Step 4: Construction of minimum automaton.

M' = (Q',{0,1},δ',q_0',F')

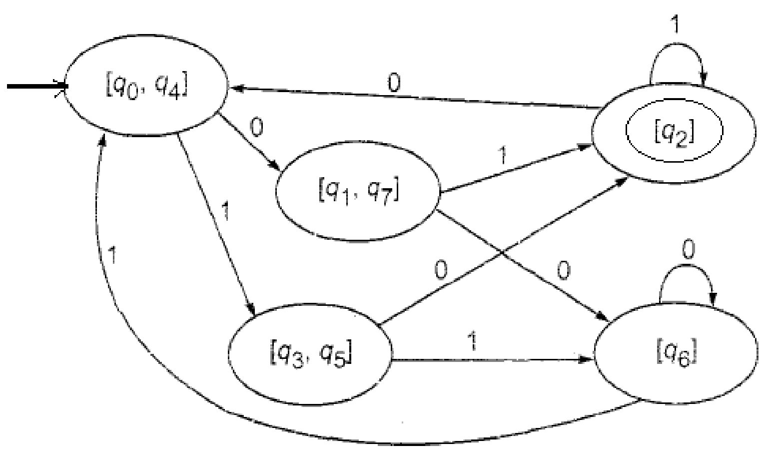
where Q' = {[q_2]. [q_0, q_4], [q_6], [q_1, q_7], [q_3, q_5]}

q_0' = [q0, q4]

F' = [q2]

δ' is given by

State/Σ	0	1
[q_0, q_4]	[q_1, q_7]	[q_3, q_5]
[q_1, q_7]	[q_6]	[q_2]
[q_2]	[q_0, q_4]	[q_2]
[q_3, q_5]	[q_2]	[q_6]
[q_6]	[q_6]	[q_0, q_4]



Equivalence between two FSM's:

Let M and M' be two FSM's over ∑ .We construct a comparison table consisting of n+1 columns where n is the number of input symbols.

Step 1: 1st column consisting of a pair of states of form (q, q') where q belongs to M and q' belongs M'.

Step 2: If (q, q') appears in the same row of 1st column then the corresponding entry in a column (a belongs to ∑) is (r,r') where (r,r') are pair from q and q' on a.

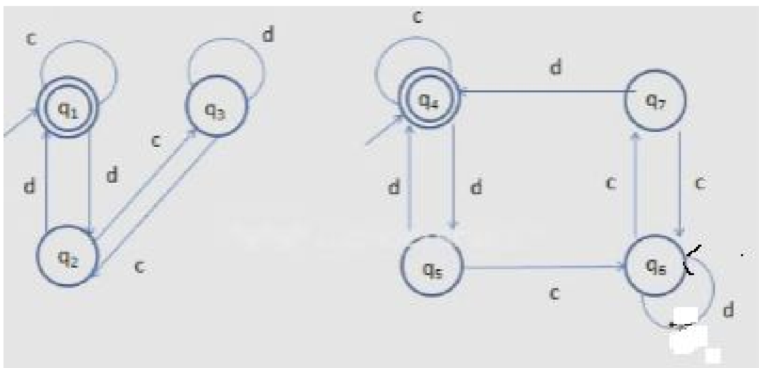
Step 3: A table is constructed by starting with a pair of initial states q_0, q_0' of M and M'. We complete construction by considering the pairs in 2nd and subsequent columns which are not in the 1st column.

(i) if we reach a pair (q,q') such that q is final states of M and q' is non-final state of M' i.e. terminate contruction and conclude that M and M' are not equivalent.

(ii) if construction is terminated when no new element appears in 2nd and subsequent columns which are not in 1st column. Conclude that M and M' are equivalent.

Example:

Check whether the given two finite automata's are equivalent or not.



Solution:

q_1 is initial state of M1 and q_4 is initial state of M2 ,make them a pair and place it in 1st row of the transition table.

Comparison table

Q/∑	c	d
(q_1,q_4)	**(q_1,q_4)**	**(q_2,q_5)**
(q_2,q_5)	**(q_3,q_4)**	

Here q3 is non-final state and q4 is final state.

Therefore, we stop constructing comparison table and conclude that the two given Finite Automata's are not equivalent.

Moore Machine

A Moore machine is a six tuple (Q, ∑, ∆, δ, q_0, λ)
where

- Q is a set of states,
- Σ is the alphabet,
- δ is the transition function that maps each pair consisting of a state and a symbol in Σ to Q i.e. .Q X Σ -> Q
- q0 is the initial state,
- Δ is output alphabet
- λ is a mapping from Q to Δ giving the output associated with each state

Note: For a Moore machine if the input string is of length n, the output string is of length n + 1. The first output is λ (qo) for all output strings.

Mealy Machine

A Mealy machine is a six tuple (Q, ∑, Δ, δ, q_0, λ)

where

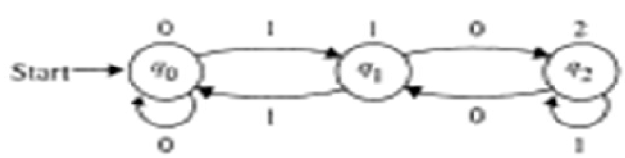
- Q is a set of states,
- Σ is the alphabet,
- δ is the transition function that maps each pair consisting of a state and a symbol in Σ to Q i.e. .Q X Σ -> Q
- Δ is output alphabet
- q0 is the initial state,
- **λ** maps Q x ∑ **to** Δ **i.e.,** λ(q,a) gives the output associated with the transition from state q on input a

Note: In the case of a Mealy machine if the input string is of length n , the output string is also of the same length n.

Example:

- The given transition diagram is moore machine because each state is associated with output.
- In the below diagram q_0 is representing 0 output, q_1 is is representing 1 output and q_2 is representing 2 output.

  λ (q_0) =0 λ (q_1)=1 λ (q_2)=2



w=011 the output is **0010**

q_0 $\xrightarrow{0}$ q_0 $\xrightarrow{1}$ q_1 $\xrightarrow{1}$ q_0 Transitions

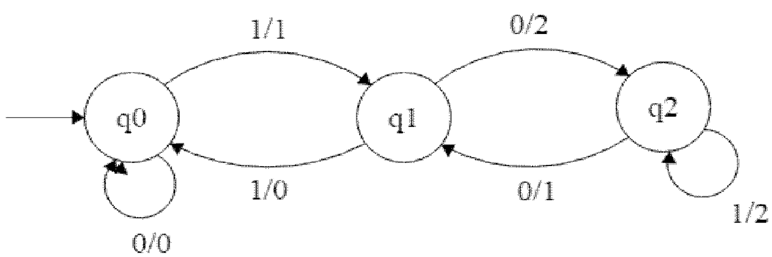
0 0 1 0 Outputs

Example:

- The given transition diagram is mealy machine because output depends on present state and present input.
- In the below diagram

$\lambda\ (q_0,0) = 0$ $\lambda\ (q_1,0) = 2$ $\lambda\ (q_2,0) = 0$

$\lambda\ (q_0,1) = 1$ $\lambda\ (q_1,1) = 0$ $\lambda\ (q_2,1) = 2$



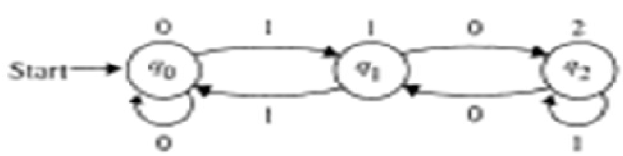
w=011 the output is **010**

q_0 $\xrightarrow{0}$ q_0 $\xrightarrow{1}$ q_1 $\xrightarrow{1}$ q_0 Transitions

0 1 0 Outputs

Example:

1. Design Moore machine to determine the residue mod 3 for each binary string treated as a binary integer.



Moore machine calculating residue mod 3

Moore Table

Present State	**Next State**		**Output**
	0	**1**	

→ q_0	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_1	q_2	2

Tuple Representation:

Q=$\{q_0,q_1,q_2\}$ $\Delta=\{0,1,2\}$ $\Sigma=\{0,1\}$

$\mathbf{q_0}=\{q_0\}$

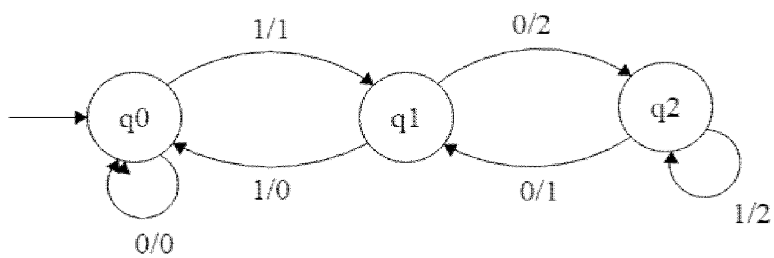
λ :$\lambda(q_0)=0$ **δ:** $\delta(q_0,0) = q_0$ $\delta(q_0,1) = q_1$

$\lambda(q_1)=1$ $\delta(q_1,0) = q_2$ $\delta(q_1,1) = q_0$

$\lambda(q_2)=2$ $\delta(q_2,0) = q_1$ $\delta(q_2,1) = q_2$

Example:

1. Design Mealy machine to determine the residue mod 3 for each binary string treated as a binary integer.



Mealy Table:

Present State	Next State		Next State	
	0	Output	1	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

Tuple Representation:

Q=$\{q_0,q_1,q_2\}$ $\Delta=\{0,1,2\}$ $\Sigma=\{0,1\}$

$\mathbf{q_0}=\{q_0\}$

λ: $\lambda(q_0,0)=0$ **δ:** $\delta(q_0,0) = q_0$ $\delta(q_0,1) = q_1$

λ (q0,1)=1 $\delta(q_{1},0) = q_2$ $\delta(q_{1},1) = q_0$

λ $(q_{1},0)$=2 $\delta(q_{2},0) = q_1$ $\delta(q_{2},1) = q_2$

λ $(q_{1},1)$=0

λ $(q_{2},0)$=1

λ $(q_{2},1)$=2

Moore to Mealy Conversion:

If M_1= (Q,∑,Δ,δ,q_0, λ) is a Moore machine, then there is a Mealy machine M_2 equivalent to M_I.

Procedure:

- Let M2 = (Q,∑,Δ,δ,q_0, λ') and define λ' (q, a) to be λ (δ (q, a)) for all states q and input symbols a.
- Then M_I and M_2 enter the same sequence of states on the same input, and with each transition M_2 emits the output that M_I associates with the state entered.

Example:

Construct a Mealy Machine which is equivalent to the Moore machine given by table below.

Present State	Next State		Output
	0	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:

λ' (q, a) to be λ(δ (q, a))

$\lambda'(q_0,0) = \lambda(\delta(q_0, 0))$ $\lambda'(q_0,1) = \lambda(\delta(q_0, 1))$

$= \lambda(q_3)$ $= \lambda(q_1)$

$=0$ $=1$

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

$= \lambda(q_1)$ $= \lambda(q_2)$

$=1$ $=0$

$\lambda'(q_2,0) = \lambda(\delta(q_2, 0))$ $\lambda'(q_1,1) = \lambda(\delta(q_2, 1))$

$= \lambda(q_2)$ $= \lambda(q_3)$

$=0$ $=0$

$\lambda'(q_3,0) = \lambda(\delta(q_3, 0))$ $\lambda'(q_3,1) = \lambda(\delta(q_3, 1))$

$= \lambda(q_3)$ $= \lambda(q_0)$

$=0$ $=0$

Mealy Table:

Present State	Next State		Next State	
	0	output	1	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

Mealy to Moore Conversion:

If $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ is a Mealy machine, then there is a Moore machine M_2 equivalent to M_l.

Procedure:

- Determine the number of different output associated with qi in the next state column.
- We split qi into different states according to different output associated with it

  For example: q_2 is associated with two different outputs 0 and 1, so we split q_2 into q_{20} and q_{21}.

Example:

Construct Moore machine for the given mealy machine.

Present State	Next State			
	a = 0		a = 1	
	State	Output	State	Output
-> q0	q3	0	q1	1
q1	q0	1	q3	0
q2	q2	1	q2	0
q3	q1	0	q0	1

Solution:

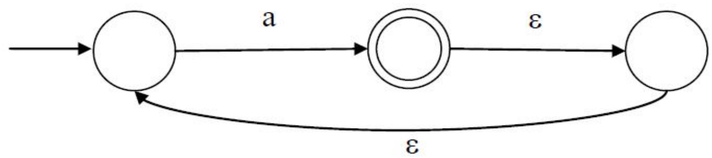
- We get two states (q1 and q2) that are associated with different outputs (0 and 1). so we split both states into q_{10} , q_{11} and q_{20}, q_{21}.
- Whole row of q_1 is copied to q_{10}, q_{11} and whole row of q_2 is copied to q_{20} and q_{21} of the sample transition table of mealy machine.
- The outputs of the next state columns of q_1 and q_2 are depend on the previous output. For ex. in the first row, q_1 becomes q_{11} because the out of q_1 is 1 in the fourth row, q_2 becomes q_{21} because the output of the q_2 is 1 and in the subsequent column q_2 becomes q_{20} because the output of q_2 in that column was 0. and so on

Present State	Next State		Output
	a = 0	a = 1	
-> q0	q3	q11	1
q10	q0	q3	0
q11	q0	q3	1
q20	q21	q20	0
q21	q21	q20	1
q3	q10	q0	0

UNIT-II
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. What is the complement of the language accepted by the NFA shown below? [ ]



(A) ∅ (B) {ε} (C) a^* (D) {a , ε}

2. NFA with ε can increase the processing time of NFA **[True/False]**
3. ______________ of a state is the set of states that can be reached by ε-transitions.
4. The number of states in DFA is _________ the number of states in NFA for the same language. [ ]
   (A) greater than (B) less than (C) equal to (D) none
5. Given a Non-deterministic Finite Automaton (NFA) with states p and r as initial states and final states respectively and transition table as given below:

	a	b
p	–	q
q	r	s
r	r	s
s	r	s

   The minimum number of states required in Deterministic Finite Automaton (DFA) equivalent to NFA is [ ]
   (A) 5 (B) 4 (C) 3 (D) 2
6. The output in __________ machine is associated with transition. [ ]
   (A) Moore (B) Mealy (C) both (D) DFA
7. The two states q_1 and q_2 are said to be ________________ if both $\delta(q_1,a)$ and (q_2,a) reach final states or both of them reach non final states for all $a \in \sum$.
8. For a Moore machine if the input string is of length n, the output string is of length n + 1. **[True/False]**
9. In a Mealy machine if the input string is of length n, the output string is of length __________.
   (A) n (B) n+1 (C) 2n (D) n+2
10. Choose incorrect statement. [ ]
   (A) Moore and Mealy machines are FSM's with output capability.
   (B) Any given Moore machine has an equivalent Mealy Machine.
   (C) Any given Mealy machine has an equivalent Moore Machine.

(D) Moore Machine in not a FSM.

11. All Moore Machine have an equivalent Finite Automata. **[True / False]**

12. Which of the following statement is true? [ ]
    (A) A Mealy machine has no terminating state
    (B) A Moore machine has no terminating state
    (C) Converting from Mealy into Moore machine and vice versa is possible
    (D) All of these

13. The output alphabet in Moore machine can be represented formally as
    (A) Δ (B) $\sum$ (C) δ (D) λ [ ]

14. Consider the table [ ]

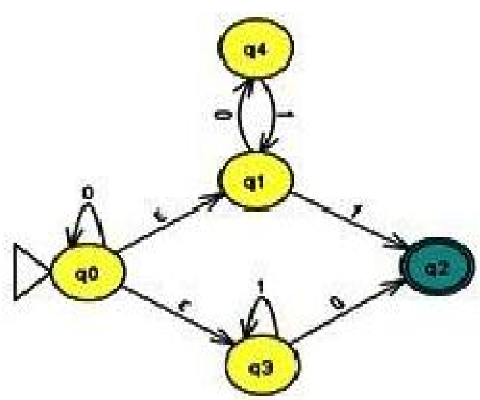
Present State	**Next State**			
	0		**1**	
	state	**output**	**state**	**output**
q_0	q_0	0	q_1	1
q_1	q_2	2	q_0	0
q_2	q_1	1	q_2	2

If the initial state is q_0. What is the output sequence for the string 101?
(A) 0012 (B) 122 (C) 112 (D) 0122

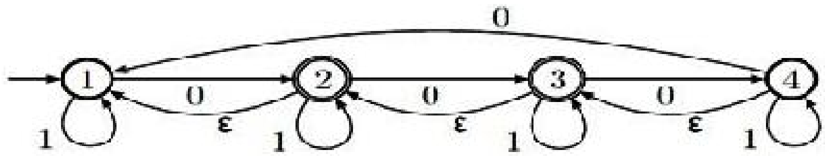
SECTION-B

SUBJECTIVE QUESTIONS

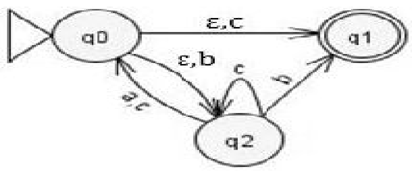
1. Consider the following finite automaton with -moves Obtain equivalent automaton without -moves.



