1657025286509 Theory-of-Automata-and-Formal-Languages-VIP-Handwritten-Notes-by-Kulbhushan - Pro- unlocked

Theory of Automata and Formal Languages (Dr. A.P.J. Abdul Kalam Technical University)

THEORY OF AUTOMATA & FORMAL LANGUAGES

HANDWRITTEN NOTES

VIP

AKTU | SEM-4

KKWORLD.IN | KRAZY KAKSHA

Theory of Automata and Formal Languages (KCS402)		
Course Outcome (CO)		**Bloom's Knowledge Level (KL)**
At the end of course , the student will be able to understand		
CO 1	Analyse and design finite automata, pushdown automata, Turing machines, formal languages, and grammars	K_4, K_6
CO 2	Analyse and design, Turing machines, formal languages, and grammars	K_4, K_6
CO 3	Demonstrate the understanding of key notions, such as algorithm, computability, decidability, and complexity through problem solving	K_1, K_5
CO 4	Prove the basic results of the Theory of Computation.	K_2,K_3
CO 5	State and explain the relevance of the Church-Turing thesis.	K_1, K_5

DETAILED SYLLABUS		**3-1-0**
Unit	**Topic**	**Proposed Lecture**
I	**Basic Concepts and Automata Theory:** Introduction to Theory of Computation- Automata, Computability and Complexity, Alphabet, Symbol, String, Formal Languages, Deterministic Finite Automaton (DFA)- Definition, Representation, Acceptability of a String and Language, Non Deterministic Finite Automaton (NFA), Equivalence of DFA and NFA, NFA with ε-Transition, Equivalence of NFA's with and without ε-Transition, Finite Automata with output- Moore Machine, Mealy Machine, Equivalence of Moore and Mealy Machine, Minimization of Finite Automata, Myhill-Nerode Theorem, Simulation of DFA and NFA	**08**
II	**Regular Expressions and Languages:** Regular Expressions, Transition Graph, Kleen's Theorem, Finite Automata and Regular Expression- Arden's theorem, Algebraic Method Using Arden's Theorem, Regular and Non-Regular Languages- Closure properties of Regular Languages, Pigeonhole Principle, Pumping Lemma, Application of Pumping Lemma, Decidability- Decision properties, Finite Automata and Regular Languages, Regular Languages and Computers, Simulation of Transition Graph and Regular language.	**08**
III	**Regular and Non-Regular Grammars**: Context Free Grammar(CFG)-Definition, Derivations, Languages, Derivation Trees and Ambiguity, Regular Grammars-Right Linear and Left Linear grammars, Conversion of FA into CFG and Regular grammar into FA, Simplification of CFG, Normal Forms- Chomsky Normal Form(CNF), Greibach Normal Form (GNF), Chomsky Hierarchy, Programming problems based on the properties of CFGs.	**08**
IV	**Push Down Automata and Properties of Context Free Languages**: Nondeterministic Pushdown Automata (NPDA)- Definition, Moves, A Language Accepted by NPDA, Deterministic Pushdown Automata(DPDA) and Deterministic Context free Languages(DCFL), Pushdown Automata for Context Free Languages, Context Free grammars for Pushdown Automata, Two stack Pushdown Automata, Pumping Lemma for CFL, Closure properties of CFL, Decision Problems of CFL, Programming problems based on the properties of CFLs.	**08**
V	**Turing Machines and Recursive Function Theory** : Basic Turing Machine Model, Representation of Turing Machines, Language Acceptability of Turing Machines, Techniques for Turing Machine Construction, Modifications of Turing Machine, Turing Machine as Computer of Integer Functions, Universal Turing machine, Linear Bounded Automata, Church's Thesis, Recursive and Recursively Enumerable language, Halting Problem, Post's Correspondance Problem, Introduction to Recursive Function Theory.	**08**

Text books:

1. Introduction to Automata theory, Languages and Computation, J.E.Hopcraft, R.Motwani, and Ullman. 2nd edition, Pearson Education Asia
2. Introduction to languages and the theory of computation, J Martin, 3rd Edition, Tata McGraw Hill
3. Elements and Theory of Computation, C Papadimitrou and C. L. Lewis, PHI
4. Mathematical Foundation of Computer Science, Y.N.Singh, New Age Internationa

UNIT - 1

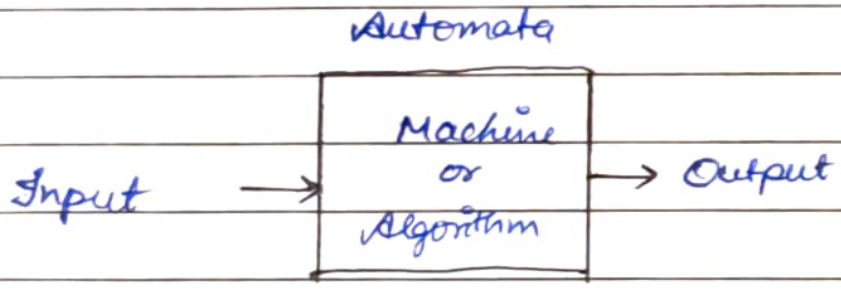
Date
Page

* Theory of Automata = Theory of computation = Theory of Computer Science = Computer theory

* Theory of Automata -
  → It is a theoretical branch of computer science and mathematics.
  → It is the study of abstract machines and the computation problem that can be solved using these machines.
  → Abstract machines are called automata.

* Automaton -
  → Singular of automata.
  → Automaton consists of states and transitions
  → State is represented by circles, and the transitions is represented by arrows.

* Automata is the kind of machine which takes some string as input and this input goes through a finite number of states and may enter in the final state.



(No Human involvement)

* Computability is about what can be computed (solvable or not).
* Complexity is about how efficient can it be computed.

* Applications of Automata Theory -
i) Automata theory is a base for formal language and these formal language are useful in programming language.
ii) Automata theory plays an important role in compiler design.
iii) To prove the correctness of the program, we use automata theory.
iv) In switching theory and design & analysis of digital circuits, automata theory is applied.
v) Automata theory deals with the design finite state machine.

* Central Concepts of Automata theory -
i) Alphabet -
→ An alphabet is a finite, non-empty set of symbols.
→ Usually it is denoted by Σ.
→ Examples -
$\Sigma = \{0, 1\}$ ⇒ Binary Alphabet
$\Sigma = \{a, b, c, \ldots, z\}$ ⇒ Set of all lower case letters

Date
Page

11) **String** –

→ It is a finite sequence of symbols chosen from some alphabet.

→ Generally denoted by w, x, y, z.

→ Example –

If $\Sigma = \{0, 1\}$ then 011, 110, 1101 are strings.

If $\Sigma = \{a, b, c \ldots z\}$ then abc, dxeyk are strings.

→ **Empty String** – String with zero occurrence of symbols, denoted by ϵ (Epsilon).

→ **Length of String** – Number of positions for symbols in string, or no. of alphabets in string.

If $w = 1011$ then $|w| = 4$.

→ **Power of an alphabet (Σ)** –

- Σ^k is set of strings of length k whose symbol is in Σ.
- If $\Sigma = \{0, 1\}$ then

$\Sigma^0 = \epsilon$

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

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

$\Sigma^3 = \{000, 001, \ldots\ldots 111\}$

Note: Σ & Σ^1 both are different

↓ ↓ Set of strings of members 0 & 1 of length 1.

Alphabet set of members 0 & 1

→ Σ^*: Set of all strings over an alphabet Σ

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots\ldots$

↳ Star Closure

→ Σ^+: Set of non-empty strings from alphabet Σ

$\Sigma^+ : \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots\ldots$

↳ Plus Closure

→ $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$

→ $\phi \neq \{\epsilon\}$

ϕ: contains no string; $\{\epsilon\}$: contains one string

→ **Concatenation of Strings** -

→ Let x & y be strings, then xy is concatenation of x & y

→ If $x = 111$ & $y = 010$, then

$xy = 111010$ & $yx = 010111$, note: $xy \neq yx$

→ If w is any string such that $\epsilon w = w\epsilon = w$ then ϵ is identity for concatenation.

Language -

→ A language is set of all strings chosen from some Σ^*, where Σ is particular alphabet.

→ If Σ is an alphabet and $L \subseteq \Sigma^*$, then L is a language over alphabet Σ.

Note: i) Σ^*: Language for any alphabet Σ.

ii) ϕ : Empty language over any alphabet.

iii) $\{\epsilon\}$: Language consisting of only empty strings is a language over any alphabet.

iv) $\phi \neq \{\epsilon\}$

→ Informal Language → Semantic Language

→ Formal Language → Syntactical Language

Finite Automata (FA)

DFA — NFA (NDFA)

* **DFA** – (Deterministic finite Automata)

→ A FA said to be deterministic if the machine goes to only one state for a particular input.

→ DFA does not accept the null move.

→ DFA is a set of five tuples and represented as

$$M = \{Q, \Sigma, \delta, q_0, F\}$$

where

Q → Finite non-empty set of states (there is atleast one state)

Σ → Finite non-empty set of input (alphabet)

δ → Transition function

$$\delta: Q * \Sigma \rightarrow Q$$

Current State, Current Input, Next State

q_0 → Initial State, (there is only one initial state & $q_0 \in Q$)

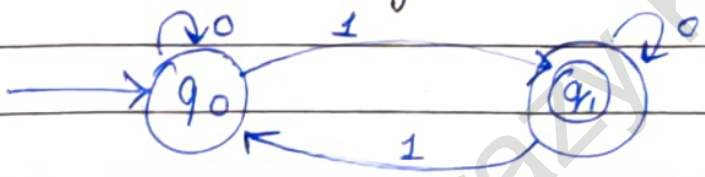
F → set of final states & $F \subseteq Q$

* Representation of DFA -

i) State transition Diagram

ii) State transition Table

i) State Transition Diagram -



Where

→ (q_0) is initial State

(q_1) is final State

ii) State Transition Table -

Q \ Σ	0	1
→ q_0	q_0	q_1
(q_1)	q_1	q_0

or

* q_1

$Q = \{q_0, q_1\}$

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

$q_0 = \{q_0\}$

$F = \{q_1\}$

$\delta(q_0, 0) \to q_0$ $\quad \delta(q_1, 0) \to q_1$

$\delta(q_0, 1) \to q_1$ $\quad \delta(q_1, 1) \to q_0$

Note: i) Total Transitions = $Q \times \Sigma$

ii) In DFA if min length of string is 'n' then min. States will be n+1.

iii) In DFA, there must be a transition for every input at every State.