2. Construct NFA for the set of strings in (0+1)* such that some two 0's are separated by a string whose length is 4i, for some i≥0.
3. Construct a NFA without ∈ for the following NFA with ∈.



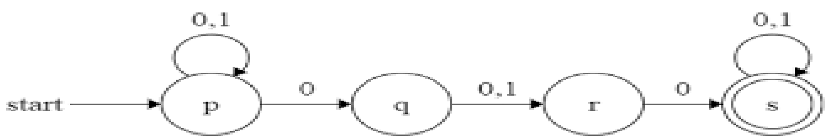
4. Define ε-closure. Find the ε-closures of the each state in the following ε-NFA.



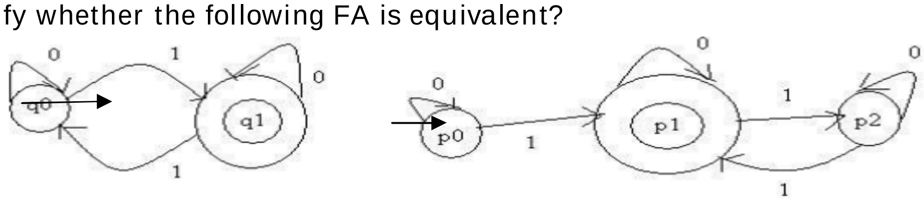
5. Construct an equivalent DFA for a NDFA M = ({q_1, q_2, q_3}, q_1, q_3) where δ is given by

$\delta(q_1,0)=\{q_2,q_3\}$, $\delta(q_1,1)=\{q_1\}$,
$\delta(q_2,0)=\{q_1,q_2\}$, $\delta(q_2,1)=\emptyset$
$\delta(q_3,0)=\{q_2\}$, $\delta(q_3,1)=\{q_1,q_2\}$

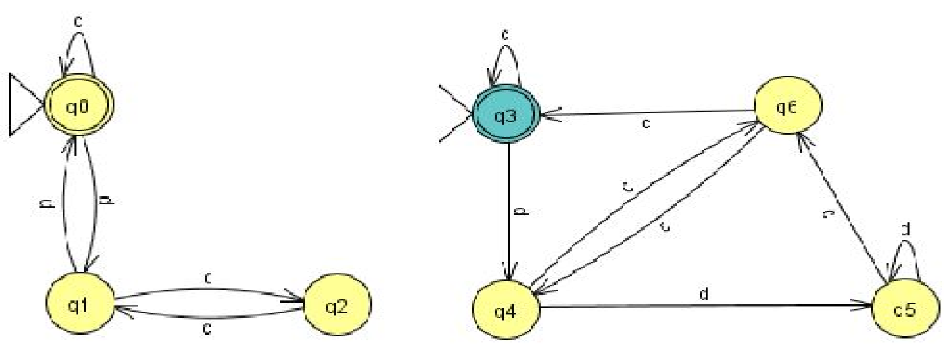
6. Construct an equivalent DFA for the following NFA



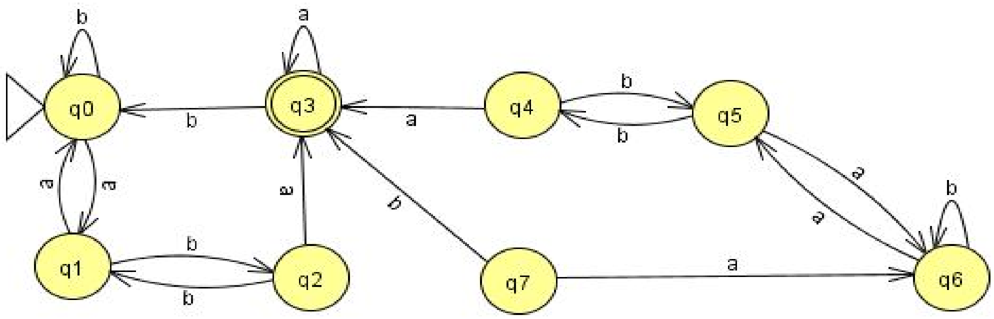
7. Verify whether the following FA is equivalent?



8. Find the equivalence between M1 & M2



9. Construct the minimum state automaton equivalent to the transition diagram



10. Construct a minimum state automaton equivalent to a given automaton *M* whose transition table is defined by table

State	Input	
	a	*b*
$\to q_0$	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

11. Explain about the finite automata with outputs in detail.

12. Construct a Mealy machine which is equivalent to the Moore machine defined by 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

13. Construct a Moore machine equivalent to the Mealy machine *M* defined by

Present state	Next state			
	$a = 0$		$a = 1$	
	state	output	state	output
$\to q_1$	q_1	1	q_2	0
q_2	q_4	1	q_4	1
q_3	q_2	1	q_3	1
q_4	q_3	0	q_1	1

14. Design a Mealy machine that uses its states to remember the last symbol read and emits output 'y' whenever current input matches to previous one, and emits n otherwise
15. Design a Moore machine to determine the residue mod 4 for each binary string treated as integers.
16. Construct a Moore machine that takes set of all strings over {a,b} as input and prints '1' as output for every occurrence of 'ab' as a substring.
17. Construct a Mealy machine which can output EVEN or ODD according as the total number of 1's encountered is even or odd. The input symbols are 0 and 1.
18. Give Mealy and Moore machines for the following process: For input from (0+1)*, if the input ends in 101, output A; If the input ends in 110 output B; otherwise output C.

SECTION-C
QUESTIONS AT THE LEVEL OF GATE

1. Let denote the transition function and $\hat{\delta}$ denote the extended transition function of the ε-NFA whose transition table is given below:

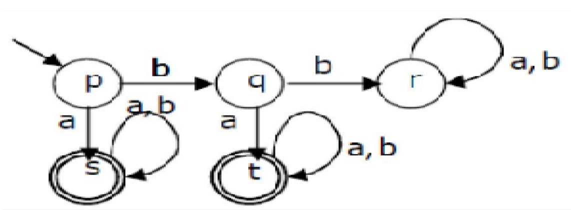
δ	ϵ	a	b
$\rightarrow q_0$	$\{q_2\}$	$\{q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\{q_2\}$	$\{q_3\}$
q_2	$\{q_0\}$	$\emptyset$	$\emptyset$
q_3	$\emptyset$	$\emptyset$	$\{q_2\}$

[GATE 2017(Set 2)]

Then $\hat{\delta}$ (q2,aba) is [ ]

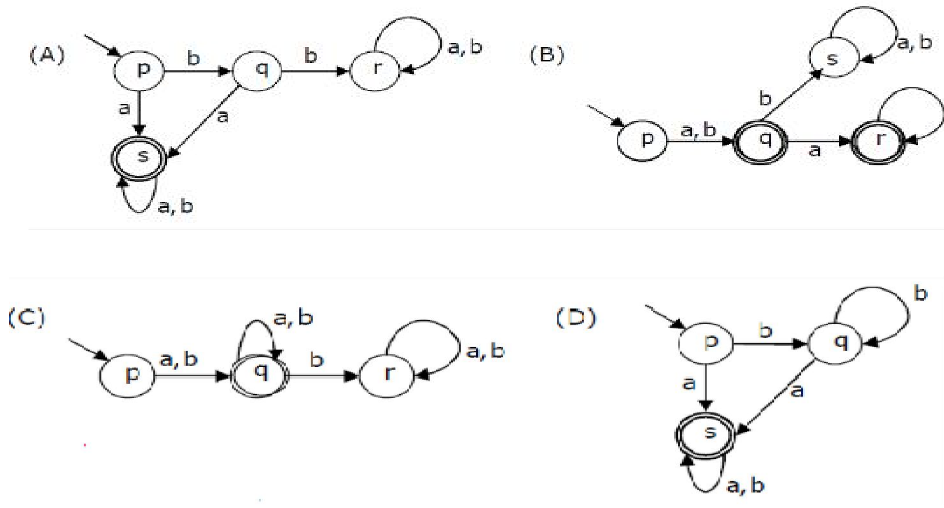
(A) ∅ (B) {q0,q1,q3} (C) {q0,q1,q2} (D) {q0,q2,q3}

2. A deterministic finite automation (DFA)D with alphabet ∑={a,b} is given below

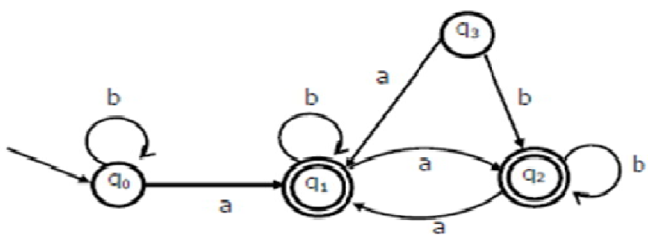


[GATE 2011]

Which of the following finite state machines is a valid minimal DFA which accepts the same language as D? [ ]



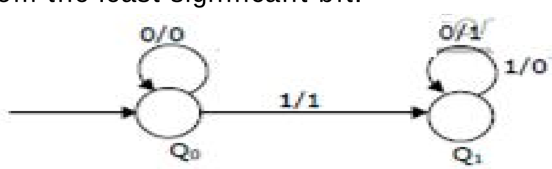
3. Consider the following finite state automaton [ ]



The minimum state automaton equivalent to the above FSA has the following number of states **[GATE 2007]**

(A) 1 (B)2 (C) 3 (D)4

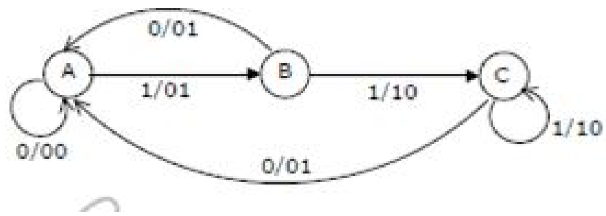
4. The following diagram represents a finite state machine which takes as input a binary number from the least significant bit. **[GATE 2005]**



Which one of the following is true? [ ]

(A) It computes 1's complement of the input number
(B) It computes 2's complement of the input number
(C) It increments the input number
(D) It decrements the input number

5. The finite state machine described by the following state diagram with A as starting state, where an arc label is x/y and x stands for 1-bit input and y stands for 2-bit output [ ] **[GATE 2002]**



(A) Outputs the sum of the present and the previous bits of the input
(B) Outputs 01 whenever the input sequence contains 11
(C) Outputs 00 whenever the input sequence contains 10
(D) None of the above

6. Given an arbitrary non-deterministic finite automaton(NFA) with N states, the maximum number of states in an equivalent minimized DFA is atleast
(A) N^2 (B) 2^N (C) 2N (D) N! [ ] **[GATE 2001]**

UNIT-III

Objective:

To familiarize how to employ regular expressions.

Syllabus:

Regular sets, regular expressions, identity rules, construction of finite Automata for a given regular expressions and its inter conversion, pumping lemma of regular sets, closure properties of regular sets (proofs not required),applications of regular languages.

Learning Outcomes:

Students will be able to:

- understand the regular sets and how to represent the regular expressions.
- construct finite Automata for a given regular expression and viceversa.
- list closure properties of regular languages.
- understand the different applications of regular languages.

Learning Material

Regular set:

A language is a regular set (or just regular) if it is the set accepted by some finite automaton.

Example:

L= {0, 1, 10, 00, 01, 11, 000, 101, ...} is a regular set representing any no of 0's and any no of 1's.

Regular expression:

The languages accepted by finite automata are easily described by simple expressions called regular expressions.

Let Σ be an alphabet. The regular expressions over Σ and the sets that they denote are defined recursively as follows.

1) Ø is a regular expression and denotes the empty set.

2) ε is a regular expression and denotes the set { ε }.

3) For each a in Σ, a is a regular expression and denotes the set {a}.

4) If r and s are regular expressions denoting the languages R and S, respectively, then

(r + s), (rs), and (r*) are regular expressions that denote the sets R U S, RS, and R*, respectively.

Some Examples on Regular expressions

1.Write regular expressions for each of the following languages over ∑={0, 1}.

a) The set representing {00}.

00

b) The set representing all strings of 0's and 1's.

(0+1)*

c) The set of all strings representing with at least two consecutive 0’s.

(0 + 1)*00(0 + 1)*

d) The set of all strings ending in 011.

(0 + 1)*011

e) The set of all strings representing any number of 0's followed by any number of 1’s followed by any number of 2's.

0*1*2*

f) The set of all strings starting with 011.

011 (0 + 1)*

2.Write regular expressions for each of the following languages over ∑={a, b}.

a) The set of all strings ending with either a or bb.

(a+b)* (a + bb)

b) The set of strings consisting of even no. of a’s followed by odd no. of b’s.

(aa)*(bb)*b

c) The set of strings representing even number of a’s.

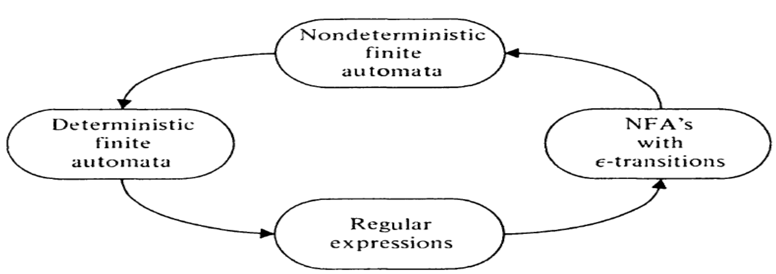
(b* a b* a b*) * + b*

Identity Rules Related to Regular Expressions

Given r, s and t are regular expressions, the following identities hold:

- $\emptyset^* = \varepsilon$
- $\varepsilon^* = \varepsilon$
- $r^+ = rr^* = r^*r$
- $r^*r^* = r^*$
- $(r^*)^* = r^*$
- $r + s = s + r$
- $(r + s) + t = r + (s + t)$
- $(rs)t = r(st)$
- $r(s + t) = rs + rt$
- $(r + s)t = rt + st$
- $(\varepsilon + r)^* = r^*$
- $(r + s)^* = (r^*s^*)^* = (r^* + s^*)^* = (r+s^*)^*$
- $r + \emptyset = \emptyset + r = r$
- $r\varepsilon = \varepsilon r = r$
- $\emptyset L = L\emptyset = \emptyset$
- $r + r = r$
- $\varepsilon + rr^* = \varepsilon + r^*r = r^*$

Construction of Finite automata for a given regular expression



Equivalence of Finite Automata and Regular Expressions

- The languages accepted by finite automata are precisely the languages denoted by regular expressions.
- For every regular expression there is an equivalent NFA with ε -transitions.
- For every DFA there is a regular expression denoting its language.

Let r be a regular expression. Then there exists an NFA with ε -transitions that accept L(r).

Zero operators:

The expression r must be ε, Ø, or a for some a in Σ. The NFA's for zero operators are

(a) $r = \epsilon$ (b) $r = \varnothing$ (c) $r = \mathbf{a}$

One or more operators:

Let r have i operators. There are three cases depending on the form of r.

Case 1: Union (r = rl + r2.)

There are NFA's M1 = **(Q1, ∑1, δ1, q1, {f1})** and M2=**(Q2, ∑2, δ2, q2, {f2})** with L(M1) = L(r1) and L(M2) = L(r2).

Construct

M = (Q1 ∪ Q2 ∪ {q0, f0}, ∑**1** ∪ ∑**2**, **δ, q0, {f0})** where **δ is defined by**

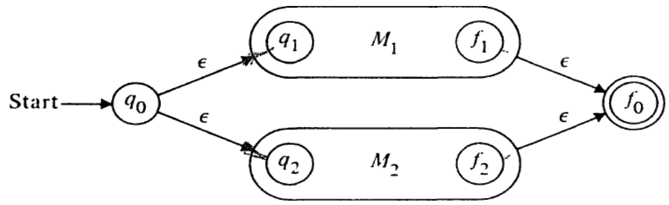
i) **δ (q0,** ε) = {q1,q2}

ii) **δ (q,** a) = **δ**1(q ,a) for q in Q1-{f1} and a in ∑**1** ∪ { ε }

iii) **δ (q,** a) = **δ**2(q ,a) for q in Q2-{f2} and a in ∑**2** ∪ { ε }

iv) **δ (f1,** ε) = **δ**1(f2, ε) = { f0 }

L(M) = L(M1) ∪ L(M2)



<u>Case 2:</u> Concatenation (r = r1 r2).

Let M1 and M2 be as in Case 1 and construct M = (Q1 ∪ Q2, ∑**1** ∪ ∑**2**, **δ, q1, {f2})**

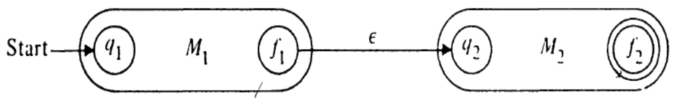
where **δ is defined by**

i) δ (q, a) = **δ**1(q ,a) for q in Q1-{f1} and a in ∑**1** ∪ { ε }

ii) δ (f1, ε) = {q2}

iii)δ (q, a) = **δ**2(q ,a) for q in Q2 and a in ∑**2** ∪ { ε }

L(M) ={xy | x is in L(M1) and y is in L(M2)} and L(M) = L(M1)L(M2)



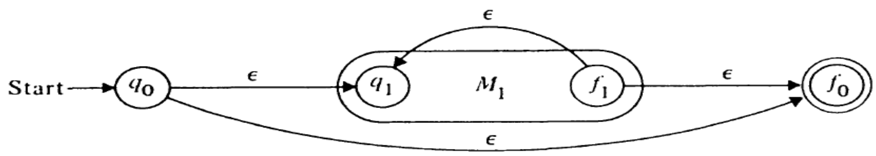
<u>Case 3:</u> Closure (r = r1*)

Let M1 = **(Q1, ∑1, δ1, q1, {f1}) and L(M1) = r1.**

Construct M = (Q1 ∪ {q0,f0}, ∑**1** , **δ, q0, {f0}),** where **δ is defined by**

i) δ (q0, ε) = **δ (f1,** ε) = {q1,f0}

ii) δ (q, a) = **δ**1(q ,a) for q in Q1-{f1} and a in ∑**1** ∪ { ε }

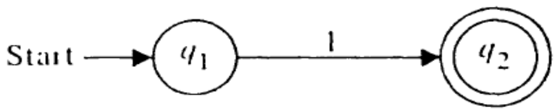


Example:

1. Construct an NFA for the regular expression 01*+1

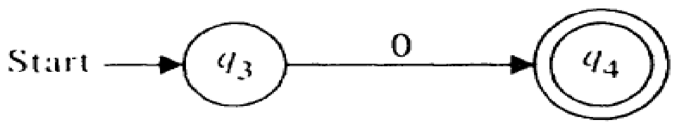
Regular expression is of the form r1 + r2, where r1 =01* and r2 = 1.

The automaton for r_2 is



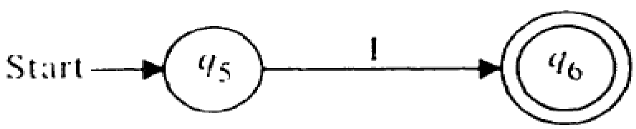
Express r_1 as r_3 and r_4, where r_3=0 and r_4= 1^*

The automaton for r_3 is

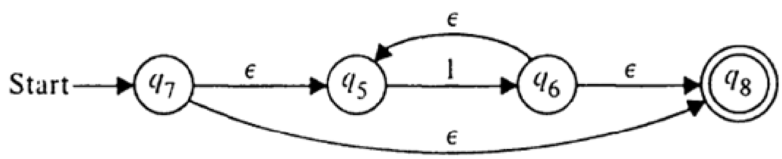


r_4 is r_5^* where r_5=1

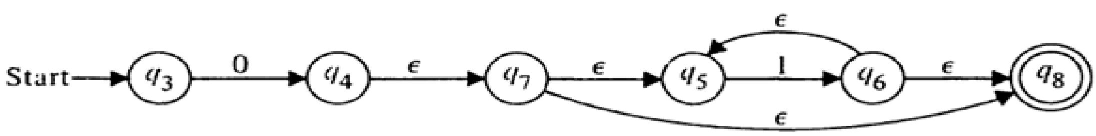
The NFA for r_5 is



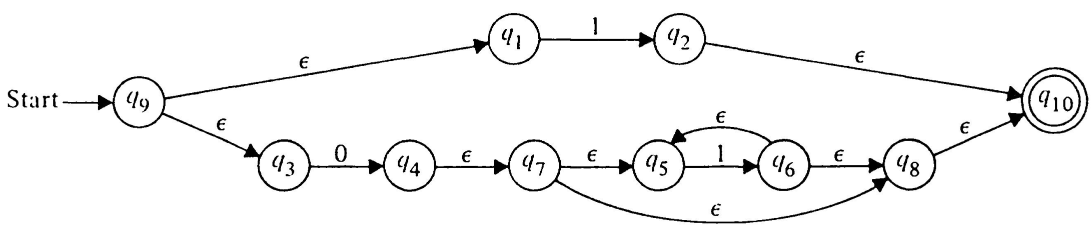
To construct an NFA for r4 = r_5^* use the construction of closure. The resulting NFA for r4 is



Then, for r1 = r3 r4 use the construction of concatenation.



Finally, use the construction of union to find the NFA for r = r1 + r2



Construction of regular expressions for the given finite Automata:

Arden's Theorem

Let P and Q be two regular expressions over Σ,and if P does not contain epsilon, then R=Q+RP has a unique solution R=QP*.

Procedure:

Assume the given finite automata should not contain any epsilons.

Step 1: Find the reachability for each and every state in given Finite automata.

Reachability of a state is the set of states whose edges enter into that state.

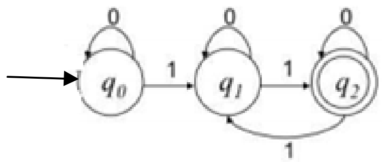
Step 2: For the initial state of finite automata ,add epsilon to the reachability equation.

Step 3: Solve the equations by using Arden's Theorem.

Step 4: Substitute the results of each state equation into the final state equation,to get the regular expression for the given DFA.

Example:

1. Construct regular expression for the given finite automaton.



The given Finite Automata is not having any ε's(epsilons).

Step 1: Find the reachability for each and every state in given Finite automata.

Reachability of a state is the set of states whose edges enter into that state.

$q_0 = q_0\,0$ ——— **1**

$q_1 = q_0\,1 + q_1 0 + q_2\,1$ ——— **2**

$q_2 = q_1\,1 + q_2 0$ ——— **3**

Step 2: For the initial state of finite automata, add epsilon to the reachability equation.

$q_0 = q_0\,0 + \epsilon$

Step 3: Solve the equations by using Arden's Theorem.

After applying arden's theorem for equation 3

$q_2 = q_1\,10^*$ ——— **4**

Substitute equation 4 in equation 2

$q_1 = q_0\,1 + q_1 0 + q_1\,10^*$

$q_1 = q_0\,1 + q_1(0+10^*)$ ——— **5**

Apply arden's theorem on equation 5

$q_1 = q_0 1\,(0+10^*)^*$ ——— **6**

Apply arden's theorem on equation 1

$q_0 = q_0\,0 + \epsilon$

$q_0 = \epsilon\,0^*$ ——— **7**

Substitute equation 7 in equation 6

$q_1 = \epsilon\,0^*\,1\,(0+10^*)^*$ ——— **8**

Step 4: Substitute the results of each state equation into the final state equation, to get the regular expression for the given DFA.

q_2= □ 0* 1 (0+10*)* 10*

Therefore, the regular expression for the given DFA is **0* 1 (0+10*)* 10*.**

Pumping Lemma for Regular Sets:

- Pumping lemma, which is a powerful tool for proving certain languages non-regular.
- It is also useful in the development of algorithms to answer certain questions concerning finite automata, such as whether the language accepted by a given FA is finite or infinite.

Lemma

Let L be a regular set. Then there is a constant n such that if z is any word in L, and |z| > n, we may write z=uvw in such a way that |uv| ≤ n, v ≥ 1, and for all i> 0, uv^iw is in L. Furthermore, n is no greater than the number of states of the smallest FA accepting L.

Example:

The set L = {0^{i^2} | i is an integer, i ≥ 1], which consists of all strings of 0's whose length is a perfect square, is not regular.

Assume L is regular and let n be the integer in the pumping lemma.

Let z = 0^{n2}.

By the pumping lemma, 0^{n2}may be written as uvw, where 1 ≤ |v| ≤ n and uv^iw is in L for all i. Let i = 2, n^2 < |uv^2w| < n^2 + n < $(n+1)^2$.

That is, the length of uv^2w lies properly between n^2 and $(n+1)^2$, and is thus not a perfect square.

Thus uv^2w is not in L, a contradiction.

We conclude that L is not regular.

Closure Properties of Regular Sets:

- The regular sets are closed under union, concatenation, and Kleene closure.
- The class of regular sets is closed under complementation. That is, if L is a regular set and L ⊆ ∑*, then ∑* - L is a regular set.
- The regular sets are closed under intersection.
- The class of regular sets is closed under substitution.

- The class of regular sets is closed under homomorphism and inverse homomorphism.
- The class of regular sets is closed under quotient with arbitrary sets.

UNIT-III
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. The languages accepted by finite automata are easily described by simple expressions called___________.
2. A language is a _______________ if it is the set accepted by some finite automaton.
3. What is the solution for equation R=Q+RP (if P and Q are RE and P does not contain ε)? [ ]

   (a) $R=QP^*$ (b) R=QP (c) $R=PQ^*$ (d) $R=P^*Q^*$

4. ∅ + R =____________.
5. ∅* =_________________.
6. ε*=______________.
7. ε + r r*=r* [True / False]
8. Pumping lemma is generally used for proving [ ]

   (a) a given grammar is regular

   (b) a given grammar is not regular

   (c) whether two given regular expressions are equivalent

   (d) none of the above

9. Regular sets are closed under [ ]

   (a) Union (b) concatenation

   (c) Kleene closure (d) All of the above

10. a + b denotes the set___________________. [ ]

   (a) {a, b} (b) {ab} (c) { a } (d) {b}

11. The set of all strings of {0, 1} having exactly two 0's is [ ]

   (a) $1^*01^*01^*$ (b) {(0+1) *} (c) $\{11+0\}^*$ (d) $\{00+11\}^*$

12. The regular expression to represent all strings with length atmost 2 over {a,b} is_______.

   (a) ε (b) ε+(a+b)+(a+b).(a+b) (c) (a+b) (d) (a+b).(a+b)

13. Which one of the following languages over the alphabet {0 ,1} is described by the regular expression: (0+1)*0(0+1)*0(0+1)*? [ ]

(a) The set of all strings containing the substring 00.

(b) The set of all strings containing atmost two 0's.

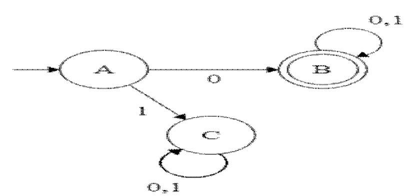
(c) The set of all strings containing atleast two 0's.

(d) The set of all strings that begin and end with either 0 or 1.

14. Consider the languages L1 = □ and L2 = {0}. Which one of the following represents L1 L2 * + L1* [ ]

(A) { □ } (B) Ø (C) 0* (D) { □ ,0}

15. What is the regular expression for the given DFA?

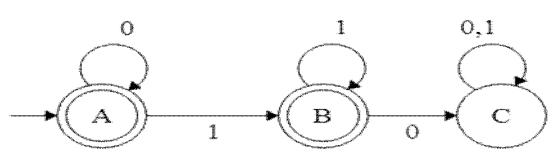


(a) (0+1)* (b) 0(0+1)* (c) 0 (d) (0+1)*0

16. Which of the following languages are not regular? [ ]

(a) L= a^n | n>=1 (b) L=$a^n b^m$ | n,m>=1 (c) $a^n b^n$ | n>=1 (d) a^{2n} | n>=0

17. What is the regular expression for the given DFA? [ ]



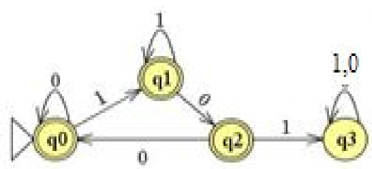
(a) 0^*1^+ (b) 0*1* (c) 1*0* (d) 1^*0^+

SECTION-B

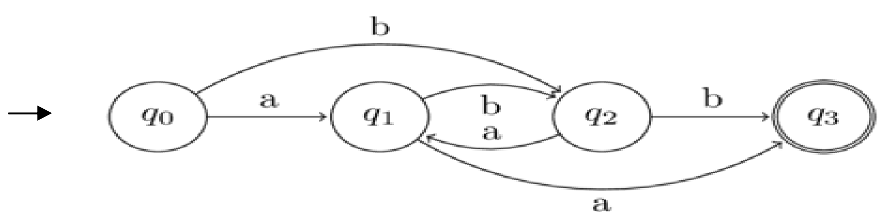
SUBJECTIVE QUESTIONS

1. Define regular set and regular expression.
2. State Arden's Theorem.
3. List the closure properties of Regular Languages.
4. Explain pumping lemma for regular languages with an example.
5. Write the regular expression for all strings ending in 1101 over the alphabet {0, 1}.
6. Design a ε-NFA for the regular expression a*bc | ab* |c*.

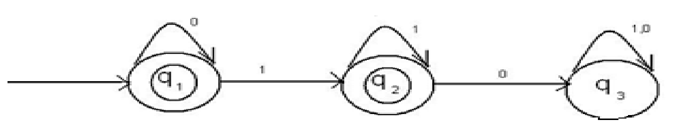
7. Construct NFA with ε-moves for the regular expression 10+(0+11)0*1
8. Construct Finite automata for the regular expression 1(01+10)*00.
9. What is pumping lemma for regular sets? Show that the language $L=\{a^n b^n c^n \mid n>=1\}$ is not regular.
10. Construct finite automation to accept the regular expression (0+1)*(00+11)(0+ 1)*.
11. Using pumping lemma, show the following language is not regular:
    L= {w □ {0,1}* | the number of 0's in w is a perfect square}
12. Construct the regular expression for the following DFA.



13. Construct regular expression for the following DFA.



14. Construct regular expression for the given DFA.



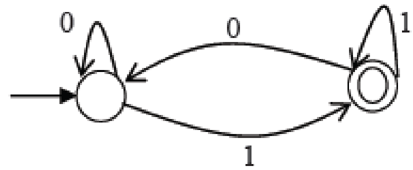
SECTION-C

QUESTIONS AT THE LEVEL OF GATE

1. The number of states in the minimum sized DFA that accepts the language defined by the regular expression (0+1)*(0+1)(0+1)* is____________.