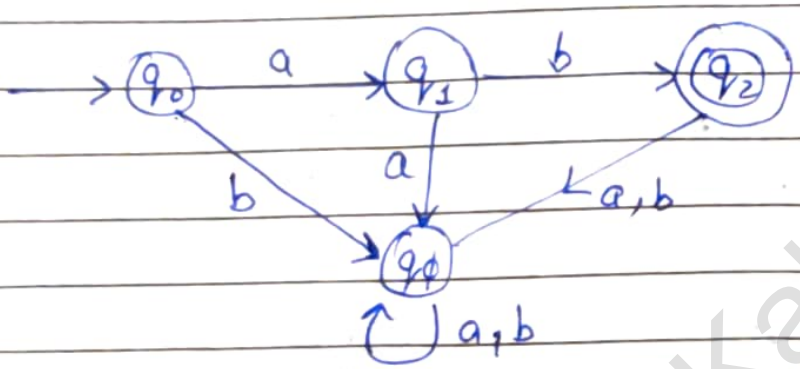
⇒ State transition table can be formed.

iv) No. of outgoing edges are equal to no. of input alphabets

* Examples of DFA

Q1 Design a DFA accept a word ab. Σ (a, b)

Sol:



Note: Dead State is denoted by q_ϕ, also called Hang or Trap State

Q2 Draw State Transition Diagram for DFA

$A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, q_2)$

(Q, Σ, q_0, F)

Where, $\delta(q_0, 0) \to q_1$

$\delta(q_0, 1) \to q_0$

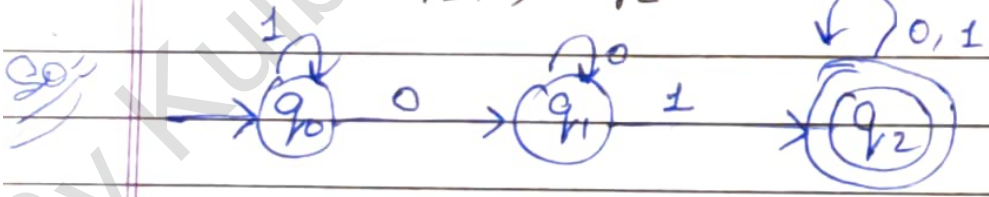
$\delta(q_1, 0) \to q_1$

$\delta(q_1, 1) \to q_2$

$\delta(q_2, 0) \to q_2$

$\delta(q_2, 1) \to q_2$

Sol:



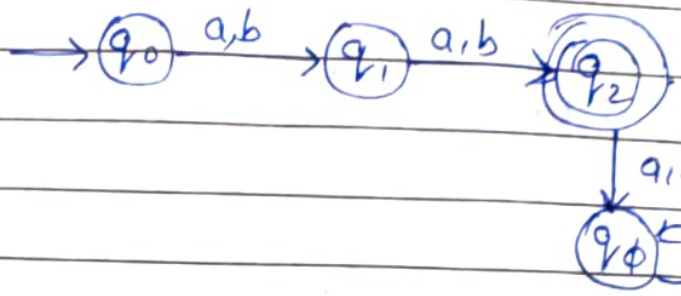
Q3- Construct a DFA that accept all string over {a, b}

i) $|\omega| = 2$

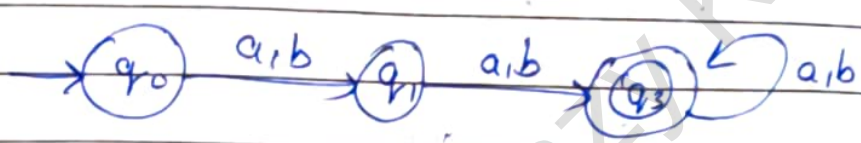
ii) $|\omega| \geq 2$

iii) $|\omega| \leq 2$

i) $|w| = 2$, $L = \{aa, ab, ba, bb\}$

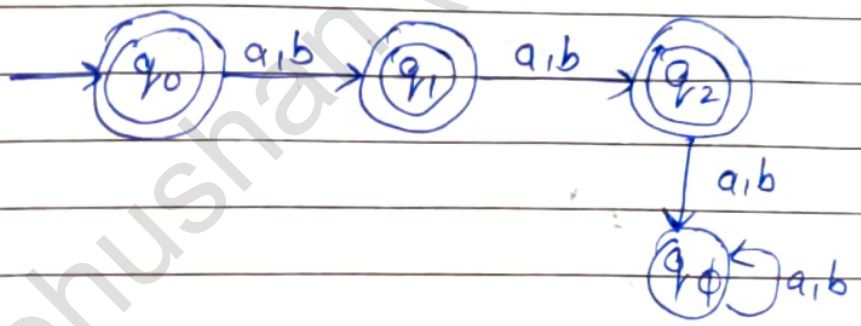


ii) $|w| \geq 2$, $L = \{aa, ab, ba, bb, aaa, abaa \ldots\}$



iii) $|w| \leq 2$, $L = \{\epsilon, a, b, ab, aa, ba, bb\}$

⇒ ~~final~~ = final state

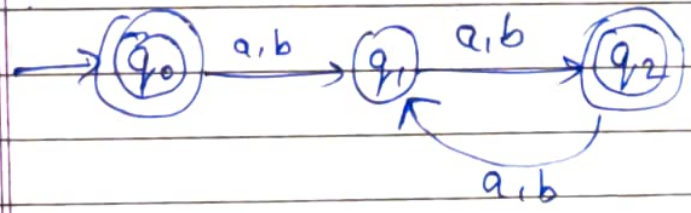


Q4- Construct DFA $w \in \{a,b\}$

i) $|w| \bmod 2 = 0$ ii) $|w| \bmod 2 = 1$

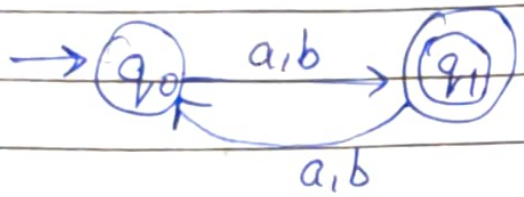
Sol= i) $|w| \bmod 2 = 0 \Rightarrow$ w is even in length

So, $L = \{\epsilon, ab, ba, aa, bb, aaaa, abab \ldots\}$



iii) $|w| \bmod 2 = 1$ ⇒ length of w is even

So, $L = \{a, b, aba, aaa, bba, \ldots\}$



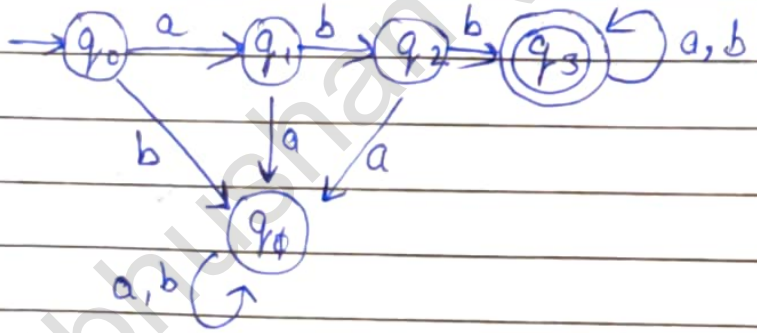
Q5- Design a DFA

i) All strings starts with abb

ii) All String ends with abb

iii) All string with abb as a substring, abb anywhere in the string.

Sol i)= $L = \{abb, abbaab, abbaa, abbbb, \ldots\}$



State Transition Table -

Q \ Σ	a	b
→ q_0	q_1	q_ϕ
q_1	q_ϕ	q_2
q_2	q_ϕ	q_3
* q_3	q_3	q_3
q_ϕ	q_ϕ	q_ϕ

$M = \{Q, \Sigma, \delta, q_0, F\}$

$Q = \{q_0, q_1, q_2, q_3, q_\phi\}$

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

$q_0 = \{q_0\}$

$F = \{q_3\}$

$\delta(q_0, a) \rightarrow q_1$

$\delta(q_0, b) \rightarrow q_\phi$

$\delta(q_1, a) \rightarrow q_\phi$

$\delta(q_1, b) \rightarrow q_2$

$\delta(q_2, a) \rightarrow q_\phi$

$\delta(q_2, b) \rightarrow q_3$

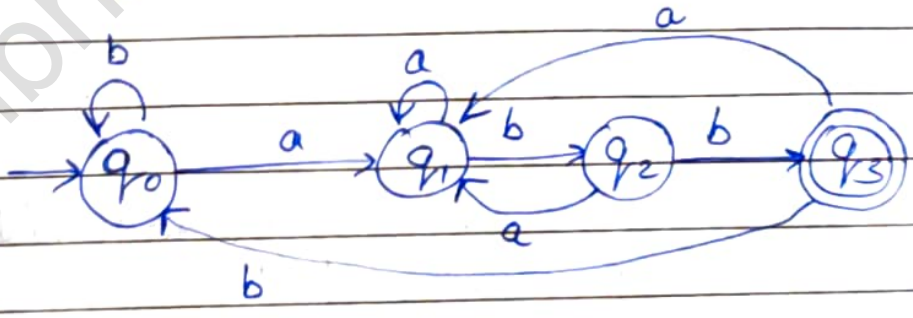
$\delta(q_3, a) \rightarrow q_3$

$\delta(q_3, b) \rightarrow q_3$

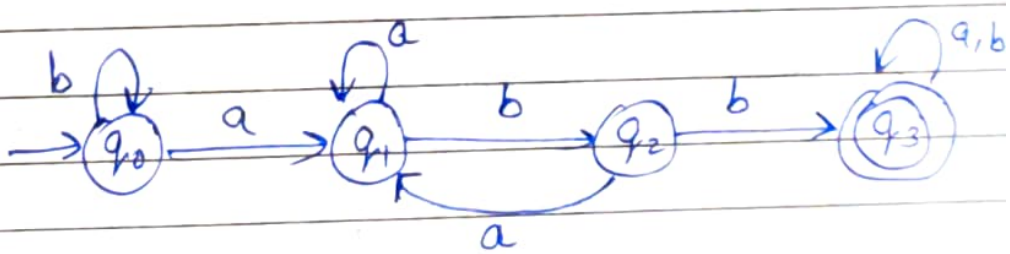
$\delta(q_\phi, a) \rightarrow q_\phi$

$\delta(q_\phi, b) \rightarrow q_\phi$

ii) $L = \{abbaaabb, babb, abb, aaabb \ldots\}$

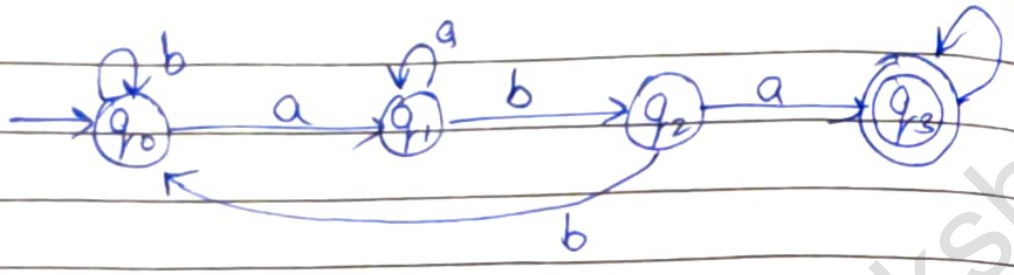


iii) $L = \{abb, aabb, bababb, \ldots\}$

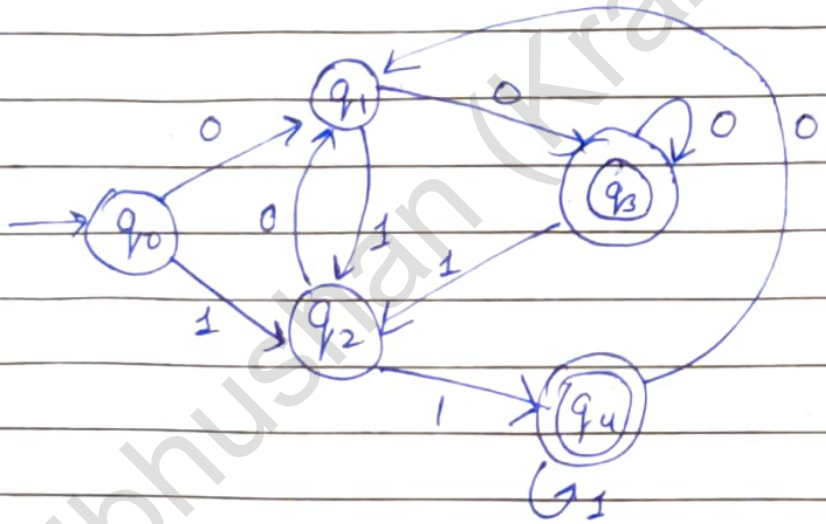


Q6 Design a DFA for language contains substring 'aba' over $\Sigma = \{a, b\}$

Sol⇒ $w = \_\_\_aba\_\_\_$

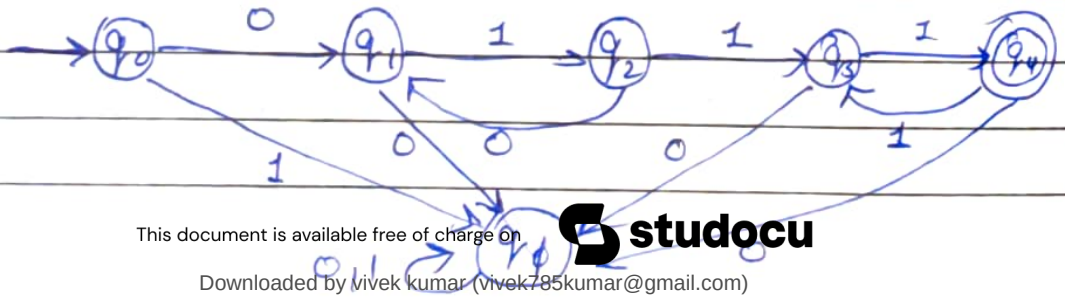


Q7- Design a DFA that consists of string over $\Sigma = \{0, 1\}$ and every string either end with 00 or 11.



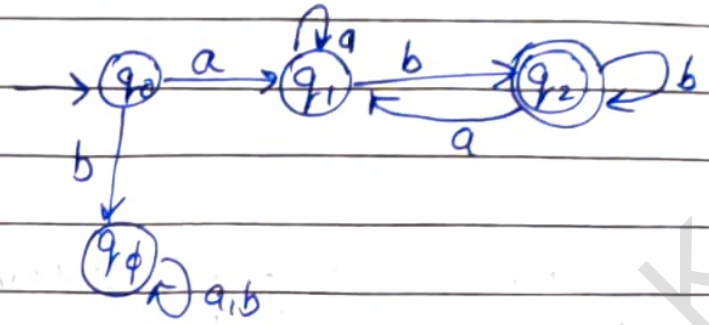
Q8- Design a DFA for language $L = \{(01)^i 1^{2j} \mid i \geq 1, j \geq 1\}$ where, $\Sigma = \{0, 1\}$

Sol⇒ $w = 01\,01\,01 \_\_\_ 11\,11\,11\,11 \_\_\_$



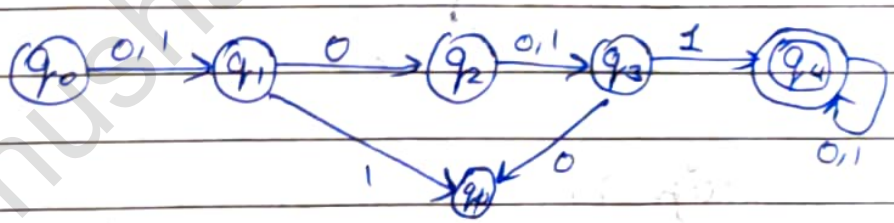
Q9- Construct a DFA which accept a language of all strings starting with 'a' and ending with 'b'.

Sol= $L = \{ab, abab, ababab, aaaabbbb \ldots\}$



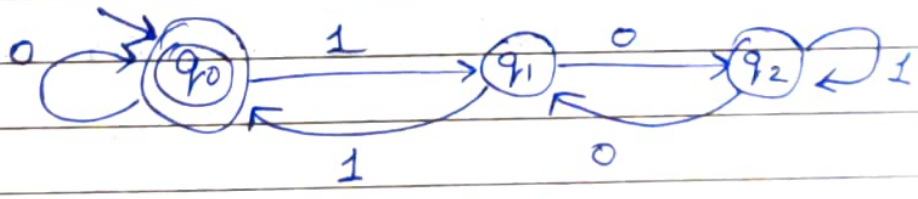
Q10- Design a DFA which accepts all strings over (0,1) in which 2nd symbol is '0' & 4th symbol is '1'.

Sol $w = \overset{0/1}{\_}\, 0 \overset{0/1}{\_}\, 1$



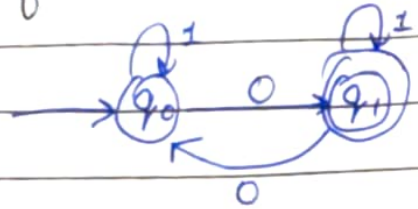
Q11- Construct a DFA which accept a language of all binary strings divisible by 3 over $\Sigma(0,1)$.

Sol=
$q_0 \rightarrow 0$
$q_1 \rightarrow 1$
$q_2 \rightarrow 2 = (10)_2$
} Remainder, So Remainder set of 3 = {0,1,2}

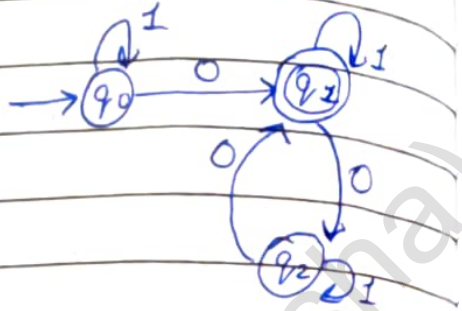


Q12 Design a DFA for language having odd no. of '0' where $\Sigma = \{0, 1\}$

Sol=



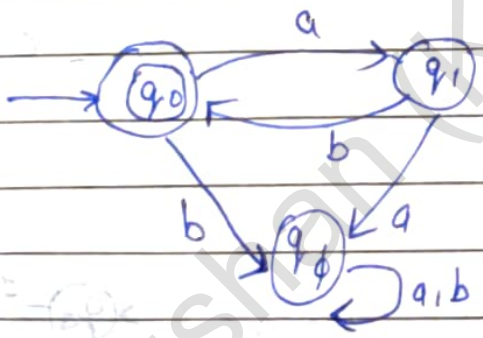
or



Q13- Design a DFA for language $L = \{(ab)^n : n \geq 0\}$

Sol= $L = \{\in, ab, abab, ababab, ------\}$

Initial state = Final state



Q14- Design DFA which accepts set of Binary strings divisible by 3.

Sol Remainder Set of 3 = {0, 1, 2}

$0 \rightarrow q_0$, $1 \rightarrow q_1$, $2 \rightarrow q_2$

$2 * \text{state} + \Sigma$

Bz Binary $\{0, 1\}$

$\delta(q_0, 0) = 2 \times \text{State} + \Sigma = 2 \times 0 + 0 = 0 \Rightarrow q_0$

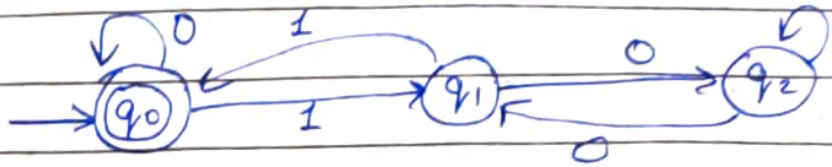
$\delta(q_1, 0) = 2 \times 1 + 0 = 2 \rightarrow q_2$

$\delta(q_2, 0) = 2 \times 2 + 0 = 4 \rightarrow 1 \rightarrow q_1$

$\delta(q_0, 1) = 2 \times 0 + 1 = 1 \rightarrow q_1$

$\delta(q_1, 1) = 2 \times 1 + 1 = 3 \rightarrow 0 \rightarrow q_0$

$\delta(q_2, 1) = 2 \times 2 + 1 = 5 \rightarrow 2 \rightarrow q_2$



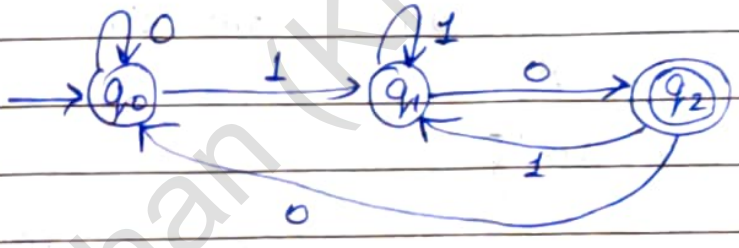
Q15- Design a DFA for a language of string 0 and 1 that.

i) Ends with 10

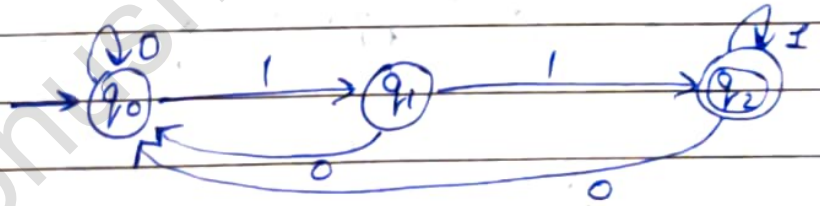
ii) Ends with 11

iii) Ends with 1

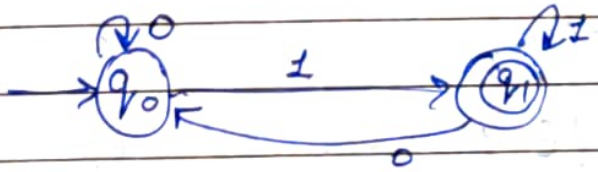
Sol i)



ii)

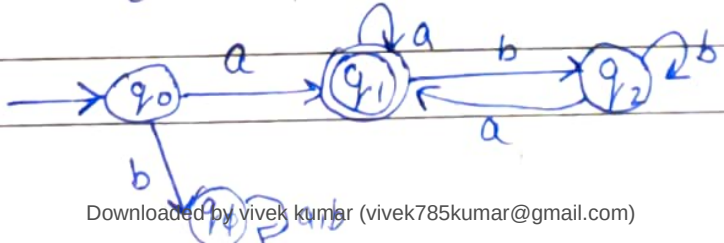


iii)



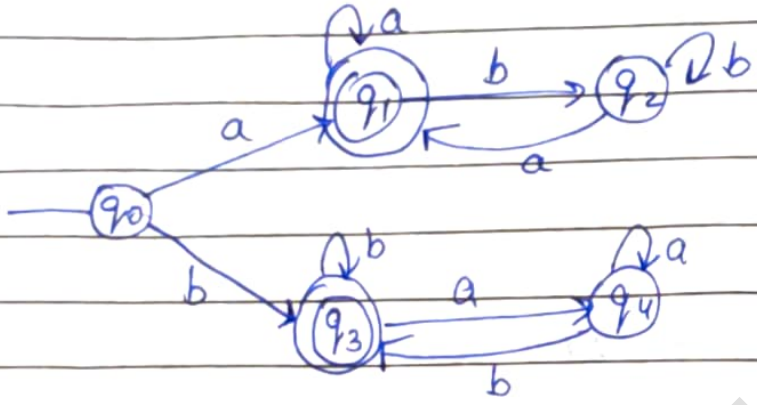
Q16- Design a DFA which accepts the language of string ends and starts with 'a' over $\Sigma = \{a, b\}$

Sol= $L = \{a, aa, aba, aaba, abbba \text{---} \}$



Q17- Design a DFA which accept all strings which start and end with same symbol over $\Sigma = \{a, b\}$

Sol=



Q18- Design a DFA which accept all the strings -

i) With exactly two 1.

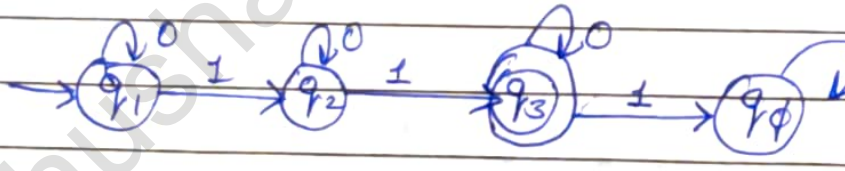
ii) Containing atleast two 0

iii) Containing atmost two 0.