[GATE 2016 Set-B]

2. Which of the regular expressions given below represent the following DFA? [ ] **[GATE 2014 Set-1]**



I) 0*1(1+00*1)*

II) 0*1*1+11*0*1

III) (0+1)*1

(a) I and II only (b) I and III only

(c) II and III only (d) I, II and III only

3. Consider the languages L1 = Ø and L2 = {a}. Which one of the following represents L1 L2 * U L1* [ ] **[GATE2013]**

(a) { □ } (b) Ø (c) a* (d) { □ ,a}

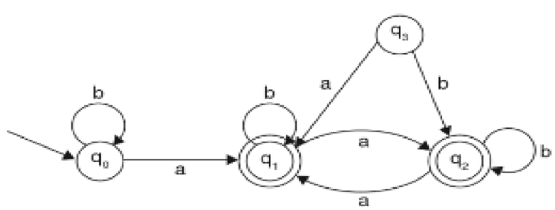
4. Let L = {w ∈ (0 + 1)*|w has even number of 1s}, i.e. L is the set of all bit strings with even number of 1s. Which one of the regular expressions below represents L? [ ]**[GATE 2010]**

(a) (0*10*1)* (b) 0*(10*10*) *

(c) 0*(10*1*)*0* (d) 0*1(10*1)*10*

5. The language accepted by this automaton is given by the regular expression [ ]**[GATE 2007]**

(a) b*ab*ab*ab* (b) (a+b)* (c) b*a(a+b)* (d) b*ab*ab*



6. Consider the language L=(111+11111)*.The minimum number of states in any DFA accepting this language is: [ ] **[GATE 2006]**

(a)3 (b) 5 (c) 8 (d) 9

UNIT-IV

Objective:

To understand regular grammars and context free grammars.

Syllabus:

Chomsky hierarchy of languages, Regular grammars- right linear and left linear grammars, Equivalence between regular linear grammar and FA and its inter conversion, Context free grammar, derivation trees, Sentential forms, right most and left most derivation of strings

Learning Outcomes:

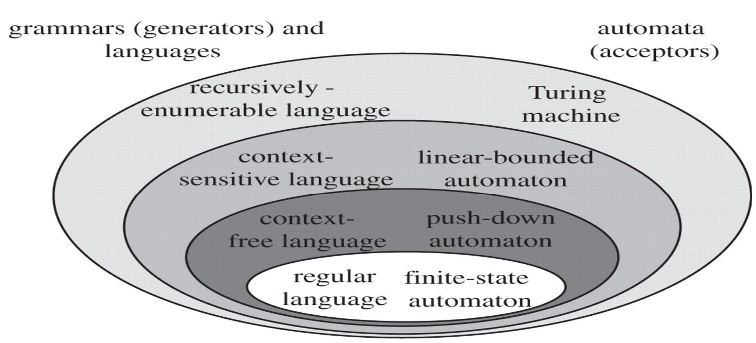
Students will be able to:

- understand Chomsky hierarchy of languages.
- understand and construct the regular grammar for the given regular language or regular expression.
- convert Regular Grammar into equivalent DFA and viceversa.
- construct Context free grammar for the given language.
- construct right most, left most derivation and derivation trees for the given string and grammar.

Learning Material

Chomsky hierarchy of languages:

The four classes of languages are often called the Chomsky hierarchy, after Noam Chomsky, who defined these classes as potential models of natural languages.



the traditional Chomsky hierarchy

Chomsky classifies the grammar into four types:

Grammar	Languages	Automaton	Production rules
Type 0	Recursively enumerable / Phrase Structured	Turing machines	$\alpha \rightarrow \beta$
Type 1	Context-sensitive	Linear-bound automata	$\alpha \rightarrow \beta$ $\lvert\alpha\rvert <= \lvert\beta\rvert$
Type 2	Context-free	Push-down automata	$A \rightarrow \alpha$
Type 3	Regular	Finite-state automata	$A \rightarrow w$ $A \rightarrow wB$ $A \rightarrow Bw$

Regular Grammar:

A right- or left-linear grammar is called a regular grammar.

Right-Linear Grammar:

If all productions of a grammar are of the form $A \rightarrow wB$ or $A \rightarrow w$, where A and B are variables and w is a (possibly empty) string of terminals, then we say the grammar is right-linear.

Example:

Represent the language 0(10)* by the right-linear grammar.

The language generated by the given Regular Expression is

L = {0, 010, 01010, 0101010, ...}

Right-Linear Grammar:

$S \rightarrow 0A$

$A \rightarrow 10A \mid \varepsilon$

Left-Linear Grammar:

If all productions are of the form $A \rightarrow Bw$ or $A \rightarrow w$, we call it left-linear.

Example:

Represent the language 0(10)* by the left-linear grammar.

The language generated by the given Regular Expression is

L = {0, 010, 01010, 0101010, ...}

Left-Linear Grammar:

$S \rightarrow S10 \mid 0$

Equivalence of regular grammars and finite automata:

A language is regular if and only if it has a left-linear grammar and if and only if it has a right-linear grammar.

Construction of a Regular Grammar for a given DFA:

Let M = ({q0, q1... qn} **, ∑, δ, qₒ, F)**. We construct G as G = ({A0, A1, ..., An},∑**, P, A0)**

where P is defined by the following rules:

(i) $A_i \rightarrow aA_j$ is included in P if **δ**$(q_i, a) = q_j \notin F$.

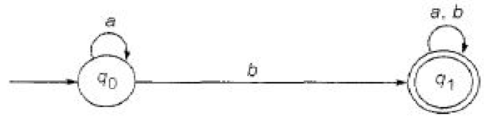
(ii) $A_i \rightarrow aA_j$ and $A_i \rightarrow a$ are included in P if **δ**$(q_i, a) = q_j \in F$.

Note: We can construct only right linear grammar for the given DFA.

If we want to construct **left linear grammar** for the given DFA, reverse the edges of the given DFA and interchange initial and final states.

Example:

1. ***Construct regular grammar (right linear grammar) for the given DFA.***



Given M= ({q0,q1}, {a,b}, **δ, qₒ, {q1})**

Construct G= ({A0,A1}, {a,b} ,P, A0) where *P* is given by

(i) $A_i \rightarrow aA_j$ is included in P if $\delta(q_i, a) = q_j \notin F$.

$\delta(q0, a) = q0 \notin F \Rightarrow A0 \rightarrow aA0$

(ii) Ai → aAj and Ai → a are included in P if δ(qi, a) = qj ∈ F.

δ(q0, b) = q1∈ F ⇒ A0→bA1 and A0→b

δ(q1, a) = q1∈ F ⇒ A1→aA1 and A1→a

δ(q1, b) = q1∈ F ⇒ A1→bA1 and A1→b

∴ P is given by

A0→aA0, A0→bA1, A0→b

A1→aA1, A1→a, A1→bA1, A1→b

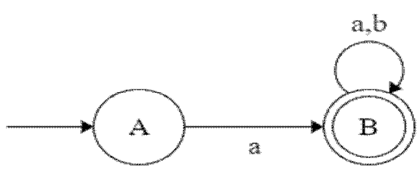
<u>Steps to convert Finite Automata to Left Linear Grammar</u>:

Step 1: Reverse all the edges of the given automata and interchange initial state and final states.

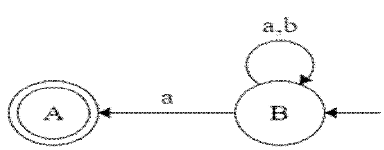
Step 2: Represent the productions using Left Linear Grammar.

Example:

2. ***Construct left linear grammar for the given DFA.***



Step 1: Reverse all the edges of the given automata and interchange initial state and final states.



Step 2: Represent the productions using Left Linear Grammar.

B→Ba B→Aa

B→Bb B→a

<u>Construction of a DFA for a given Regular Grammar</u>:

Let G = **({A0, A1, …., An},∑, P, A0).** We construct a DFA M whose

(i) states correspond to variables.

(ii) initial state corresponds to A0.

(iii) transitions in M correspond to productions in P. As the last production applied in any derivation is of the form Ai→ a, the corresponding transition terminates at a new state, and this is the unique final state.

We define M as **({q0, q1... qn, qf} , ∑, δ, q_o, {qf})** where **δ** is defined as follows:

(i) Each production Ai→aAj induces a transition from qi to qj with label a,

(ii) Each production Ak →a induces a transition from qk to qf with label a.

Example:

1. G= ({A0, A1}, {a,b} ,P, A0) where P consists of A0→aA1, A1→bA1, A1→a, A1→bA0. Construct a DFA M accepting L(G).

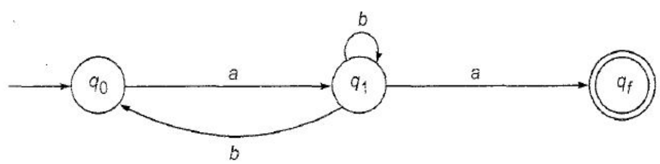
A0→aA1 induces a transition from q0 to q1 with label a.

A1→ bA1 induces a transition from q1 to q1 with label b.

A1→bA0 induces a transition from q1 to q0 with label b.

A1→a induces a transition from q1 to qf with label a.

M = **({q0, q1, qf} , ∑, δ, q_o, {qf}),** where q0 and qf correspond to A0 and A1 respectively and qf is the new final state introduced.

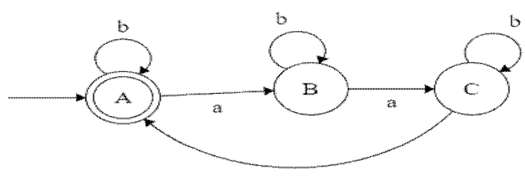


2. Construct Finite Automata for the grammar which consists of the productions

A→ aB | bA | b

B→ aC | bB

C→ aA | bC | a



<u>Context-Free Grammar</u>:

A context-free grammar (CFG or just grammar) is denoted G = (V, T, P, S), where

- V and T are finite sets of variables and terminals, respectively.
- P is a finite set of productions; each production is of the form $A \rightarrow \alpha$, where A is a variable and α is a string of symbols from $(V \cup T)^*$.
- S is a special variable called the start symbol.

The language generated by G [denoted L(G)] is {w | w is in T* and $S \overset{*}{\underset{G}{\Rightarrow}} w$}.
That is, a string is in L(G) if:

1) The string consists solely of terminals.

2) The string can be derived from S.

We call L a context-free language (CFL) if it is L(G) for some CFG G.

Note: **C language is an example for Context Free Language.**

Examples:

1. Write CFG for the language L= $\{a^n b^n \mid n>=1\}$.

   L= {ab, aabb, aaabbb, aaaabbbb, aaaaabbbbb, …………}

   G = ({S}, {a, b}, P, S)

   P: S -> aSb | ab

   (Or)

   S -> aSB

   S -> aB

   B -> b

2. Write CFG for the language L= $\{a^n b^m \mid n , m >=1\}$.

   L= {a, b, ab, aab, abb,aabb, aaabbb, aaaabbbb, aaaaabbbbb,……………………}

   G = ({S, A, B}, {a, b}, P, S)

   P: S -> AB

   A -> aA | a

   B -> bB | b

3. Write CFG for the language L={aa,ab,ba,bb}

    G = ({S, A}, {a, b}, P, S)

    P: S -> AA

    A -> a | b

4. Write CFG for the language L= { a^n | n>=0}.

    L= { ε, a, aa, aaa, aaaa, aaaaa, aaaaaa,……………………….}

    G = ({A}, {a}, P, A)

    P: A -> aA | ε

5. Write CFG for the regular expression (a+b)*.

    L= { ε, a, b, aa, ab , ba, bb, aaa,abb,aba,……………………….}

    G = ({S}, {a, b}, P, S)

    P: S -> aS | bS | ε

6. Write CFG to generate all strings of {a, b} whose length is atleast 2.

    L= { aa, ab , ba, bb, aaa,abb,aba,……………………….}

    G = ({S, A, B}, {a, b}, P, S)

    P: S -> AAB

    A -> a | b

    B -> aB | bB | ε

7. Write CFG to generate all strings of {a, b} whose length is atmost 2.

    L= { ε , a,b, aa, ab , ba, bb}

    G = ({S, A}, {a, b}, P, S)

    P: S -> AA

    A -> a | b | ε

8. Write CFG to generate palindromes over {a, b}.

    L= { ε , a,b, aa,bb,aba,bab,aaaa,abba,……………..}

    G = ({S}, {a, b}, P, S)

    P: S -> aSa | bSb

    S -> a | b | ε

9. Write CFG to generate equal number of a's and b's.

    L= { ab, ba,aabb, abab, bbaa,baba,……………………….}

    G = (V, T, P, S), where V = {S, A, B}, T = {a, b},S and P .

    P: S -> aB A -> bAA

S -> bA   B ->b

A ->a   B ->bS

A ->aS   B -> aBB

Sentential Form:

A string of terminals and variables α is called a sentential form if $S \overset{*}{\Rightarrow} \alpha$.

Derivation:

Derivation is the process of applying productions repeatedly to expand non-terminals in terms of terminals or non-terminals, until there are no more non-terminals.

A derivation can be either Leftmost derivation or Right most derivation.

Leftmost derivation:

If at each step in a derivation a production is applied to the leftmost variable, then the derivation is said to be leftmost.

Example:

Consider the grammar G = ({S, A}, {a, b}, P, S), where P consists of

S →aAS | a

A→ SbA|SS|ba

The corresponding leftmost derivation is

S => aAS => aSbAS => aabAS => aabbaS => aabbaa.

Rightmost derivation:

A derivation in which the rightmost variable is replaced at each step is said to be rightmost.

Example:

Consider the grammar G = ({S, A}, {a, b}, P, S), where P consists of

S →aAS | a

A→ SbA|SS|ba

The corresponding rightmost derivation is

S => aAS => aAa => aSbAa => aSbbaa => aabbaa.

Note:"If w is in L(G) for CFG G, then w has at least one parse tree, and corresponding to a particular parse tree, w has a unique leftmost and a unique rightmost derivation."

Derivation Trees (or) Parse tree:

The derivations in a CFG can be represented using trees. Such trees representing derivations are called derivation trees.

Let **G = (V, T, P, S)** be a CFG. A tree is a derivation (or parse) tree for G if:

1) Every vertex has a label, which is a symbol of V ∪ T ∪ {ε}.
2) The label of the root is S.
3) If a vertex is interior and has label A, then A must be in V.
4) If n has label A and vertices n1, n2, n3, ..., nk are the sons of vertex n, in order from the left, with labels X1, X2, ..., Xk, respectively, then A→X1X2 ....Xk must be a production in P.
5) If vertex n has label ε, then n is a leaf and is the only son of its father.

Example:

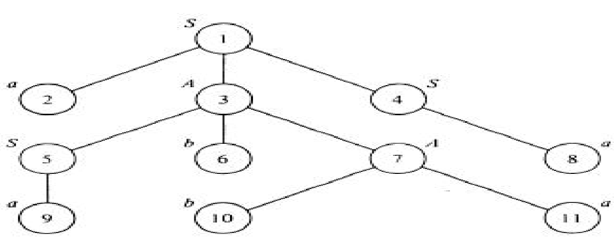
Consider the grammar G = ({S, A}, {a, b], P, S), where P consists of

S →aAS | a

A→ SbA|SS|ba

Construct a derivation tree for the string "aabbaa"

A derivation tree is a natural description of the derivation of a particular sentential form of the grammar G. If we read the labels of the leaves from left to right, we have a sentential form. We call this string the yield of the derivation tree.



S => aAS => aSbAS => aabAS => aabbaS => aabbaa.

Note: Some leaves could be labelled by ε.