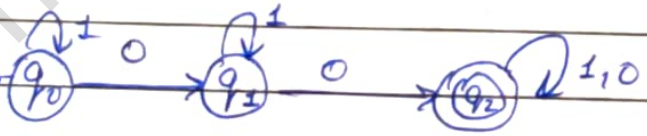
iv) Such that length of the string is divisible by 3.

v) Almost 5 of length.

Sol= i)

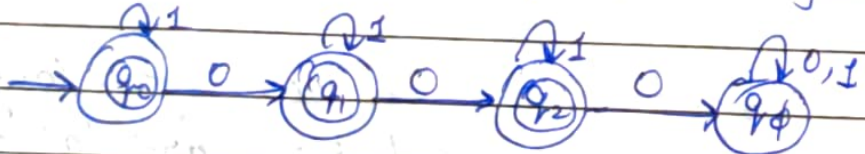


ii)

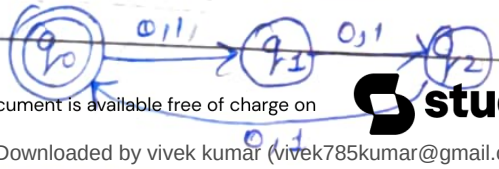


$L = \{00, 100, 010, 0001, 00000\}$

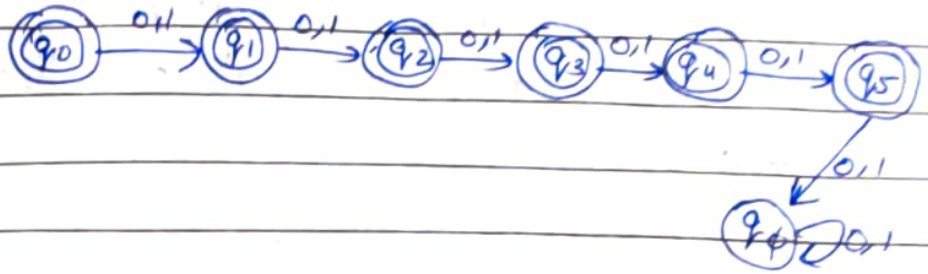
iii) $L = \{\epsilon, 1, 01, 00, 11, 1110, 001, \ldots\}$



iv) $L = \{\epsilon, 010, 111010, 110111, 000010001 \ldots\}$



v)



Q19- Design a DFA which accept all the strings containing —

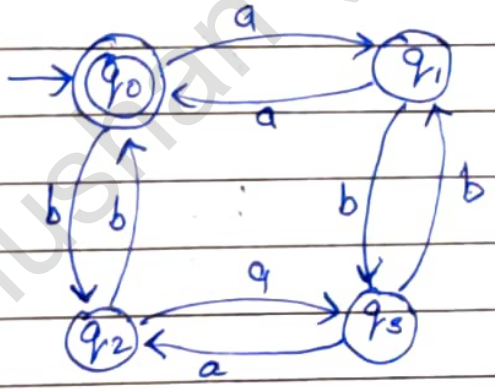
i) Even no. of a and Even no. of b.

ii) Odd no. of a and Even no. of b.

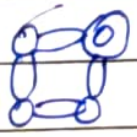
iii) Even no. of a and Odd no. of b.

iv) Odd no. of a and Odd no. of b.

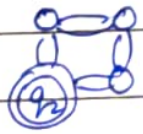
Sol i) $L = \{ \in, aa, bb, abab, aabb, bbaa, baba, abba, baab, ---- \}$



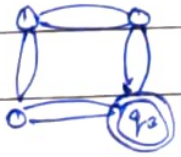
ii)



iii)

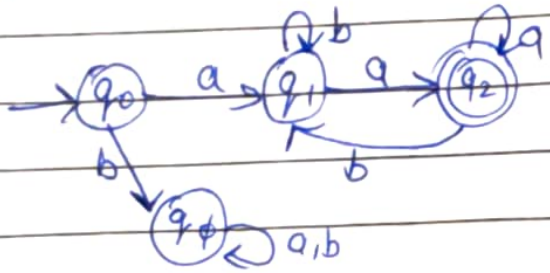


iv)

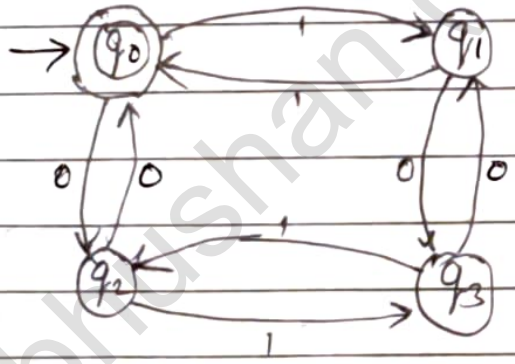


Q20 Design a DFA which accept all the string starts and ends with 'a' over $\Sigma = \{a, b\}$ and $|w| \geq 2$.

So= $L = \{aa, aba, aaa, ababa, abbba, aaba, \ldots\}$



Q21. Consider the following DFA and test whether the string "110101" is accepted by the FA represented by transition diagram –



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

So, string 110101 will be accepted as 'q_0' is the final state.

* Complement of DFA -

→ Final State changes to Non-final State

→ Non-Final State changes to final State.

* Language of DFA

→ The set of all strings that results in a sequence of state transition from start to an accepting state.

→ Now we can define the language of DFA → $M = \{Q, \Sigma, \delta, q_0, F\}$

→ Language is denoted by $L(M)$

$$L(M) = \{w | \delta(q_0, w) \text{ is in } F\}$$

→ If L is $L(M)$ for some deterministic finite automata then we can say L is a regular language.

* Properties of Transition functions.

i) $\delta(q, \Lambda) = q$

ii) For all strings w & input symbol a,

$$\delta(q, aw) = \delta(\delta(q, a), w)$$

$$\delta(q, wa) = \delta(\delta(q, w), a)$$

Q21

Date
Page

* NDFA / NFA –

→ A finite automata is said to be non-deterministic, if there is more than one possible transition from one state on the same input symbol.

→ A Non-Deterministic Finite Automata (NDFA) is a set of five tuples and represented as –

$$M = \{Q, \Sigma, \delta, q_0, F\}$$

Where

$Q \rightarrow$ Finite Non-Empty set of States.

$\Sigma \rightarrow$ Finite Non-Empty set of Inputs.

$\delta \rightarrow$ Transition function

$$\delta : Q * \Sigma \rightarrow 2^Q$$

↳ Power set of Q

e.g. $Q = \{a, b\}$

$= \{\phi, \{a\}, \{b\}, \{a, b\}\}$ for $\{a, b\}$

$q_0 \rightarrow$ Initial State

$F \rightarrow$ Set of final states. $(F \subseteq Q)$.

→ **Characteristics of NFA –**

i) One input symbol causes to move to more than one state.

ii) This is not compulsory that all the states have to consume all the input symbols

→ **Language Accepted by NFA –**

- The language L accepted by NFA $M = \{Q, \Sigma, \delta, q_0, F\}$ is defined as

$$L(M) = \{w \in \Sigma^* : \delta^*(q_0, w) \cap F \neq \phi\}$$

- In other words, language consists of all string w for which there is a walk

labelled w from initial vertex of transition graph to final vertex.

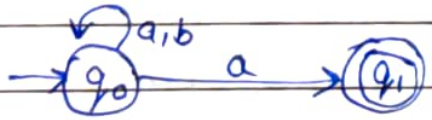
*	DFA	NDFA
i)	Dead configuration is not allowed.	i) Dead configuration is allowed.
ii)	Multiple choice are not available corresponding to an I/p.	ii) Multiple choices are available corresponding to an Input.
iii)	ϵ-Move is not allowed.	iii) ϵ move is allowed.
iv)	Digital computers are deterministic	iv) Non-deterministic feature is not associated with real computers.
v)	Designing and understanding is not easy.	v) Designing and understanding is easy.
		vi) Can convert into DFA

* Examples of NDFA

Q1- Construct NFA for the language -

$L = \{$ Ends with 'a' $\}$

Sol:-



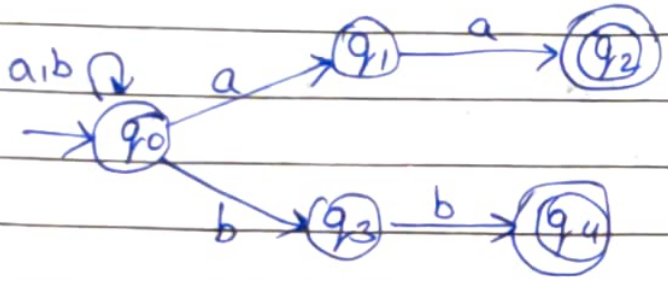
Transition Table -

	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	q_0
$*q_1$	ϕ	ϕ

↓ Dead configuration

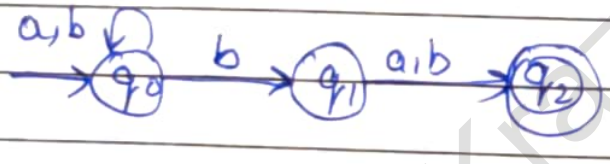
Q2- Design a NFA in which strings over {a,b} ends with aa or bb.

Sol=



Q3- Design a NFA in which strings over {a,b} has 'b' as second last symbol.

Sol=



* NFA to DFA Conversion (Equivalent DFA to NFA)

Q1- Construct DFA equivalent to NFA for

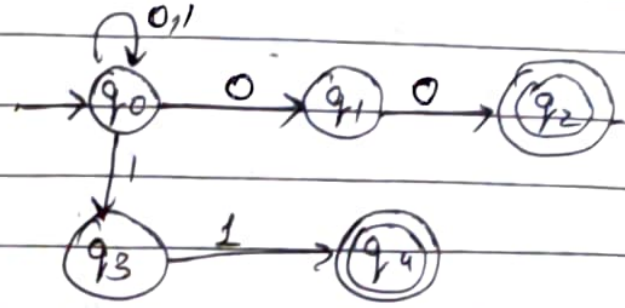
a) $m = \{\{q_0, q_1, q_2\}, \{a,b\}, \delta, q_0, \{q_2\}\}$

Σ / States	a	b
→ q_0	q_0, q_1	q_2
q_1	q_0	q_1
(q_2)	–	q_0, q_1 → Two states

Sol=

Σ / States	a	b
→ $\{q_0\}$	$\{q_0, q_2\}$	$\{q_2\}$
($\{q_2\}$)	ϕ	$\{q_0, q_1\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_1, q_2\}$ → Order doesn't matter
($\{q_1, q_2\}$)	$\{q_0\}$	$\{q_0, q_1\}$
ϕ	ϕ	ϕ → 1 State → Dead State

(ii)



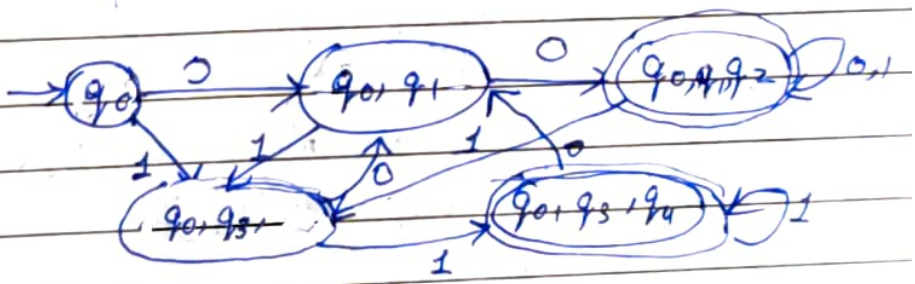
Sol:=

→	E	0	1
→	q_0	q_0, q_1	q_0, q_3
	q_1	q_2	—
	q_2	—	—
	q_3	—	q_4
	q_4	—	—

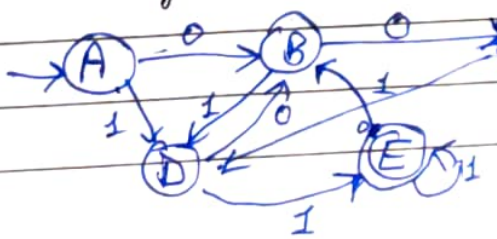
{ NDFA Transition Table

DFA Transition Table :

→	Σ	0	1
→	$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0, q_3\}$
	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_3\}$
	$\{q_0, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_3, q_4\}$
*	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_3\}$
*	$\{q_0, q_3, q_4\}$	$\{q_0, q_1\}$	$\{q_0, q_3, q_4\}$



Note: We can change the Name of the States of DFA.



* NFA with ϵ - Transitions

→ NFA with ϵ- transitions can change their state on empty input string that is without any input or ϵ as an input.

→ NFA with ϵ - moves is denoted By 5 tuples -

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

Where

Q: Finite set of states

Σ: finite set of I/P alphabet

q_0: Initial State

F: Finite set of final states.

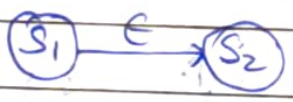
δ: Transition function -

$$\delta: Q * (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$$

* ϵ - Closure -

ϵ - Closure of q is the set of all states p such that there is a path from q_0 to p with label ϵ.

* Eliminating ϵ - Moves (ϵ - NFA to NFA conversion)



Step 1- Find all edges starting from S_2.

Step 2- Duplicate all edges to S_1 without changing edge labels.

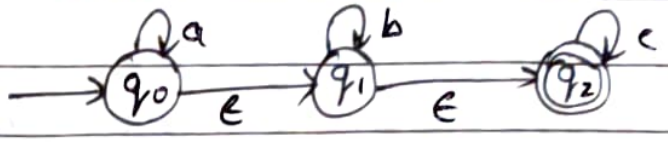
Step 3- If S_1 is Initial State, make S_2 Initial too.

Step 4- If S_2 is Final State, make S_1 final too.

Step 5- Remove Dead state (optional)

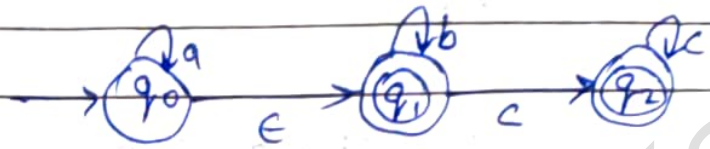
Q- Consider a FA with null move given below, make an equivalent FA without null move.

I)

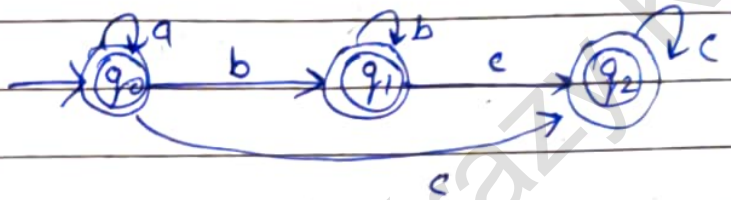


Sol:

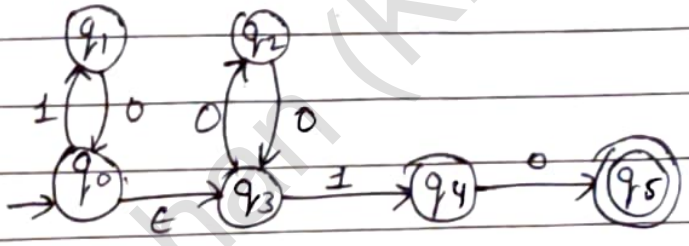
i)



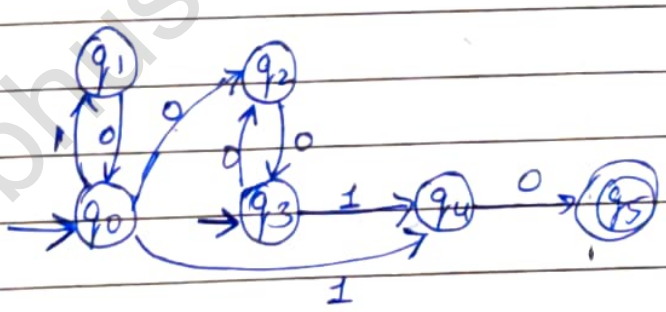
ii)



II)



Sol:



* ε-NFA to DFA Conversion

Method 1 - i) Convert ε-NFA to NFA
ii) Convert NFA to DFA

Method 2 - Start directly with initial state & go on adding states ----

Q - Convert below ε-NFA to DFA.

Sol ε - Closures -

ε-Closure (q_0) = $\{q_0, q_1, q_2\}$
ε-Closure (q_1) = $\{q_1, q_2\}$
ε-Closure (q_2) = $\{q_2\}$

Transition Table for NFA

States \ Σ	0	1
→ q_0	q_0	–
q_1	–	q_1
* q_2	q_2	–

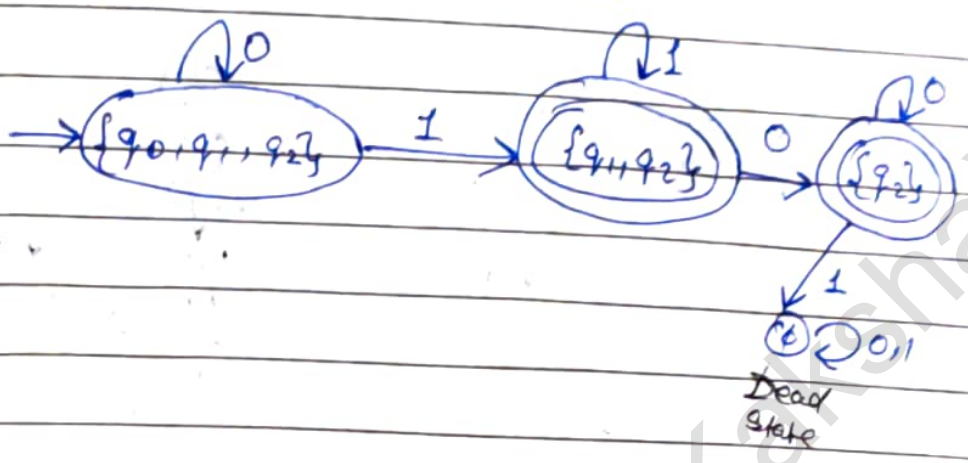
Transition Table for DFA →

q_0, q_2 → ε-Closure → $\{q_0, q_1, q_2\}$

Bcz initial state is q_0

ε-Closure \ Σ	0	1
→ $\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$
$\{q_1, q_2\}$ *	$\{q_2\}$	$\{q_1, q_2\}$
$\{q_2\}$ *	$\{q_2\}$	ϕ
ϕ	ϕ	ϕ

DFA Transition Diagram -



* Equivalence of DFA

→ Two FA M_1 & M_2 are said to be equivalent if they accept the same language.

$$L(M_1) = L(M_2)$$

i.e., both FA M_1 & M_2 accept exactly the same set of input strings.

→ Two FA are equivalent if -

i) The initial and final states of both the automata must be same.

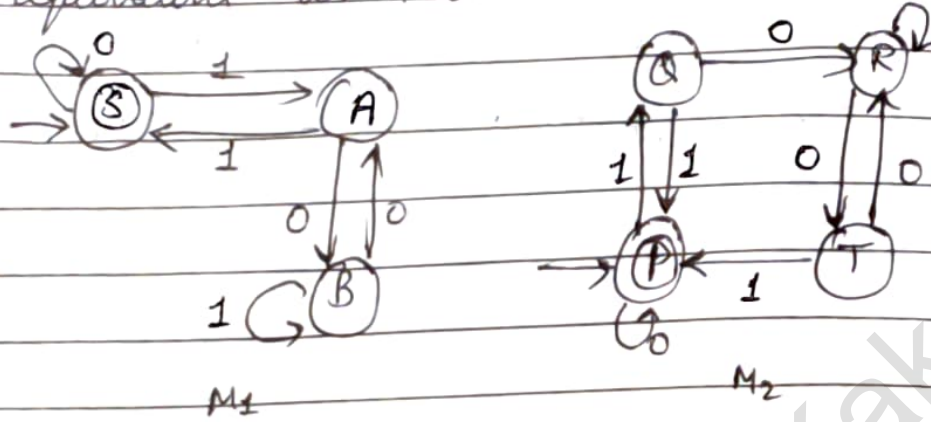
ii) Every pair of states chosen is from a different automaton only.

iii) The pair of states must be either both final states or intermediate states.

iv) If the pair has different types of states then it will be non-equivalent.

Q1- Check whether the two automata are equivalent or not.

i)



M_1 M_2

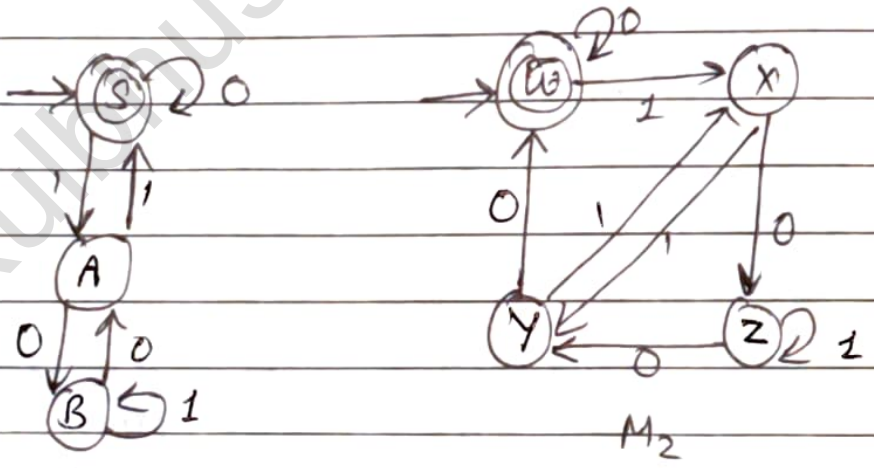
Sol^n i)

Pair \ Σ	0	1
(S, P)	(S, P)	(A, Q)
(A, Q)	(B, R)	(S, P)
(B, R)	(A, T)	(B, R)
(A, T)	(B, R)	(S, P)

Sol^n, $M_1 = M_2$ ⇒ Both FA are Equivalent

$\therefore L(M_1) = L(M_2)$

ii)



M_1 M_2

	0	1
(S, W)	(S, W)	(A, X)
(A, X)	(B, Z)	(S, Y)

Here, we stopped the procedure because the pair (S, Y) is incompatible b/z S is final state & Y is intermediate state

$\therefore M_1 \neq M_2$ & $L(M_1) \neq L(M_2)$

* Minimization of DFA

→ Any DFA defines a unique language, but converse is not true.

→ For a single language, we may have multiple DFAs.

→ For a storage efficiency, it is desirable to reduce the number of states as far as possible.

Note: i) Two states p, q of DFA are called Indistinguishable (Equivalent) if

$\hat{\delta}(p,w) \in F \Rightarrow \hat{\delta}(q,w) \in F$

and $\hat{\delta}(p,w) \notin F \Rightarrow \hat{\delta}(q,w) \notin F$

where $w \in \Sigma^*$

ii) If $\hat{\delta}(p,w) \in F$ and $\hat{\delta}(q,w) \notin F$ or vise-versa then the states p & q are said to be Distinguishable.