UNIT-IV
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. The C language is [ ]
   a) A context free language b) A context sensitive language
   c) A regular language d) None

2. Every regular grammar is context free grammar. (True | False)

3. The finite automata accepts the following language: [ ]
   a) Context free language b) regular language
   c) Context sensitive language d) all of the above

4. Context-free grammar can be recognized by [ ]
   a) Finite Automata b) Linear bounded Automata
   c) Push down Automata d) both (b) and (c)

5. The language accepted by a Turing Machine: [ ]
   a) Type 0 b) Type 1 c) Type 2 d) Type 3

6. Match the following

1. Context Free Language	a. Turing Machine	[ ]
2. Recursively Enumerable	b. Finite Automata	[ ]
3. Regular Language	c. Linear Bounded Automata	[ ]
4. Context Sensitive Language	d. Push Down Automata	[ ]

7. For every right linear grammar, there will be an equivalent FA. [True/ False]

8. Recursively Enumerable language is also called as ______________.

9. A context free grammar is [ ]
   a) Type 0 b) Type 1 c) Type 2 d) Type 3

10. Which word can be generated by S->d| bA ,A->d |ccA [ ]
    a) bccccd b) aabccd c) ababccd d) abbbd

11. Which of the following strings is in the language defined by grammar
    S→ 0A , A→ 1A | 0A | 1 [ ]
    a) 01100 b) 00101 c) 10011 d) 11111

12. Recognize the CFL for the given CFG. [ ]
    S-> aB| bA,
    A-> a|aS|bAA,
    B-> b|bS|aBB
    a) strings contain equal number of a's and equal number of b's.
    b) strings contain odd number of a's and odd number of b's.

c) strings contain odd number of a's and even number of b's.
d) strings contain even number of a's and even number of b's

13. Given the following productions of a grammar: [ ]
$S \rightarrow aA \mid aBB$ $A \rightarrow aaA \mid \varepsilon$ $B \rightarrow bB \mid bbC$ $C \rightarrow B$
Which of the following is true?
a) The language corresponding to the given grammar is a set of even number of a's.
b) The language corresponding to the given grammar is a set of odd number of a's.
c) The language corresponding to the given grammar is a set of even number of a's followed by odd number of b's.
d) The language corresponding to the given grammar is a set of odd number of a's followed by even number of b's.

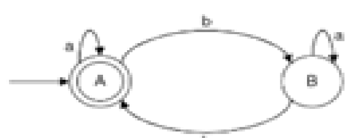
14. A regular grammar for the language L={ $a^n b^m$ | n is even and m is even} is [ ]
a) $S \rightarrow aSb \mid X$; $X \rightarrow bXa \mid \varepsilon$
b) $S \rightarrow aaS \mid X$; $X \rightarrow bSb \mid \varepsilon$
c) $S \rightarrow aSb \mid X$; $X \rightarrow Xab \mid \varepsilon$
d) $S \rightarrow aaS \mid X$; $X \rightarrow bbX \mid \varepsilon$

15. Which of the regular expressions corresponds to this grammar?
$S \rightarrow AB \mid AS$ $A \rightarrow a \mid aA$ $B \rightarrow b$
a) (aa)*b b) aa*b c) (ab)* d) a(ab)* [ ]

16. Identify the language generated by the following grammar [ ]
$S \rightarrow aS \mid bS \mid abA$
$A \rightarrow aA \mid bA \mid \varepsilon$
a) L= x | ab is a substring of x, x ∈{a,b}*
b) L= x | a is a substring of x, x ∈{a,b}*
c) L= x | b is a substring of x, x ∈{a,b}*
d) L= x | ba is a substring of x, x ∈{a,b}*

17. The CFG $S \rightarrow aS \mid bS \mid a \mid b$ is equivalent to the regular expression
a) (a*+b)* b) (a+b)* c) (a+b) (a+b)* d) (a+b)(a+b) [ ]

18. The regular grammar for the given FA is [ ]

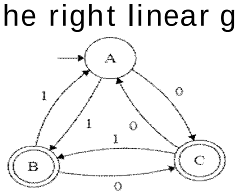


a) $A \rightarrow aA \mid bB \mid a$
$B \rightarrow bA \mid aB \mid b$
b) $A \rightarrow aA \mid bB \mid \varepsilon$
$B \rightarrow bA \mid aB \mid \varepsilon$
c) $A \rightarrow aA \mid bB \mid b$
$B \rightarrow bA \mid aB \mid a$
d) $A \rightarrow bA \mid aB \mid a$
$B \rightarrow aA \mid bB \mid b$

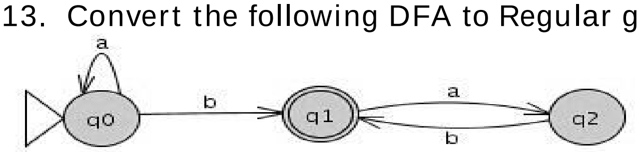
SECTION-B

SUBJECTIVE QUESTIONS

1. Show the Venn diagram of Chomsky hierarchy language and their counterpart automata.
2. Define Regular grammar with an example.
3. Define Context Free Grammar with an example.
4. What is sentential form? Explain with an example.
5. Explain derivation tree with an example.
6. Define LMD and RMD.
7. Show that **id+id*id** can be generated by two distinct derivation trees for the grammar
   E → E+E | E*E | (E) | id
8. Design CFG for odd palindromes?
9. Let G be the grammar
   S→ aB | bA
   A→ a | aS | bAA
   B → b | bS | aBB.
   For the string aaabbabbba find a
   i. Left most derivation
   ii. Right most derivation
   iii. Parse Tree
10. Obtain the right linear grammar for the following FA.



11. Obtain a Right Linear Grammar for the language L = {a^n b^m | n>=2 , m>=3}
12. Obtain the left linear grammar for (11+01)*101.
13. Convert the following DFA to Regular grammar

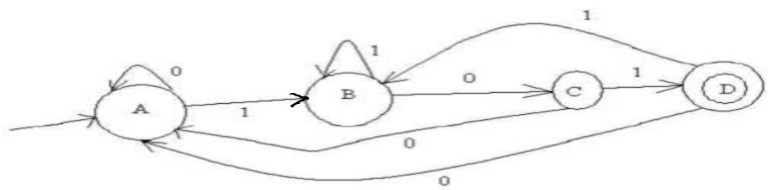


14. Is the following grammar ambiguous?
    S→ AB|aaB
    A→> a|Aa
    B→b

15. Find the language generated by the following grammar.
$S \rightarrow SS$ $S \rightarrow aa$ $S \rightarrow \varepsilon$

16. Draw a derivation tree for the string abaaba for the CFG given by G where P = {$S \rightarrow aSa$ $S \rightarrow bSb$ $S \rightarrow a \mid b \mid \epsilon$ }

17. Obtain a right linear grammar and left linear grammar for the following FA.



SECTION-C
QUESTIONS AT THE LEVEL OF GATE

1. $G1$: $S \rightarrow aS \mid B$, $B \rightarrow b \mid bB$ **[GATE 2016]**

$G2$: $S \rightarrow aA \mid bB$; $A \rightarrow aA \mid B \mid$ □ , $B \rightarrow bB \mid$ □

Which one of the following pairs of languages is generated by $G1$ and $G2$, respectively? [ ]

a) $\{a^m b^n \mid m > 0 \text{ or } n > 0\}$ and $\{a^m b^n \mid m > 0 \text{ and } n > 0\}$

b) $\{a^m b^n \mid m > 0 \text{ and } n > 0\}$ and $\{a^m b^n \mid m > 0 \text{ or } n > =0\}$

c) $\{a^m b^n \mid m >= 0 \text{ or } n > 0\}$ and $\{a^m b^n \mid m > 0 \text{ and } n > 0\}$

d) $\{a^m b^n \mid m >= 0 \text{ and } n > 0\}$ and $\{a^m b^n \mid m > 0 \text{ or } n > 0\}$

2. $S \rightarrow aSa \mid bSb \mid a \mid b$ [ ] **[GATE 2009]**
The language generated by the above grammar over the alphabet {a, b} is the of
a) all palindromes
b) all odd length palindromes
c) strings that begin and
d) all even length palindromes end with the same symbol

3. Consider the CFG with {S,A,B} as the non-terminal alphabet {a,b} as the terminal alphabet,S as the start symbol and the following set of production rules: **[GATE 2007]**
$S \rightarrow aB$ $S \rightarrow bA$
$B \rightarrow b$ $A \rightarrow a$
$B \rightarrow bS$ $A \rightarrow aS$

B → aBB S → bAA

Which of the following strings is generated by the grammar? [ ]

a) aaaabb b) aabbbb c) aabbab d) abbbba

4. How many derivation trees are there for the grammar in Question 3?

a) 1 b) 2 c) 3 d) 4 [ ]

5. **[GATE 2006]**

Which one of the following grammars generates the language $L = \{a^i b^j \mid i \neq j\}$?

(A)

$S \to AC \mid CB$
$C \to aCb \mid a \mid b$
$A \to aA \mid \epsilon$
$B \to Bb \mid \epsilon$

(B) $S \to aS \mid Sb \mid a \mid b$

(C)

$S \to AC \mid CB$
$C \to aCb \mid \epsilon$
$A \to aA \mid \epsilon$
$B \to Bb \mid \epsilon$

(D)

$S \to AC \mid CB$
$C \to aCb \mid \epsilon$
$A \to aA \mid a$
$B \to Bb \mid b$

6. Consider the regular grammar: **[GATE 2005]**

S→Xa | Ya
X→Za
Z→Sa | □
Y→Wa
W→Sa

where S is the starting symbol, the set of terminals is {a} and the set of non-terminals is {S, W, X, Y, Z}. We wish to construct a deterministic finite automaton (DFA) to recognize the same language. What is the minimum number of states required for the DFA? [ ]

a) 2 b) 3 c)4 d)5

UNIT-V

Objective:

To understand and design push down automata's for a given Context free language.

Syllabus:

Ambiguity in context free grammars, minimization of Context Free Grammars, Chomsky normal form, Greibach normal form, pumping lemma for Context Free Languages, closure properties of CFL (proofs not required), applications of CFLs

Push down automata:

Push down automata, model of PDA, design of PDA.

Learning Outcomes:

Students will be able to:

- understand ambiguity in context free grammars.
- minimize the given context free grammar.
- apply Chomsky and Greibach Normal Forms on context free grammars.
- understand and design PDA for given context free languages.

Learning Material

Ambiguity in context free grammars:

A context-free grammar G is said to be ambiguous if it has two parse trees for some word.

(or)

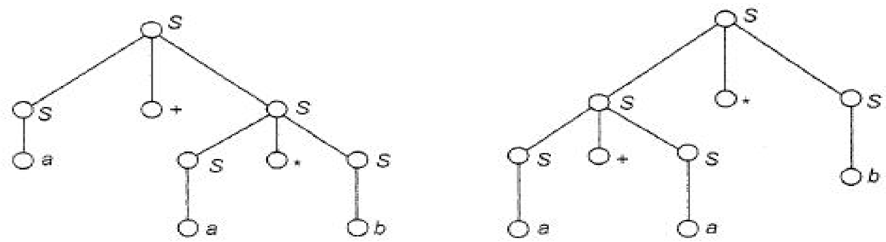
A word which has more than one leftmost derivation or more than one rightmost derivation is said to be ambiguous.

Note: A CFL for which every CFG is ambiguous is said to be an inherently ambiguous CFL.

Example:

G = *({S}, {a, b,* +, *}, *P. S),* where *P* consists of S→S+S | S*S | a | b

We have two derivation trees for *a* + *a* * *b*



Two derivation trees for a + a * *b*

<u>Minimization of Context Free Grammars:</u>

1) Elimination of useless symbols.

2) Elimination of ε –Productions.

3) Elimination of Unit Productions.

<u>Elimination of Useless Symbols</u>:

Let G=(V, T, P, S) be a grammar. A symbol X is useless if it is not involved in derivation.

(or)

A symbol X is useless if there is no way of getting a terminal string from it.

Example:

Consider the grammar

S→AB | a

A→ a

We find that no terminal string is derivable from B. We therefore eliminate B and the production S → AB.

Then the grammar is

S→a

A→a

We find that only S and a appear in sentential forms. Thus ({S}, {a}, {S → a}, S) is an
equivalent grammar with no useless symbols.

Elimination of ε –Productions:

A production of the form *A* → ε, where *A* is a variable, is called a *null production.*

If L = L(G) for some CFG G = (V, T, P, S), then L - { ε } is L(G') for a CFG G' with no useless symbols or ε -productions.

Example:

Consider the grammar

A→0B1 | 1B1

B→0B | 1B | ε

Remove ε-productions from the grammar.

B→ ε is the null production.

The new productions after elimination of ε are

A→0B1 | 1B1| 01 | 11

B→0B | 1B | 0 | 1

Elimination of Unit Productions:

A production of the form A→B whose right-hand side consists of a single variable is called a unit production.

All other productions, including those of the form A →a and ε -productions, are nonunit productions.

Example:

Consider the grammar

S→0A | 1B | C

A→0S | 00

B→1 | A

C→01

Remove unit production from the grammar.

S→C and B→A are the unit productions

The new productions after elimination of unit productions are

$S \rightarrow 0A \mid 1B \mid 01$

$A \rightarrow 0S \mid 00$

$B \rightarrow 1 \mid 0S \mid 00$

$C \rightarrow 01$

C is a useless symbol. So eliminate C production.

The final set of productions are

$S \rightarrow 0A \mid 1B \mid 01$

$A \rightarrow 0S \mid 00$

$B \rightarrow 1 \mid 0S \mid 00$

Chomsky Normal Form :(CNF)

Any context-free language without ε is generated by a grammar in which all productions are of the form $A \rightarrow BC$ or $A \rightarrow a$. Here, A, B, and C, are variables and a is a terminal.

Step 1: Simplify the grammar.

a) Eliminate ε –productions

b) Eliminate unit productions

c) Eliminate Useless symbols.

The given grammar does not contain ε –productions, unit productions and useless symbols.

It is in optimized form.

Step 2: Consider a production in P,of the form $A \rightarrow X_1X_2X_3.....X_m$ where $m>=2$.If X_i is a terminal a, introduce a new variable C_a and a production $C_a \rightarrow a$.Then replace X_i by Ca.

Step 3: Consider a production $A \rightarrow B_1B_2B_3.....B_m$ where $m>=3$,create new variables $D_1,D_2,....D_{m-2}$ and replace $A \rightarrow B_1B_2B_3...B_m$ by the set of productions $\{A \rightarrow B_1D_1, D_1 \rightarrow B_2D_2, ... D_{m-3} \rightarrow B_{m-2}\, D_{m-2}, D_{m-2} \rightarrow B_{m-1}B_m\}$

Example:

Consider the grammar ({S, A, B}, {a, b}, P, S) that has the productions:

$S \rightarrow bA \mid aB$

$A \rightarrow bAA \mid aS \mid a$

$B \rightarrow aBB \mid bS \mid b$

Find an equivalent grammar in CNF.

Step 1: Simplify the grammar.

a) Eliminate ε –productions

b) Eliminate unit productions

c) Eliminate Useless symbols.

The given grammar does not contain ε –productions, unit productions and useless symbols.

It is in optimized form.

Step 2: The only productions already in proper form are $A \rightarrow a$ and $B \rightarrow b$.

So we may begin by replacing terminals on the right by variables, except in the case of the productions $A \rightarrow a$ and $B \rightarrow b$.

$S \rightarrow bA$ is replaced by $S \rightarrow C_bA$ and $C_b \rightarrow b$.

Similarly, $A \rightarrow aS$ is replaced by $A \rightarrow C_aS$ and $C_a \rightarrow a$; $A \rightarrow bAA$ is replaced by $A \rightarrow C_bAA$; $S \rightarrow aB$ is replaced by $S \rightarrow C_aB$;

$B \rightarrow bS$ is replaced by $B \rightarrow C_bS$, and $B \rightarrow aBB$ is replaced by $B \rightarrow C_aBB$.

In the next stage, the production $A \rightarrow C_bAA$ is replaced by $A \rightarrow C_bD_1$ and $D_1 \rightarrow AA$, and the production $B \rightarrow C_aBB$ is replaced by $B \rightarrow C_aD_2$ and $D_2 \rightarrow BB$.

Step 3: The productions for the grammar in CNF are :

$S \rightarrow C_bA \mid C_aB$

$A \rightarrow C_aS \mid C_bD_1 \mid a$

$B \rightarrow C_bS \mid C_aD_2 \mid b$

$D_1 \rightarrow AA$

$D_2 \rightarrow BB$

$C_a \rightarrow a$

$C_b \rightarrow b$

Greibach Normal Form:

Every context-free language L without e can be generated by a grammar for which every production is of the form A → aα, where A is a variable, a is a terminal, and α is a (possibly empty) string of variables.