There are two methods to minimize the DFA
- Using Equivalence Class Partition Method
- Using Myhill Nerode Theorem (less errors in complex problem)

* Myhill Nerode Theorem to Minimize a DFA

i) Remove all inaccessible states.

ii) Consider all pair (p,q) states. If $p \in F$ & $q \notin F$ or vice-versa, mark the pair (p,q) as distinguishable.

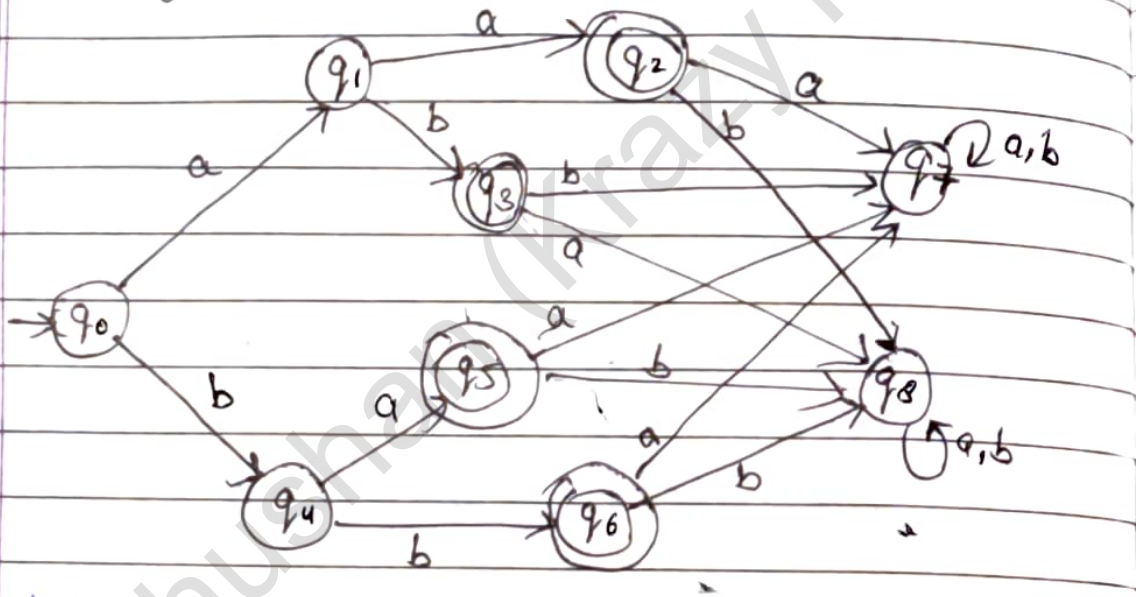
iii) Repeat the below steps untill no previous unmarked pair are marked.

a) For all pairs (p, q) & all $a \in \Sigma$, compute $\delta(p,a) = p_a$ & $\delta(q,a) = q_a$

b) If the pair (p_a, q_a) is marked as distinguishable then (p,q) will also distinguishable states, mark them also.

iv) Two or more states are equivalent if they are not marked.

Q- Minimize the DFA :-



* Method :

So Using Myhill Nerode Theorem :-

* Transition Table

State \ Σ	a	b
→ q_0	q_1	q_4
q_1	q_2	q_3
* q_2	q_7	q_8
* q_3	q_8	q_7
q_4	q_5	q_6
* q_5	q_7	q_8
* q_6	q_7	q_5
q_7	q_7	q_7
q_8	q_8	q_8

Step 0- Remove all unreachable states

Step 1- Non Final States = $\{q_0, q_1, q_4, q_7, q_8\}$
Final States = $\{q_2, q_3, q_5, q_6\}$

→ So, Cartesian product b/w Non-final state and final states will form the pairs of distinguishable pairs.

→ Marking these distinguishable pairs with 'X' sign.

	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8
q_0	✓								
q_1	X	X							
q_2	X	X							
q_3	✓		X	X					
q_4	X	X			X				
q_5	X	X			X				
q_6	O	✓	X	X	✓	X	X		
q_7	O	✓	X	X	✓	X	X		
q_8									

~~Iteration-1~~

Step 2- If there is any unmarked pair (q, p) such that $\delta(q, x)$ & $\delta(p, x)$ is marked then also mark (p, q) also for input x, ~~Refer~~ marking with '✓' in above table

For

i) $\delta(q_0, a) \to q_1$ } Marked (✓)
$\delta(q_1, a) \to q_2$ }

ii) $\delta(q_0, a) \to q_4$ } Marked (✓)
$\delta(q_4, a) \to q_5$ }

iii) $\delta(q_0, A) \rightarrow q_1$ } Not marked
$\delta(q_7, a) \rightarrow q_7$
Checking for input b
$\delta(q_0, b) \rightarrow q_4$ } Not marked
$\delta(q_7, b) \rightarrow q_7$

iv) $\delta(q_0, a) \rightarrow q_1$ } Not marked
$\delta(q_8, a) \rightarrow q_8$
Checking for input b
$\delta(q_0, b) \rightarrow q_4$ } Not Marked
$\delta(q_8, b) \rightarrow q_8$

v) $\delta(q_1, a) \rightarrow q_2$ } Not marked
$\delta(q_4, a) \rightarrow q_5$
Checking for input b
$\delta(q_1, b) \rightarrow q_3$ } Not Marked
$\delta(q_4, b) \rightarrow q_6$

vi) $\delta(q_1, a) \rightarrow q_2$ } Marked (V)
$\delta(q_7, a) \rightarrow q_7$

vii) $\delta(q_2, a) \rightarrow q_7$ } Not marked
$\delta(q_3, a) \rightarrow q_8$
Checking for input b
$\delta(q_2, b) \rightarrow q_8$ } Not marked
$\delta(q_3, b) \rightarrow q_7$

viii) $\delta(q_2, a) \rightarrow q_7$ } Indistinguishable Pair
$\delta(q_5, a) \rightarrow q_7$

So on...

Step 3- Iteration - 2

→ Again repeating step 2 and check for remaining unmarked pairs.

→ Marking them with '0'

i) $\delta(q_0, a) \rightarrow q_1$ } Marked (0)
$\delta(q_7, a) \rightarrow q_7$ }

ii) $\delta(q_0, a) \rightarrow q_1$ } Marked (0)
$\delta(q_8, a) \rightarrow q_8$ }

⋮

So on - - -

Step 4- Iteration - 3

→ Again repeating & checking for remaining unmarked pairs.

i) $\delta(q_1, a) \rightarrow q_2$ } Not marked.
$\delta(q_4, a) \rightarrow q_5$ }

ii) $\delta(q_2, a) \rightarrow q_7$ } Not marked
$\delta(q_3, a) \rightarrow q_8$ }

⋮

So on - - -

Nothing is changed and no new pair is marked.

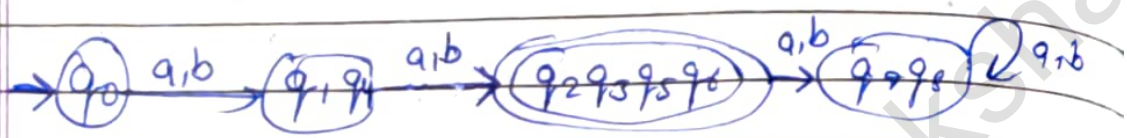
So we can stop the process here.

Step 5- Combine all Equivalent unmarked pairs & make them single state

$q_1 \approx q_4$

$q_2 \approx q_3 \approx q_5 \approx q_6$

$q_7 \approx q_8$



* Method 2

Using Equivalence Class Partition Method

Step2 Transition Table

States \ Σ	a	b
→ q_0	q_1	q_4
q_1	q_2	q_3
* q_2	q_2	q_8
* q_3	q_8	q_7
q_4	q_5	q_6
* q_5	q_7	q_8
* q_6	q_7	q_8
q_7	q_7	q_7
q_8	q_8	q_8

Step- Remove all unreachable States,

Step3- Π_0 or P_0 = {Set of Non-final State, Set of Final State}

$P_0 = \{\{q_0, q_1, q_4, q_7, q_8\}, \{q_2, q_3, q_5, q_6\}\}$

Step 4- To check whether sets of P_0 can be partitioned or not. (calculate P_1)

i) For Set $\{q_0, q_1, q_4, q_7, q_8\} = G1$

→ $\delta(q_0, a) \to q_1$ } Distinguishable
$\delta(q_1, a) \to q_2$ }
$\{q_0\}, \{q_1\}$

→ $\delta(q_0, a) \to q_1$ } Distinguishable
$\delta(q_4, a) \to q_5$ }
$\{q_0\}, \{q_1, q_4\}$

→ $\delta(q_0, a) \to q_1$ } Indistinguishable
$\delta(q_7, a) \to q_7$ }
$\{q_0, q_7\}, \{q_1, q_4\}$

→ $\delta(q_0, a) \to q_1$ } Indistinguishable
$\delta(q_8, a) \to q_8$ }
$\{\{q_0, q_7, q_8\}, \{q_1, q_4\}\}$
$G11$ $G12$

ii) For Set $\{q_2, q_3, q_5, q_6\} = G2$

→ $\delta(q_2, a) \to q_7$ } Indistinguishable
$\delta(q_3, a) \to q_8$ }
$\{q_2, q_3\}$

→ $\delta(q_2, a) \to q_7$ } Indistinguishable
$\delta(q_5, a) \to q_7$ }
$\{q_2, q_3, q_5\}$

→ $\delta(q_2, a) \to q_7$ } Indistinguishable
$\delta(q_6, a) \to q_8$ }
$\{\{q_2, q_3, q_5, q_6\} = G2$

So, $P_1 = \{\{q_0, q_7, q_8\}, \{q_1, q_4\}, \{q_2, q_3, q_5, q_6\}\}$
$G11$ $G12$ $G2$

Step5- To calculate P_2, check Group G_{11}, G_{12}, G_2 can be partitioned or not.

i) for $G_{11} = \{q_0, q_7, q_8\}$

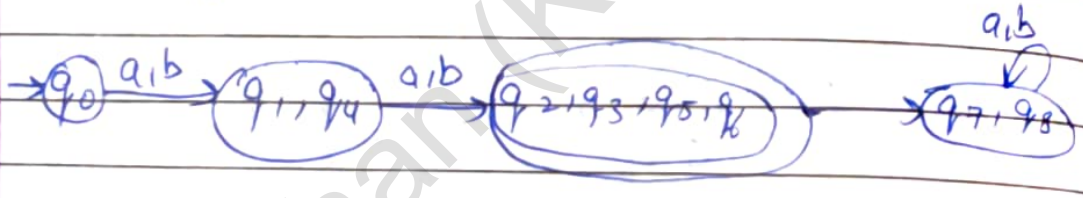
ii) for $G_{12} = \{q_1, q_4\}$

iii) For $G_2 = \{q_2, q_3, q_5, q_6\}$

$P_2 = \{\{q_0\}, \{q_7, q_8\}, \{q_1, q_4\}, \{q_2, q_3, q_5, q_6\}\}$

Step6- $P_3 = \{\{q_0\}, \{q_7, q_8\}, \{q_1, q_4\}, \{q_2, q_3, q_5, q_6\}\}$

Since $P_2 = P_3$, this is the final partition.



* Finite Automata with Output -

→ In FA, on reading the input string if we get final state then string is accepted else rejected.

→ If we need to produce some precise information (output) & not only "accept" or "reject" the we use finite set of output symbols & machine function.

→ There are two FA with Output -
  - Moore Machine
  - Mealy Machine

i) Moore Machine -

→ The output depends only upon present state.

→ Moore m/c is a 6 tuple

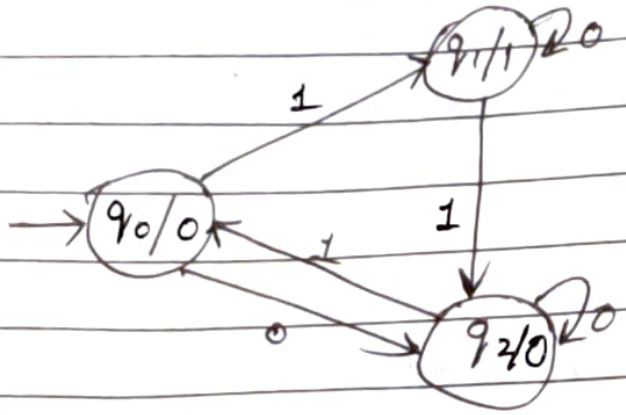
$$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$$

Where,

Q: Finite set of states
Σ: Finite set of input alphabets
Δ: Output alphabet
δ: Transition function, $\Sigma \times Q \rightarrow Q$
λ: Output funtion mapping, $Q \rightarrow \Delta$
q_0: Initial state of m/c

Note: If 'n' is length of input string then 'n+1' will be length of output.

Ex-



Present State	Next State		Output
	0	1	
→q_0	q_2	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_0	0

ii) Mealy Machine -

→ The output depends on present state and input.

→ Mealy m/c is a 6 tuple.

$$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$$

Q: Finite set of states

Σ: Finite set of i/p symbols.

Δ: Output alphabet

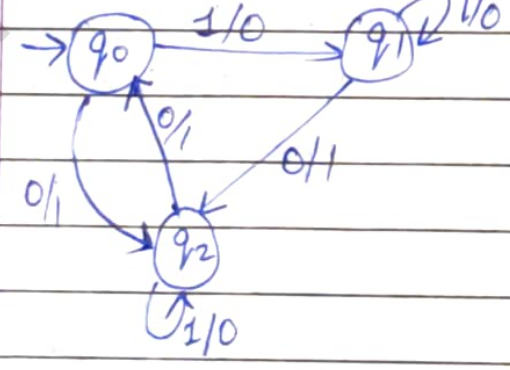
δ: Transition functions, $Q \times \Sigma \to Q$

λ: Machine (Output) funct^n, $\lambda: Q \times \Sigma \to \Delta$

q_0: Initial state of machine

Note: length of output is equal to length of input string.

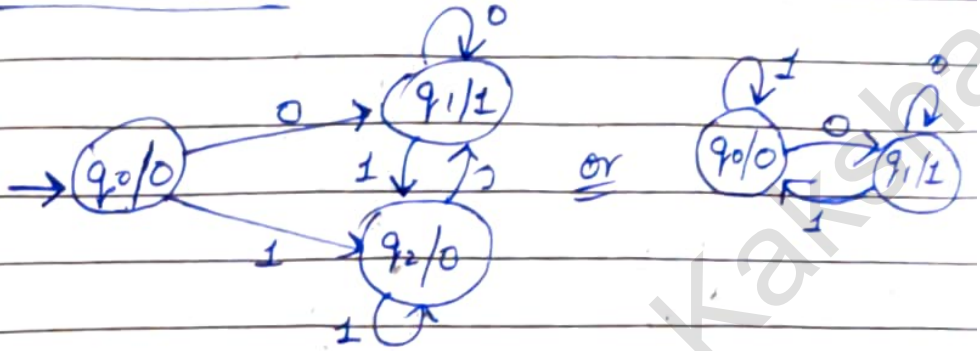
Ex-



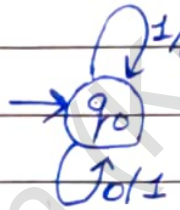
Present State	0		1	
	State	o/p	State	o/p
→q_0	q_2	1	q_1	0
q_1	q_2	1	q_1	0
q_2	q_0	1	q_2	0

Q1- Design a Moore Machine & Mealy Machine to generate 1's complement of given Binary number.

Sol- Moore Machine –

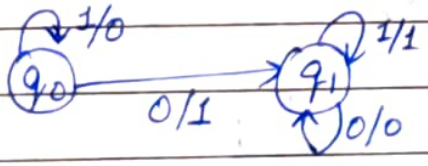


Mealy Machine –



Q2- Design a Mealy machine which will increment the given Binary number by 1.

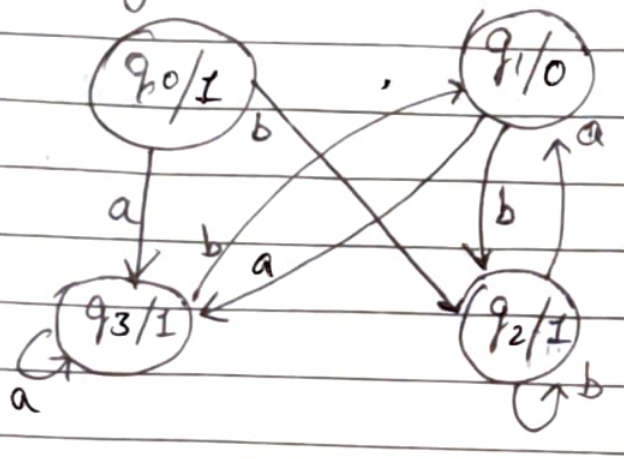
Sol- 110011 ← Read from this side bit by bit
110100



100 101
101 110

* Equivalence of Moore & Mealy Machine

Q1- Convert the following Moore machine into mealy machine -



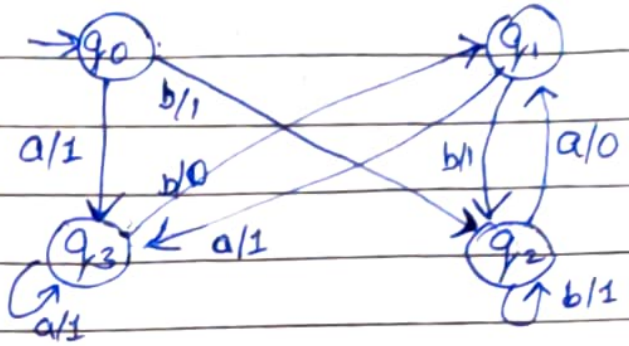
* Moore m/c transition Table

Present State	Next State a	Next State b	Output
→ q_0	q_3	q_2	1
q_1	q_3	q_2	0
q_2	q_1	q_2	1
q_3	q_3	q_1	1

* Mealy m/c transition table

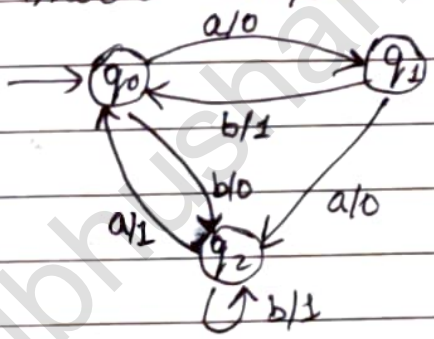
Present State	a State	a o/p	b State	b o/p
→ q_0	q_3	1	q_2	1
q_1	q_3	1	q_2	1
q_2	q_1	1	q_2	1
q_3	q_3	1	q_1	0

* ~~Moore~~ Mealey m/c



Note: 'm' states & 'n' output in moore m/c then there will be 'm' state maximum in mealey m/c.

Q2- Convert the following Mealey m/c into moore m/c-



Sol = * Mealey m/c table

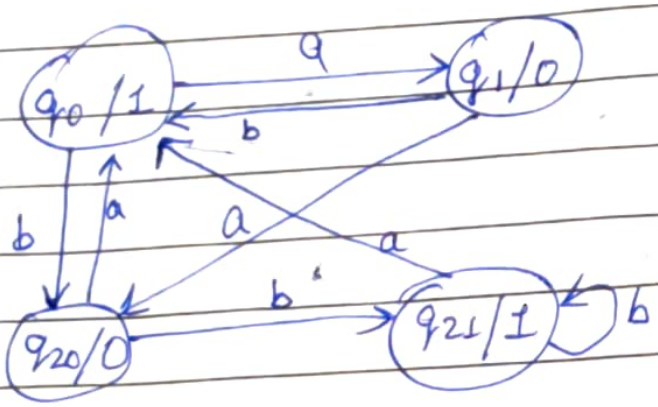
Present State	a		b	
	State	o/p	State	o/p
→q_0	q_1	0	q_2	0
q_1	q_2	0	q_0	1
q_2	q_0	1	q_2	1

* Moore Table

Present State	Next State a	Next State b	O/p
→q_0	q_1	q_{20}	1
q_1	q_{20}	q_0	0
q_{20}	q_0	q_{21}	0
q_{21}	q_0	q_{21}	1

first fill this w/ o/p

* Moore M/C



Note: If there are 'm' states & 'n' output in mealy machine then there will be maximum possible 'm x n' states in moore machine.

Unit - 2

Regular Expressions & Languages

* Regular Expression (RE)

→ The language accepted by FA can be easily described by simple expressions called regular Expressions.

→ It is the most effective way to represent any language.

→ Set of string accepted by FA known as Regular Lang

→ Set of Operators used in RE –

- $+$ → Union Operator
- $\bullet$ → Concatenat
- $*$ → Closure Operator

Note: ϕ → An Empty set of Regular Expression.
ϵ, ^ → Null or Empty string.

→ We can define RE of Σ as follows –

1- Any terminal symbol (Σ), ϵ & ϕ are RE

e.g., $RE = \epsilon$ $L = \{\epsilon\}$

$RE = \phi$ $L = \phi$

$RE = a$ $L = \{a\}$

2- Union of two RE R_1 & R_2 written as $R_1 + R_2$ is also RE

e.g., $RE = a+b$ $L = \{a, b\}$

3- Concatenat of two RE R_1 & R_2 written as $R_1 \cdot R_2$ is RE

e.g., $RE = a \cdot b$ $L = \{ab\}$

4. The iterat^{n} (Closure) of RE R written as R^* is RE
e.g., $RE = a^*$ $L = \{\epsilon, a, aa, aaa, \ldots\}$

5. If RE R then (R) is also Regular Expression.

6. The RE over Σ are those obtained recursively by the Applicat^{n} of rule 1-5 once or several times.
e.g., $RE = (ab+ba)a^*$

* Some Algebraic Laws for RE

Let L, M, N Be RE

1- Associativity & commutativity -

$L+M = M+L$ } comm.

$(L+M)+N = L+(M+N)$ } Associativity

$(LM)N = L(MN)$

2- Identities & Annihilators

$\phi + L = L + \phi = L$ } Id.

$\epsilon L = L\epsilon = L$

$\phi L = L\phi = \phi$ } Anni.

3- Distributive Laws

$L(M+N) = LM + LN$

$(M+N)L = ML + NL$

4- Idempotent law

$L+L = L$

5- Laws Involving Closure (*)

$(L^*)^* = L^*$

$\phi^* = \epsilon$

$\epsilon^* = \epsilon$

$L^+ = LL^*$

$L^* = L^+ + \epsilon$

$LL^* = L^*L$

$L^*L^* = L^*$

$\epsilon + RR^* = R^* = \epsilon + R^*R$

$(LM)^*L = L(ML)^*$

$(L+M)^* = (L^*M^*)^* = (L^* + M^*)^*$

Q1- Write the regular Expression for the Langugs where -

i) String starts w/ 0 & Ends w/ 1 over $\Sigma = \{0,1\}$

Sol: $RE = 1(0+1)^*0$

ii) string begins or ends w/ 00 & 11

Sol: $RE = [(00+11)(0+1)^*] + [(0+1)^*(00+11)]$

iii) $L = \{a^n b^m \mid n \geq 4,\ m \leq 3\}$

Sol: $a^4a^*(\epsilon + b + b^2 + b^3)$

iv) $L = \{ab^n w \mid n \geq 3,\ w \in (a,b)^+\}$

Sol: $ab^3b^*(a+b)^+$

v) String do not ends w/ double letter over $\Sigma = \{a,b\}$

Sol: $(a+b)^*(ab+ba)$

vi) String containing exactly two 'b' over $\Sigma = \{a, b\}$

Sol: $RE = a^* b a^* b a^*$

vii) $L = \{abba\}$

Sol: $RE = abba$

viii) $L = \{\Lambda, 01\}$

Sol: $RE = \Lambda + 01$

ix) $L = \{\epsilon, 11, 1111, 11111, \dots\}$

Sol: $RE = (11)^*$

x) all strings contains atleast two 0's.