Lemma 1: Define an A-production to be a production with variable A on the left. Let G = (V, T, P, S) be a CFG. Let A → α1Bα2 be a production in P and B→β1 | β2| ……|βr be the set of all B-productions. Let G1 = (V, T, P1, S) be obtained from G by deleting the production A → α1Bα2 from P and adding the productions A→ α1β1α2 | α1β2α2 | ……| α1βrα2. Then L(G) = L(G1).

Lemma 2: Let G = (V, T, P, S) be a CFG. Let A→Aα1 | Aα2 | ……| Aαr be the set of A-productions for which A is the leftmost symbol of the right-hand side. Let A → β1 | β2| ……|βs be the remaining A-productions. Let G1 = (V U {B}, T, P1, S) be the CFG formed by adding the variable B to V and replacing all the A-productions by the productions:

$$1)\ \left.\begin{array}{l} A \rightarrow \beta_i \\ A \rightarrow \beta_i B \end{array}\right\} 1 \le i \le s, \qquad 2)\ \left.\begin{array}{l} B \rightarrow \alpha_i \\ B \rightarrow \alpha_i B \end{array}\right\} 1 \le i \le r.$$

Then L(G1) = L(G).

Example:

Convert to Greibach normal form the grammar

G=i{A1,A2,A3}, {a, b}, P A1),

where P consists of the following:

$$A_1 \rightarrow A_2 A_3$$
$$A_2 \rightarrow A_3 A_1 | b$$
$$A_3 \rightarrow A_1 A_2 | a$$

Step 1 Since the right-hand side of the productions for A_1 and A_2 start with terminals or higher-numbered variables, we begin with the production $A_3 \rightarrow A_1 A_2$ and substitute the string $A_2 A_3$ for A_1. Note that $A_1 \rightarrow A_2 A_3$ is the only production with A_1 on the left.

The resulting set of productions is:

$$A_1 \rightarrow A_2 A_3$$
$$A_2 \rightarrow A_3 A_1 | b$$
$$A_3 \rightarrow A_2 A_3 A_2 | a$$

Since the right side of the production $A_3 \rightarrow A_2 A_3 A_2$ begins with a lower-numbered variable, we substitute for the first occurrence of A_2 both $A_3 A_1$ and b. Thus $A_3 \rightarrow A_2 A_3 A_2$ is replaced by $A_3 \rightarrow A_3 A_1 A_3 A_2$ and $A_3 \rightarrow bA_3 A_2$. The new set is

$$A_1 \rightarrow A_2 A_3$$
$$A_2 \rightarrow A_3 A_1 | b$$
$$A_3 \rightarrow A_3 A_1 A_3 A_2 | bA_3 A_2 | a$$

We now apply Lemma 2 to the productions

$$A_3 \rightarrow A_3 A_1 A_3 A_2 | bA_3 A_2 | a.$$

Symbol B_3 is introduced, and the production $A_3 \rightarrow A_3 A_1 A_3 A_2$ is replaced by $A_3 \rightarrow bA_3 A_2 B_3$, $A_3 \rightarrow aB_3$, $B_3 \rightarrow A_1 A_3 A_2$, and $B_3 \rightarrow A_1 A_3 A_2 B_3$. The resulting set is

$$A_1 \rightarrow A_2 A_3$$
$$A_2 \rightarrow A_3 A_1 | b$$
$$A_3 \rightarrow bA_3 A_2 B_3 | aB_3 | bA_3 A_2 | a$$
$$B_3 \rightarrow A_1 A_3 A_2 | A_1 A_3 A_2 B_3$$

Step 2 Now all the productions with A_3 on the left have right-hand sides that start with terminals. These are used to replace A_3 in the production $A_2 \to A_3 A_1$ and then the productions with A_2 on the left are used to replace A_2 in the production $A_1 \to A_2 A_3$. The result is the following.

$$A_3 \to bA_3A_2B_3 \qquad A_3 \to bA_3A_2$$
$$A_3 \to aB_3 \qquad A_3 \to a$$
$$A_2 \to bA_3A_2B_3A_1 \qquad A_2 \to bA_3A_2A_1$$
$$A_2 \to aB_3A_1 \qquad A_2 \to aA_1$$
$$A_2 \to b$$
$$A_1 \to bA_3A_2B_3A_1A_3 \qquad A_1 \to bA_3A_2A_1A_3$$
$$A_1 \to aB_3A_1A_3 \qquad A_1 \to aA_1A_3$$
$$A_1 \to bA_3$$
$$B_3 \to A_1A_3A_2 \qquad B_3 \to A_1A_3A_2B_3$$

Step 3 The two B_3-productions are converted to proper form, resulting in 10 more productions. That is, the productions

$$B_3 \to A_1A_3A_2 \qquad \text{and} \qquad B_3 \to A_1A_3A_2B_3$$

are altered by substituting the right side of each of the five productions with A_1 on the left for the first occurrences of A_1. Thus $B_3 \to A_1 A_3 A_2$ becomes

$$B_3 \to bA_3A_2B_3A_1A_3A_3A_2, \qquad B_3 \to aB_3A_1A_3A_3A_2.$$
$$B_3 \to bA_3A_3A_2, \qquad B_3 \to bA_3A_2A_1A_3A_3A_2, \qquad B_3 \to aA_1A_3A_3A_2.$$

The other production for B_3 is replaced similarly. The final set of productions is

$$A_3 \to bA_3A_2B_3 \qquad A_3 \to bA_3A_2$$
$$A_3 \to aB_3 \qquad A_3 \to a$$
$$A_2 \to bA_3A_2B_3A_1 \qquad A_2 \to bA_3A_2A_1$$
$$A_2 \to aB_3A_1 \qquad A_2 \to aA_1$$
$$A_2 \to b$$
$$A_1 \to bA_3A_2B_3A_1A_3 \qquad A_1 \to bA_3A_2A_1A_3$$
$$A_1 \to aB_3A_1A_3 \qquad A_1 \to aA_1A_3$$
$$A_1 \to bA_3$$
$$B_3 \to bA_3A_2B_3A_1A_3A_3A_2B_3 \qquad B_3 \to bA_3A_2B_3A_1A_3A_3A_2$$

$$B_3 \to aB_3A_1A_3A_3A_2B_3 \qquad B_3 \to aB_3A_1A_3A_3A_2$$
$$B_3 \to bA_3A_3A_2B_3 \qquad B_3 \to bA_3A_3A_2$$
$$B_3 \to bA_3A_2A_1A_3A_3A_2B_3 \qquad B_3 \to bA_3A_2A_1A_3A_3A_2$$
$$B_3 \to aA_1A_3A_3A_2B_3 \qquad B_3 \to aA_1A_3A_3A_2$$

Pumping Lemma for CFL's:

Let L be any CFL. Then there is a constant n, depending only on L, such that if z is in L and | z | ≥ n, then we may write z = uvwxy such that

1) $|vx| \geq 1$,
2) $|vwx| \leq n$, and
3) for all $i \geq 0$ uv^iwx^iy is in L.

Example:

Consider the language L = {$a^ib^ic^i$ | ≥ 1}. Suppose L were context free and let n be the constant.

Consider z = $a^nb^nc^n$. Write z = uvwxy so as to satisfy the conditions of the pumping lemma.

Since |vwx| ≤ n, it is not possible for vx to contain instances of a's and c's, because the rightmost a is n + 1 positions away from the leftmost c.

If v and x consist of a's only, then uwy (the string uv^iwx^iy with i = 0) has n b's and n c's but fewer than n a's since |vx| ≥ 1.

Thus, uwy is not of the form $a^ib^ic^i$. But by the pumping lemma vwy is in L, a contradiction.

The cases where v and x consist only of b's or only of c's are disposed of similarly.

If vx has a's and b's, then uwy has more c's than a's or b's, and again it is not in L.

If vx contains b's and c's, a similar contradiction results.

We conclude that L is not a context-free language.

Closure Properties of CFL's:

- Context-free languages are closed under union, concatenation and Kleene closure.
- The context-free languages are closed under substitution.
- The CFL's are closed under homomorphism.
- The CFL's are not closed under intersection.
- The CFL's are not closed under complementation.

Applications of the pumping lemma:

The pumping lemma can be used to prove a variety of languages not to be context free, using the same "adversary" argument as for the regular set pumping lemma.

Push down automata:

A *pushdown automaton* M is a system $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where

1) Q is a finite set of *states*;
2) Σ is an alphabet called the *input alphabet*;
3) Γ is an alphabet, called the *stack alphabet*;
4) q_0 in Q is the *initial state*;
5) Z_0 in Γ is a particular stack symbol called the *start symbol*;
6) $F \subseteq Q$ is the set of *final states*;
7) δ is a mapping from $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$.

Moves:

The interpretation of

$$\delta(q, a, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \ldots, (p_m, \gamma_m)\}$$

where q and p_i, $1 \le i \le m$, are states, a is in Σ, Z is a stack symbol, and γ_i is in Γ^*, $1 \le i \le m$, is that the PDA in state q, with input symbol a and Z the top symbol on the stack can, for any i, enter state p_i, replace symbol Z by string γ_i, and advance the input head one symbol. We adopt the convention that the leftmost symbol of γ_i will be placed highest on the stack and the rightmost symbol lowest on the stack. Note that it is not permissible to choose state p_i and string γ_j for some $j \ne i$ in one move.

The interpretation of

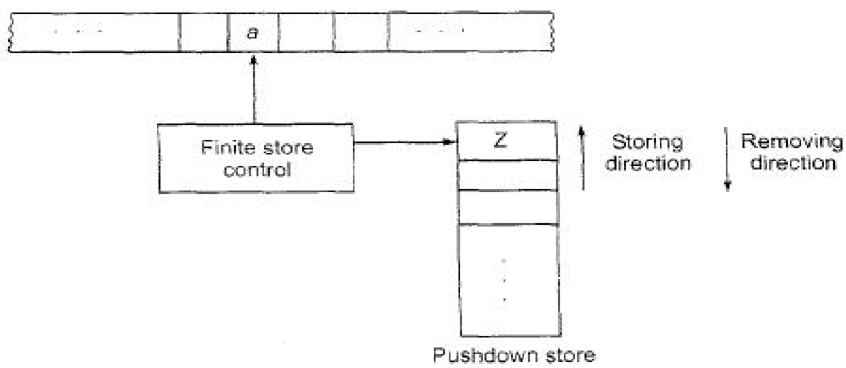
$$\delta(q, \epsilon, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \ldots, (p_m, \gamma_m)\}$$

is that the PDA in state q, independent of the input symbol being scanned and with Z the top symbol on the stack, can enter state p_i and replace Z by γ_i for any i, $1 \le i \le m$. In this case, the input head is not advanced.

Model of PDA:

- Pushdown automaton has a read-only input tape, an input alphabet a finite state control, a set of final states, and an initial state as in the case of an FA.

- In addition to these, it has a stack called the pushdown store. It is a read-write pushdown store as we can add elements to PDS or remove elements from PDS.
- A finite automaton is in some state and on reading, an input symbol moves to a new state.
- The pushdown automaton is also in some state and on reading an input symbol and the topmost symbol in PDS, it moves to a new state and writes (adds) a string of symbols in PDS.



Instantaneous description:

Instantaneous description (ID) is the configuration of a PDA at a given instant. We define an ID to be a triple (q, w, γ), where q is a state, w a string of input symbols, and γ a string of stack symbols.

If $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a PDA,

we say $(q, aw, Z\alpha) \vdash_M (p, w, \beta\alpha)$ if $\delta(q, a, Z)$ contains (p, β).

Note that a may be ϵ or an input symbol.

Accepted Languages:

For PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ we define $L(M)$, the *language accepted by final state*, to be

$$\{w \mid (q_0, w, Z_0) \overset{*}{\vdash} (p, \epsilon, \gamma) \text{ for some } p \text{ in } F \text{ and } \gamma \text{ in } \Gamma^*\}.$$

We define $N(M)$, the *language accepted by empty stack* (or *null stack*) to be

$$\{w \mid (q_0, w, Z_0) \overset{*}{\vdash} (p, \epsilon, \epsilon) \text{ for some } p \text{ in } Q\}.$$

When acceptance is by empty stack, the set of final states is irrelevant, and, in this case, we usually let the set of final states be the empty set.

Example:

Design a PDA that accepts $\{ww^R \mid w \text{ in } (0+1)^*\}$

L = { ε, 0, 1, 00, 11, 0110, 1001, ………}

Let M= (Q, ∑, Γ, δ, q_0, Z_0, F) be the PDA

Consider M = ({q1, q2}, {0, 1}, {0, 1, Z_0}, δ, q_1, Z_0, Ø)

$\delta(q1,0,Z_0)=\{(q1,0Z_0)\}$

$\delta(q1,1,Z_0)=\{(q1,1Z_0)\}$

$\delta(q1,0,0)=\{(q1,00),(q2, \varepsilon)\}$

$\delta(q1,1,0)=\{(q1,10)\}$

$\delta(q1,0,1)=\{(q1,01)\}$

$\delta(q1,1,1)=\{(q1,11),(q2, \varepsilon)\}$

$\delta(q2,0,0)=\{(q2, \varepsilon)\}$

$\delta(q2,1,1)=\{(q2, \varepsilon)\}$

$\delta(q1,\varepsilon,Z_0)=\{(q2, \varepsilon \}$

$\delta(q2,\varepsilon,Z_0)=\{(q2, \varepsilon)\}$

Deterministic PDA:

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

Formally we say a PDA M is deterministic if:

1) for each q in Q and Z in Γ, whenever $\delta(q, \epsilon, Z)$ is nonempty, then $\delta(q, a, Z)$ is empty for all a in Σ;
2) for no q in Q, Z in Γ, and a in $\Sigma \cup \{\epsilon\}$ does $\delta(q, a, Z)$ contain more than one element.

Equivalence of PDA's and CFL's :

CFG to PDA Conversion

If L is a context-free language, then there exists a PDA M such that L = N(M).

Procedure:

Let L=L(G), where G=(V_N, Σ,P,S) is a context free grammar.

We construct a PDA M as

M= ((q), Σ, V_N U Σ, Z_0,q, δ, Ø)

Where δ is defined by the following rules:

R1: δ (q, ε,A) = {(q, α) | A -> α is in P}

R2: δ (q,a,a) = { (q, ε)} for every a in Σ.

Example:

Construct a pda M equivalent to the following context free grammar:

S->0BB

B->0S|1S|0.

Test whether 010^4 is in N(M).

Solution:

Define pda A as follows:

A=(({q},{0,1},{S,B,0,1}, δ,q, Z_0, Ø)

δ is defined by the following rules:

R_1: δ(q, ε, S)={(q,0BB)}

R_2: δ(q, ε, B)={(q,0S),(q,0S),(q,0)}

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

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

String Checking

$(q,010^4,S)$	
$\vdash (q, 010^4, 0BB)$	by Rule R_1
$\vdash (q, 10^4, BB)$	by Rule R_3
$\vdash (q, 10^4,1SB)$	by Rule R_2 since $(q,1S)\in \alpha(q, \wedge, B)$
$\vdash (q, 0^4, SB)$	by Rule R_4
$\vdash (q, 0^4, 0BBB)$	by Rule R_1
$\vdash (q, 0^3,BBB)$	by Rule R_3
$\vdash^* (q, 0^3,000)$	by Rule R_2 since $(q,0) \in \alpha(q, \wedge, B)$
$\vdash^* (q, \varepsilon , \varepsilon)$	by Rule R_3

UNIT-V

Assignment-Cum-Tutorial Questions

A. Objective Questions

1. Grammar that produce more than one Parse tree for same word is:
   a) Ambiguous b) Unambiguous [ ]
   c) Complementation d) Concatenation Intersection
2. For every grammar there will an equivalent grammar in CNF. [True/False]
3. The derivation trees of strings generated by a context free grammar in Chomsky Normal Form are always binary trees [True |False]
4. Which of the following conversion is not possible (algorithmically)? [ ]
   a) Regular grammar to Context-free grammar
   b) Nondeterministic FSA to Deterministic FSA
   c) Nondeterministic PDA to Deterministic PDA
   d) All of the above
5. CFL's are not closed intersection and complementation. [True | False]
6. CFL's are closed under [ ]
   a) union b) concatenation c) closure d) All
7. The grammar G with the productions [ ]
   $A \rightarrow AA \mid (a) \mid \varepsilon$ is an
   a) Ambiguous grammar b) Unambiguous grammar
   c) Grammar d) None
8. Identify the useless symbol in the grammar given below. [ ]
   S->AB | C A->a B-> BC C->b
   a) S b) A c) B d) C
9. Find an equivalent reduced grammar for the given grammar. [ ]
   S-> 0 | 1 | ε S-> 0S0 | 1S1
   a) S->0 | 1 ,S->0S0 |1S1 | 0 | 1 b) S->0 | 1 ,S->SS|0S1 | 1S1
   c) S-> 0 | 1, S->00 |11 d) None

10. Which one of the following is a Chomsky Normal Form grammar? [ ]

(i) A->BC |a (ii) A->aA |a |b (iii) A->BCD |a, B->a, C->c, D->d

a) (i) only b) (i) and (iii) c) (ii) and (iii) d) (i),(ii) and (iii)

11. Which one of the following is not a Greibach Normal form grammar? [ ]

(i) S->a |bA |aA |bB
A->a
B->b

(ii) S->a |aA|AB
A->a
B->b

(iii) S->a | A| aA
A->a

a) (i) and (ii) b) (i) and (iii) c) (ii) and (iii) d)(i),(ii) and (iii)

12. L={ 0^n 1^{2n} |n>=1} is [ ]

a) regular b) context-free but not regular

c) context-free but regular d) None

13. Recognize the language accepted by the PDA with the following moves [ ]

δ (q_0,a,Z_0)=(q_0,aZ_0) , δ (q_0,a,a)=(q_0,aa)

δ (q_0,b,a)=(q_1, ε) , δ (q_1,b,a)=(q_1, ε)

δ (q_1,c,Z_0)=(q_2, Z_0), δ (q_2,c,Z_0)=(q_2, Z_0)

a) L={$a^n b^n c^n$ | n, m>=1} b) L={$a^n b^n c^m$ | n, m>=1}

c) L={$a^m b^n c^n$ | n, m>=1} d) L={$a^m b^n c^m$ | n, m>=1}

14. The grammars G1 and G2 are

G1: S -> 0S0| 1S1 | 0|1| ε

G2: is S -> as |asb| X, X -> Xa | a.

Which is the correct statement? [ ]

a) G1 is ambiguous, G2 is unambiguous

b) G1 is unambiguous, G2 is ambiguous

c) Both G1 and G2 are ambiguous

d) Both G1 and G2 are unambiguous

B. Descriptive questions

1. What is an ambiguous grammar? Explain with an example.
2. Define Useless symbol and give example.
3. What is an Null production and Unit producation? Explain with an example.
4. List the applications of CFG.
5. List the closure properties of CFL.
6. Explain pumping lemma for CFL's with an example.
7. Explain the model of PDA.
8. Show that the grammar is ambiguous.
   $S \rightarrow 0A \mid 1B$
   $A \rightarrow 0AA \mid 1S \mid 1$
   $B \rightarrow 1BB \mid 0S \mid 0$
9. Convert the following grammar in to GNF
   S→ XA |BB
   B→ b |SB
   X→ b
10. Design PDA for $L=\{wcw^r \mid w \in (0+1)^*\}$
11. Design PDA for the language $L = \{ a^n b^{n+m} c^m \mid n,m >= 1 \}$
12. What is the language generated by the grammar G=(V,T,P,S) where P={S->aSb, S->ab}?
13. For the following grammar :
    S -> ABC | BbB , A-> aA | BaC|aaa , B-> bBb| a|D ,C->CA|AC ,D-> ε
    i. Eliminate ε-productions.
    ii. Eliminate any unit productions in the resulting grammar.
    iii. Eliminate any useless symbols in the resulting grammar.
    iv. Put the resulting grammar in Chomsky Normal Form
14. Find a CFG, without ε productions, unit productions and useless productions equivalent to the grammar defined by
    S→ABaC
    A→ BC
    B→ b|ε
    C→ D|ε
    D→ d

15. Obtain the PDA for the given regular language: L= $\{ww^r | w$ is in $(0+1)^*\}$.

16. Convert the following Grammar into CNF.

$S \rightarrow AbcD$ / abc
$A \rightarrow aASB$ / d
$B \rightarrow b$/ cb
$D \rightarrow d$

17. Consider the grammar ({S, A, B}, {a, b}, P, S) that has the productions:

S→bA | aB
A→ bAA | aS | a
B→aBB | bS | b

Find an equivalent grammar in CNF.

18. Show that $L= \{a^n b^n c^n \mid n \geq 0\}$ is not a context free language.

C. *Gate Questions*

1. Identify the language generated by the following grammar, Where S is the start variable. [ ] **[Gate 2017]**

S->XY

X->aX |a

Y->aYb |epsilon

A) $\{a^m b^n \mid m>=n, n>0\}$ B) $\{a^m b^n \mid m>=n, n>=0\}$

C) $\{a^m b^n \mid m>n, n>=0\}$ D) $\{a^m b^n \mid m>n, n>0\}$

2. Consider the following statements about the context free grammar
$G = \{S \rightarrow SS, S \rightarrow ab, S \rightarrow ba, S \rightarrow E\}$ [ ] **[Gate 2006]**

I. G is ambiguous

II. G produces all strings with equal number of a's and b's

III. G can be accepted by a deterministic PDA.

Which combination below expresses all the true statements about G?

a) I only **b)** I and III only
c) I and II only **d)** I, II and III

3. Consider the languages: [ ] **[Gate 2005]**

$L_1 = \{ww^R \mid w$ belongs $\{0,1\}^*\}$

$L_2 = \{w\#w^R \mid w \text{ belongs } \{0,1\}^*\}$,where # is a special symbol

$L_3 = \{ww \mid w \text{ belongs } \{0,1\}^*\}$

Which one of the following is TRUE?

a) L_1 is a deterministic CFL b) L_2 is a deterministic CFL

c) L_3 is a CFL, but not a deterministic CFL d) L_3 is a deterministic CFL

4. If L1 is context free language and L2 is a regular language which of the following is/are false? [ ] **[Gate 1999]**
   a) L1-L2 is not context free b) L1 ∩ L2 is context free
   c) ~L1 d) ~L2

5. Let L_D be the set of all languages accepted by a PDA by final state and L_E the set of all languages accepted by empty stack. Which of the following is true? [ ] **[Gate 1999]**

   a) $L_D = L_E$ b) $L_D \subset L_E$ c) $L_E \supset L_D$ d) None of the above

6. Context-free languages are closed under: [ ] **[Gate1998]**

   a) Union, Intersection b) Union, Kleene closure
   c) Intersection, complement d) Complement, Kleene closure

UNIT VI

Objective:

To understand and design Turing Machines for the given recursively enumerable languages.

Syllabus:

Turing Machine: Turing Machine, model, Design of TM, Types of Turing Machines, Computable functions, Recursively enumerable languages, church's hypothesis.

Computability Theory: Decidability of problems, universal Turing Machine, Undecidability of posts correspondence problem, Turing reducibility, definition of P and NP problems, NP complete and NP hard problems.

Learning Outcomes:

Students will be able to:

- understand turing machine and its model.
- design Turing Machine's for Recursively Enumerable languages.
- define P and NP class of problems.
- define decidability and undecidability of problems.

Learning Material

Turing Machine:

A Turing machine (TM) is denoted by

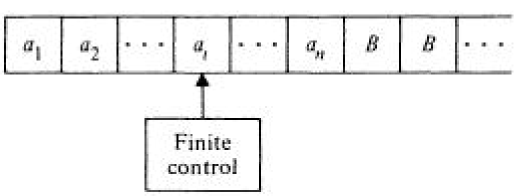
$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F),$$

where

Q is the finite set of *states*,
Γ is the finite set of allowable *tape symbols*,
B, a symbol of Γ, is the *blank*,
Σ, a subset of Γ not including B, is the set of *input symbols*,
δ is the *next move function*, a mapping from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$ (δ may, however, be undefined for some arguments),
q_0 in Q is the *start state*,
$F \subseteq Q$ is the set of *final states*.

The Turing Machine Model:

- The basic model has a finite control, an input tape that is divided into cells, and a tape head that scans one cell of the tape at a time.
- The tape has a leftmost cell but is infinite to the right. Each cell of the tape may hold exactly one of a finite number of tape symbols.
- Initially, the n leftmost cells, for some finite n ≥ 0, hold the input, which is a string of symbols chosen from a subset of the tape symbols called the input symbols.
- The remaining infinity of cells each hold the blank, which is a special tape symbol that is not an input symbol.



Moves of Turning Machine

In one move the Turing machine, depending upon the symbol scanned by the tape head and the state of the finite control,

1) changes state,

2) prints a symbol on the tape cell scanned, replacing what was written there, and

3) moves its head left or right one cell.

Note : The difference between a Turing machine and a two-way finite automaton lies in the former's ability to change symbols on its tape.

Instantaneous description (ID):

- Instantaneous description of the Turing machine M is denoted by $\alpha 1 q \alpha 2$.
- Here q, the current state of M, is in Q; $\alpha 1\alpha 2$ is the string in Γ^* that is the contents of the tape up to the rightmost nonblank symbol or the symbol to the left of the head, whichever is rightmost. (Observe that the blank B may occur in $\alpha 1\alpha 2$.).
- The tape head is assumed to be scanning the leftmost symbol of $\alpha 2$, or if $\alpha 2 = \boldsymbol{\varepsilon}$, the head is scanning a blank.

Acceptance by Turning Machine

The language accepted by M, denoted L(M), is the set of those words in $\sum^*$ that cause M to enter a final state when placed, justified at the left, on the tape of M, , with M in state q0, and the tape head of M at the leftmost cell.

Formally, the language accepted by M = $(Q,\sum,\Gamma,\delta,q0,B,F)$ is

$$\{w \mid w \text{ in } \Sigma^* \text{ and } q_0 w \overset{*}{\vdash} \alpha_1 p \alpha_2 \text{ for some } p \text{ in } F, \text{ and } \alpha_1 \text{ and } \alpha_2 \text{ in } \Gamma^*\}.$$

Example:

Design a TM to accept the language $L = \{0^n1^n \mid n\geq1\}$.

Initially, the tape of M contains 0^n1^n followed by infinity of blanks.

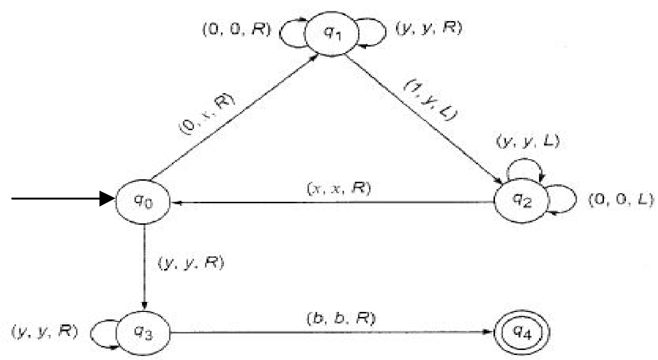
Repeatedly, M replaces the leftmost 0 by X, moves right to the leftmost 1, replacing it by Y,, moves left to find the rightmost X, then moves one cell right to the leftmost 0 and repeats the cycle.

If, however, when searching for a 1, M finds a blank instead, , then M halts without accepting.

If, after changing a 1 to a Y, M finds no more 0's, then M checks that no more 1's remain, accepting if there are none.

State	0	1	Symbol X	Y	B
q_0	(q_1, X, R)	—	—	(q_3, Y, R)	—
q_1	$(q_1, 0, R)$	(q_2, Y, L)	—	(q_1, Y, R)	—
q_2	$(q_2, 0, L)$	—	(q_0, X, R)	(q_2, Y, L)	—
q_3	—	—	—	(q_3, Y, R)	(q_4, B, R)
q_4	—	—	—	—	—

The function δ



Transition Diagram

String Verfication by Turning Machine

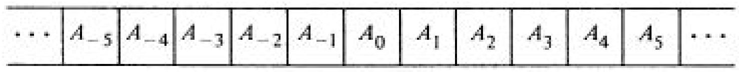
$$
\begin{array}{llll}
q_0 0011 \vdash Xq_1 011 & \vdash X0q_1 11 & \vdash Xq_2 0Y1 & \vdash \\
q_2 X0Y1 \vdash Xq_0 0Y1 & \vdash XXq_1 Y1 & \vdash XXYq_1 1 & \vdash \\
XXq_2 YY \vdash Xq_2 XYY & \vdash XXq_0 YY & \vdash XXYq_3 Y & \vdash \\
XXYYq_3 \vdash XXYYBq_4 & & &
\end{array}
$$

A computation of M

Types of Turing Machines:

i) Two-way infinite tape:

A Turing machine with a two-way infinite tape is denoted by M = (Q,∑,Γ,δ,q0,B,F). As its name implies, the tape is infinite to the left as well as to the right. We denote an ID of such a device as for the one-way infinite TM. We imagine, however, that there is an infinity of blank cells both to the left and right of the current nonblank portion of the tape.



ii) <u>Multitape Turing machines</u>:

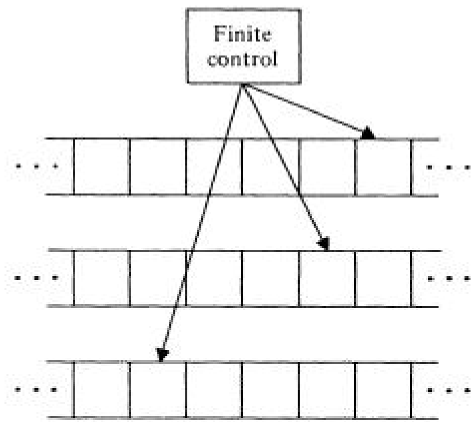
A multitape Turing machine consists of a finite control with k tape heads and k tapes; each tape is infinite in both directions.. On a single move, depending on the state of the finite control and the symbol scanned by each of the tape heads, the machine can:

1) change state;

2) print a new symbol on each of the cells scanned by its tape heads;

3) move each of its tape heads,, independently, one cell to the left or right, or keep it stationary.

Initially, the input appears on the first tape, and the other tapes are blank.



iii) <u>Nondeterministic Turing machines</u>:

A nondeterministic Turing machine is a device with a finite control and a single, one-way infinite tape. For a given state and tape symbol scanned by the tape head, the machine has a finite number of choices for the next move. Each choice consists of a new state, a tape symbol to print, and a direction of head motion. Note that the nondeterministic TM is not permitted to make a move in which the next state is selected from one choice, and the symbol printed and/or direction of head motion are selected from other choices. The nondeterministic TM accepts its input if any sequence of choices of moves leads to an accepting state.

iv) Multidimensional Turing machines:

The device has the usual finite control, but the tape consists of a k-dimensional array of cells infinite in all 2k directions, for some fixed k. Depending on the state and symbol scanned, the device changes state, prints a new symbol, and moves its tape head in one of 2k directions, either positively or negatively, along one of the k axes. Initially, the input is along one axis, and the head is at the left end of the input. At any time, only a finite number of rows in any dimension contain nonblank symbols, and these rows each have only a finite number of nonblank symbols.

B	B	B	a_1	B	B	B
B	B	a_2	a_3	a_4	a_5	B
a_6	a_7	a_8	a_9	B	a_{10}	B
B	a_{11}	a_{12}	a_{13}	B	a_{14}	a_{15}
B	B	a_{16}	a_{17}	B	B	B

v) Multihead Turing machines:

A k-head Turing machine has some fixed number,,k, of heads. The heads are numbered 1 through k,, and a move of the TM depends on the state and on the symbol scanned by each head.. In one move, the heads may each move independently left,, right, or remain stationary.

vi) Off-line Turing machines:

An off-line Turing machine is a multitape TM whose input tape is read-only. Usually we surround the input by endmarkers, ⊄ on the left and $ on the right. The Turing machine is not allowed to move the input tape head off the region between ⊄ and $.

Recursive function: a function which calls itself directly or indirectly and terminates after finite number of steps.

Total recursive function

- A function is called total recursive function if it is defined for all its arguments.
- Let f(a1,a2.....,a) be a function and defined on function g(b1,b2,....,bm), then f is total function if every element of f is assigned to some unique element of function g.

- From the definition it is clear that total recursive function is the subset of partial recursive function.
- All those partial functions for which TM halts are called total recursive functions.

Partial recursive function

- A function is called partial recursive function if it is defined for some of its arguments.
- Let f(a1,a2.....,a) be a function and defined on function g(b1,b2,....,bm), then f is partial function if some elements of f is assigned to almost one element of function g.
- Partial recursive function are turing computable.It means that there exist a turing machine for every partial recursive function.

Recursively enumerable languages

A language that is accepted by a Turing machine is said to be recursively enumerable (r.e.).

- Recursively enumerable languages are equivalent to the class of partial recursive functions.

Recursive Language:
A subclass of the r.e. sets, called the recursive sets, which are those languages accepted by at least one Turing machine that halts on all inputs.

Church's Hypothesis:
The assumption that the intuitive notion of "computable function" can be identified with the class of partial recursive functions is known as Church's hypothesis or the Church-Turing thesis.

Decidable and undecidable problems:

- A problem whose language is recursive is said to be decidable.
- A problem is undecidable if there is no algorithm that takes as input an instance of the problem and determines whether the answer to that instance is "yes" or "no."

Post's Correspondence Problem:

An instance of Post's Correspondence Problem (PCP) consists of two lists, A = w1,, ..., wk and B = x1, ... , xk, of strings over some alphabet ∑. This instance of PCP has a solution if there is any sequence of integers i1, i2, .. ., im, with m ≥1, such that ,wi1, wi2,…, wim = xi1, xi2,… xim

.

The sequence i1,…,im is a solution to this instance of PCP.

Example 1:

Let ∑ = {0, 1}. Let A and B be lists of three strings each, as defined

	List A	List B
i	w_i	x_i
1	1	111
2	10111	10
3	10	0

In this case PCP has a solution. Let m = 4, i1 = 2,i2 =1, i3 = 1, and i4 = 3. Then W2W1W1W3 = X2X1X1X3 = 101111110.

Example 2: Show that PCP problem with 2 lists

X=(b,bab^3,ba) and y=(b^3,ba,a) has a solution.

Given lists are x=(b,bab^3,ba) y=(b^3,ba,a)

The instances of PCP is as follows

	List X	List Y
i	X_i	Y_i
1	a	b^3
2	bab^3	ba
3	ba	a

In this case PCP is as follows

$X2x1x1x3=y2y1y1y3=bab^3bbba$

The solution sequence is 2113 PCP has a solution.

Example 3: Prove that PCP with two lists X =(01,1,1) Y=(0101,10,11) has no solution.

sol) Instance of PCP is given as

	List X	List Y
i	X_i	Y_i
1	01	0101
2	1	10
3	1	11

Where X1=01 Y1=0101

X2=1 Y2=10

X3=1 Y3=11

For any i $|X_i| < |Y_i|$The last Y is having strings of greater lengths. So to get same string for same sequences of x_1,x_2,x_3 and y_1,y_2,y_3 is difficult.

We cannot get solution sequence. Therefore the given PCP is having no solution.

Turing Reducibility:

Language Ll is reduced to L2 by finding an algorithm that mapped strings in L1 to strings in L2 and strings not in L1 to strings not in L2. This notion of reducibility is often called many-one reducibility.

A more general technique is called Turing reducibility, and consists simply of showing that L1 is recursive in L2.

If L1 is many-one reducible to L2, then surely L1 is Turing-reducible to L2.

P and NP problems:

The languages recognizable in deterministic polynomial time form a natural and important class, the class $U_{i\geq1}$ DTIME(n^i), which we denote by P. It is an intuitively appealing notion that P is the class of problems that can be solved efficiently.

There are a number of important problems that do not appear to be in P but have efficient nondeterministic algorithms. These problems fall into the class $U_{i\geq1}$ NTIME(n^i), which we denote by NP.

NP complete and NP hard problems:

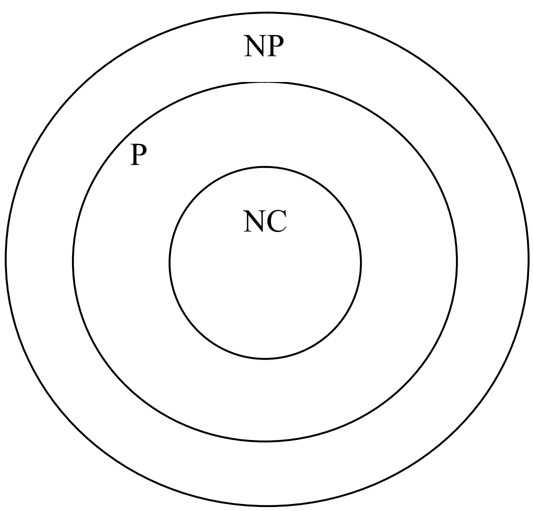
Let l be a class of languages.

A language L is complete for l with respect to polynomial-time reductions if L is in l, and every language in l is polynomial-time reducible to L.

L is NP-complete if L is complete for NP with respect to log-space reductions.

L is hard for l with respect to polynomial-time reductions if every language in l is polynomial-time reducible to L, but L is not necessarily in l.

L is NP-hard if L is hard for NP with respect to log-space reductions.



HALTING PROBLEM

The problem of determining whether a program halts on a given input is undecidable.This is to say that no program can correctly code halts.There is no algorithm for deciding halting problem

Halting problem is simply not solvable.

Let K_0= Turing acceptable language.

A problem that can be solved by an algorithm is called solvable.
A problem that cannot be solved by an algorithm called unsolvable.
An algorithm that solves a problem is called a decision procedure.
The most famous of the unsolvable problems is the problems described by Ko.It is generally called halting problem for turing machine to determine for arbitrary given turing machine M and input w,whether M will eventually halt on input W.

Closure properties of recursive languages

- **Union**: If L_1 and If L_2 are two recursive languages, their union $L_1 \cup L_2$ will also be recursive because if TM halts for L_1 and halts for L_2, it will also halt for $L_1 \cup L_2$.
- **Concatenation:** If L_1 and If L_2 are two recursive languages, their concatenation $L_1.L_2$ will also be recursive.
- **Kleene Closure:** If $L_1$is recursive, its kleene closure L_1^* will also be recursive.
- **Intersection and complement**: If L_1 and If L_2 are two recursive languages, their intersection $L_1 \cap L_2$ will also be recursive.

Closure properties of recursively enumerable languages

- Recursively enumerable languages are not closed under complementation
- If L is recursively enumerable language, its kleene closure L^* will also be recursively enumerable language.
- If L_1 and If L_2 are two recursively enumerable languages, their concatenation $L_1.L_2$ will also be recursively enumerable languages.
- If L_1 and If L_2 are two recursively enumerable languages, their union $L_1 \cup L_2$ will also be recursively enumerable languages.
- If L_1 and If L_2 are two recursively enumerable languages, their union $L_1 \cap L_2$ will also be recursively enumerable languages.

Assignment-Cum-Tutorial Questions

A. Questions testing the understanding / remembering level of students

I) Objective Questions

1. The move function of Turing Machine is ________________.
2. The language accepted by a Turning machine is called ________________language.
3. Recursively enumerable languages are equivalent to the class of ________________ functions.
4. Recursively enumerable languages are closed under complementation. [True |False]
5. The set of all recursive languages is a subset of the set of all recursively enumerable languages. [True |False]
6. Phrase structured languages are accepted by TM. [True | False]
7. The power of Non-deterministic Turning machine and deterministic Turning Machine are same. [True | False]
8. A problem whose language is recursive is called _________________________.
9. Recursive languages are [ ]
   a. a). A proper subset of CFL b). Always recognizable by PDA
   b. c). Also called Type 0 languages d). Recognizable by TM
10. Phrase structured languages are also called as Type 0 languages. [True |False]

II) Descriptive questions

1. Define Turning Machine. Explain about model of Turning Machine
2. Explain about types of turing machines.
3. Write short notes on halting problem of a Turing Machine.
4. Discuss Church's Hypothesis?
5. Write short notes on P and NP problems and give examples.
6. Write short notes on NP Complete and NP hard problems and give examples.
7. Discuss in details about Turing Reducibility.
8. List properties of recursive and recursively enumerable languages.
9. What is post correspondence problem? Explain with an example

B. Question testing the ability of students in applying the concepts.

I) Multiple Choice Questions:

1. Which of the following languages are accepted by a Turning Machine? [ ]

(i) L= $\{a^n b^n \mid n>=0\}$

(ii) L= $\{a^n b^{2n} c^{2n} \mid n>=0\}$

(iii) The set of palindromes over alphabet {a,b}

a) Only (i) b) Only (ii) c) (i) and (iii) d) (i), (ii) and (iii)

2. A single tape Turing Machine M has three states q0, q1 and q2, of which q0 is the starting state. The tape alphabet of M is {0, 1, B} and its input alphabet is {0, 1}. The symbol B is the blank symbol used to indicate end of an input string. The transition function of M is described in the following table

	0	**1**	**B**
q0	**q_0,1,R**	**q_0,0,R**	**q_1,B,L**
q1	**q_1,0,L**	**q_1,1,L**	**q2,B,R**

Which of the following statements is true about M ? [ ]

a) M halts after computing 1's complement of a binary number

b) M halts after computing 2's complement of a binary number

c) M halts after reversing of a binary number

d) None

3. A single tape Turing Machine M has four states q0, q1, q2 and q3, of which q0 is the starting state. The tape alphabet of M is {0, 1, B} and its input alphabet is {0, 1}. The symbol B is the blank symbol used to indicate end of an input string. The transition function of M is described in the following table

	0	**1**	**B**
q0	**q0,0,R**	**q0,1,R**	**q1,B,L**
q1	**q1,0,L**	**q2,1,L**	
q2	**q2,1,L**	**q2,0,L**	**q3,B,R**

Which of the following statements is true about M ? [ ]

a. M halts after computing 1's complement of a binary number

b. M halts after computing 2's complement of a binary number

c. M halts after reversing of a binary number

d. None

4. The given table represents a Turing machine which accepts [ ]

Present state	1
$\rightarrow q_1$	bq_2R
q_2	bq_1R

a) even number of 1's

b) odd number of 1's

c) even number of 1's and odd number of 1's

d) even number of 1's or odd number of 1's

5. The transitions of a Turing Machine are given below [ ]

δ (q0,1) = (q0,1,R)

δ (q0,B) = (q1,1,R)

δ (q1,B) = (q2,B,R)

The input on the tape is $q_0$11B then the output on the tape is [ ]

a) 111Bq_2B b) 1111Bq_2B c) 111Bq_1B d) 1111Bq_1B

II) Problems

1. Design TM for the language $L=\{a^n b^n c^n \mid n>=1\}$

2. Design TM for the language $L = \{ a^n b^m c^{n+m} \mid n,m >= 1 \}$

3. Design a Turing machine that accepts the language $L =\{ WW^R / W \in (0+1)^*$ and W^R is reverse of $W\}$

4. Consider the TM described by the transition table given below. Represent the processing of

a) 011 b) 0011 using ID’s. Which of the strings are accepted by TM?

Present state	0	1	x	y	b
$\rightarrow q_1$	xRq_2				bRq_5
q_2	$0Rq_2$	yLq_3		yRq_2	
q_3	$0Lq_4$		xRq_5	yLq_3	
q_4	$0Lq_4$		xRq_1		
q_5				$yxRq_5$	bRq_6
q_6					

5. Design TM for subtraction of two numbers.

6. Show that the following post correspondence problem has a solution and give the solution.

i	ListA	ListB
1	11	11
2	100	001
3	111	11

C. GATE/NET/SLET

1. Which of the following statements is/are FALSE? **GATE CS 2013** [ ]
   1. For every non-deterministic Turing machine, there exists an equivalent deterministic Turing machine.
   2. Turing recognizable languages are closed under union and complementation.
   3. Turing decidable languages are closed under intersection and complementation.
   4. Turing recognizable languages are closed under union and intersection.

   a)1 and 4 only b) 1 and 3 only c) 2 only d)3 only

2. Which of the following is true for the language $\{a^p \mid p \text{ is a prime}\}$? **GATE CS 2008** [ ]

a) It is not accepted by a Turing Machine

b) It is regular but not context-free

c) It is context-free but not regular

d) It is neither regular nor context-free, but accepted by a Turing machine

3. Let L1 be a recursive language. Let L2 and L3 be languages that are [ ]
recursively enumerable but not recursive. Which of the following statements is not necessarily true?
(A) L2 – L1 is recursively enumerable. (B) L1 – L3 is recursively enumerable
(C) L2 ∩ L1 is recursively enumerable (D) L2 ∪ L1 is recursively enumerable
GATE CS 2010
a)A b)B c)C d)D

4. If L and L' are recursively enumerable, then L is **GATE CS 2008** [ ]
a) regular b) context-free
c) Context-sensitive d) recursive

5. Let L1 be a recursive language, and let L2 be a recursively enumerable but not a recursive language. Which one of the following is TRUE? **GATE-CS-2005** [ ]
L1' --> Complement of L1
L2' --> Complement of L2
a) L1' is recursive and L2' is recursively enumer-able
b) L1' is recursive and L2' is not recursively enumerable
c) L1' and L2' are recursively enumerable
d) L1' is recursively enumerable and L2' is recursive

6. Consider the following types of languages: **GATE-CS-2016 (Set 2)**
L1 Regular, L2: Context-free,
L3: Recursive, L4: Recursively enumerable.
Which of the following is/are TRUE? [ ]
I. L3' U L4 is recursively enumerable II. L2 U L3 is recursive
III. L1* U L2 is context-free IV. L1 U L2' is context-free
a) I only b) I and III only c) I and IV only d) I, II and III only

7. A single tape Turing Machine M has two states q0 and q1, of which q0 is the starting state. The tape alphabet of M is {0, 1, B} and its input alphabet is {0, 1}. The symbol B is the blank symbol used to indicate end of an input string. The transition function of M is described in the following table
GATE-CS-2003 [ ]

	0	1	B
q0	q1, 1, R	q1, 1, R	Halt
q1	q1, 1, R	q0, 1, L	q0, B, L

The table is interpreted as illustrated below. The entry (q1, 1, R) in row q0 and column 1 signifies that if M is in state q0 and reads 1 on the current tape square, then it writes 1 on the same tape square, moves its tape head one position to the right and transitions to state q1. Which of the following statements is true about M ?

a) M does not halt on any string in (0+1)+
b) M does not halt on any string in (00+1)+
c) M halts on all string ending in a 0
d) M halts on all string ending in a 1

8. Which of the following is true? **GATE-CS-2002** [ ]

a) The complement of a recursive language is recursive.
b) The complement of a recursively enumerable language is recursively enumerable.
c) The complement of a recursive language is either recursive or recursively enumerable.
d) The complement of a context-free language is context-free

9. Define languages L0 and L1 as follows : **GATE-CS-2003** [ ]

L0 = {< M, w, 0 > | M halts on w}

L1 = {< M, w, 1 > | M does not halts on w}

Here < M, w, i > is a triplet, whose first component. M is an encoding of a Turing Machine, second component, w, is a string, and third component, i, is a bit. Let L = L0 ∪ L1. Which of the following is true?

a) L is recursively enumerable, but L' is not
b) L' is recursively enumerable, but L is not
c) Both L and L' are recursive
d) Neither L nor L' is recursively enumerable

10. Nobody knows yet if P = NP. Consider the language L defined as follows: $L = \begin{cases} (0+1)^* \text{ if } P = NP \\ \phi \text{ otherwise} \end{cases}$

GATE-CS-2003 [ ]

Which of the following statements is true ?

a) L is recursive
b) L is recursively enumerable but not recursive
c) L is not recursively enumerable
d) Whether L is recursive or not will be known after we find out if P = NP