Sol: $(1+0)^* 0 1^* 0 (1+0)^*$ or $1^* 0 1^* 0 ((1+0)^*$

xi) all strings do not end w/ 01.

Sol: $(0+1)^* (00 + 11 + 10)$

xii) all strings has odd no. of 0's.

Sol: $(1^* 01^* 01^*)^* 01^*$

Q2- Prove $\epsilon + 1^*(011)^*[1^*(011)^*]^* = (1+011)^*$

So LHS = $\epsilon + 1^*(011)^*[1^*(011)^*]^*$

$= [1^*(011)^*]^*$ [Using $\epsilon + RR^* = R^*$]

$= (1+011)^* = RHS$ [Using $(L+M)^* = (L^*M^*)^*$]

Q3- Prove $(1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1) = 0^*1(0+10^*1)^*$

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

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

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

$= \phi(\epsilon + 00^*)1(0+10^*1)^*$

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

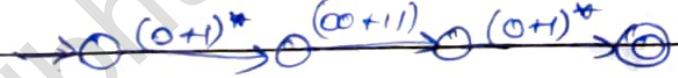
* **Klen's Theorem** ⇒ (RE to FA)

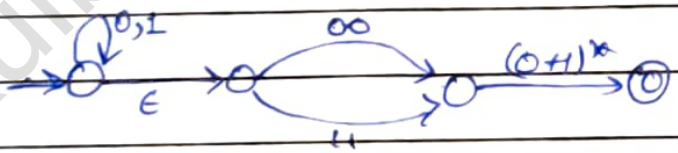
Klen's theorem states that -

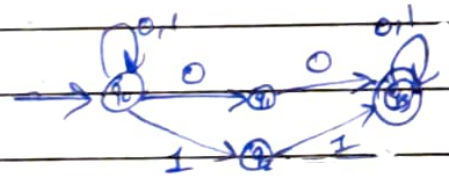
i) Any regular language can be accepted by FA.

ii) The language accepted by any FA is regular.

Q4- Construct FA Equivalent to RE $(0+1)^*(00+11)(0+1)^*$



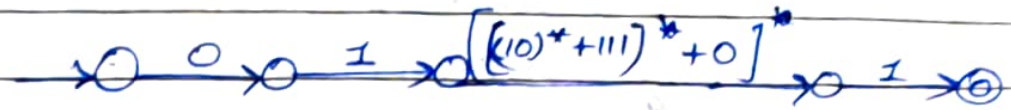


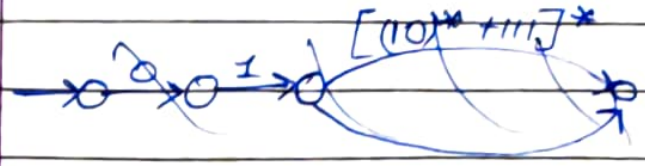


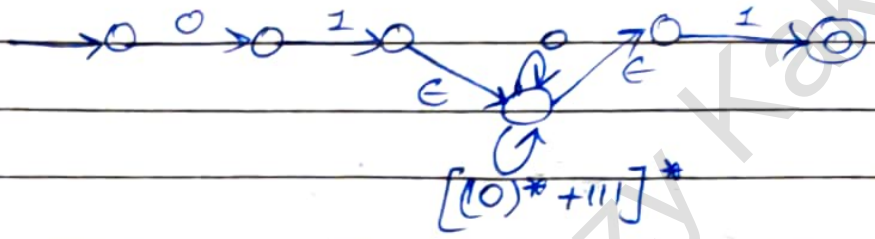
State	0	1
→ q_0	q_0q_1	q_0q_2
q_1	q_3	–
q_2	–	q_3
q_3	q_3	q_3

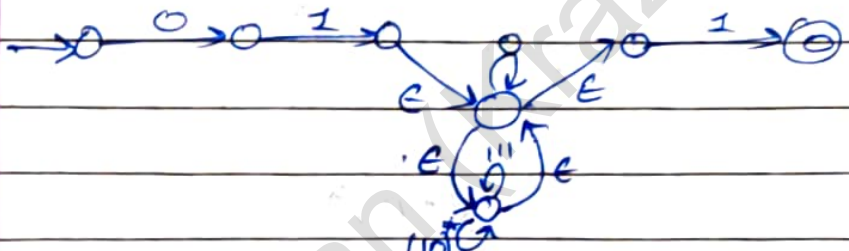
Q5- Construct FA from RE $01[(10)^* + 111)^* + 0]^* 1$

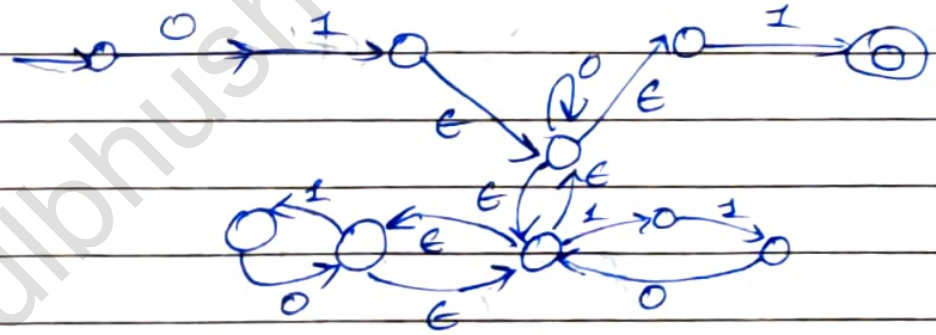
Sol =







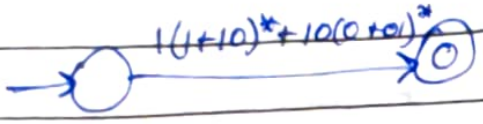




Q6 - Draw FA recognizing the following language.

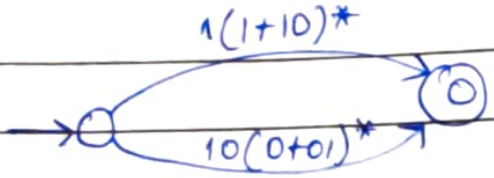
$1(1+10)^* + 10(0+01)^*$

So =

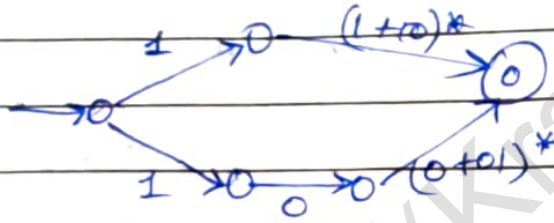


Step 1-

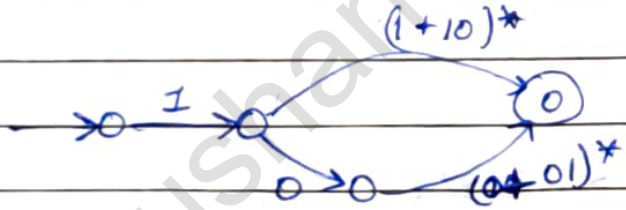
Step 2-



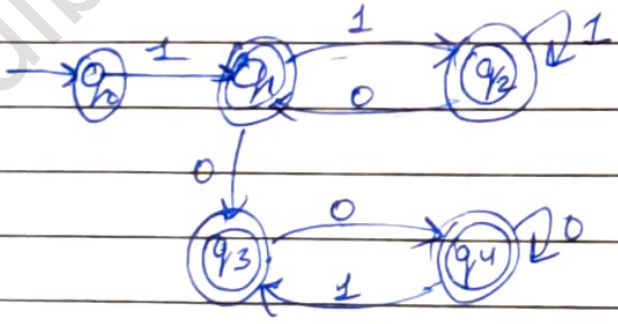
3-



4-



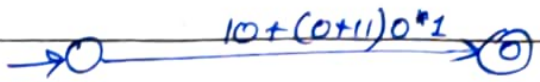
5-



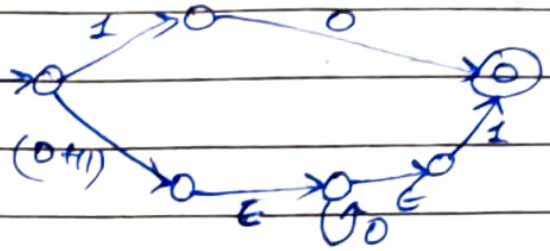
Q7 - Construct FA by reduced state equivalent to $10 + (0+11)0^*1$

Sol:

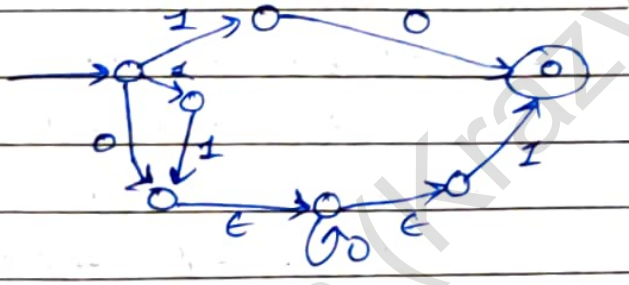
Step 1 -



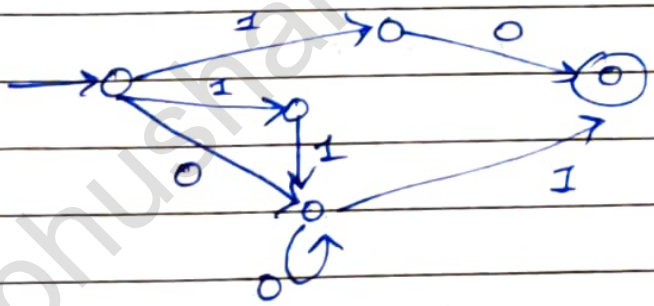
Step 2 -



Step 3 -



Step 4 -



* **Arden's Theorem** ⇒ (FA to RE)

Let P & Q be two RE over Σ. If P does not contains $\wedge$, then the following equation in R

$$R = Q + RP \quad \text{——— (i)}$$

has unique solution given by

$$R = QP^* \quad \text{——— (ii)}$$

Proof Using Eq. (i), $R = Q + RP$

$$R = Q + (Q + RP)P$$
$$= Q + QP + RP^2$$
$$= Q + QP + (Q + RP)P^2$$
$$= Q + QP + QP^2 + RP^3$$

If we substitute the value of R recursively

$$R = Q + QP + QP^2 + QP^3 + \dots$$
$$R = Q(\wedge + P + P^2 + P^3 + \dots)$$
$$R = QP^*$$

For correct

$$R = Q + RP$$
$$= Q + (QP^*)P$$
$$= Q(\wedge + P^*P)$$
$$= QP^*$$

* Algebraic method using Arden's theorem

→ This algebraic method is used to find RE recognized by transition system.

→ The following assumptn are made –

i) Transitn graph does not have null move.

ii) It has only one initial state (V_1).

iii) Vertices are $V_1, V_2 \dots V_n$.

iv) Let α_{ji} represents the transitn from V_j to V_i

v) Calculate V_i such that

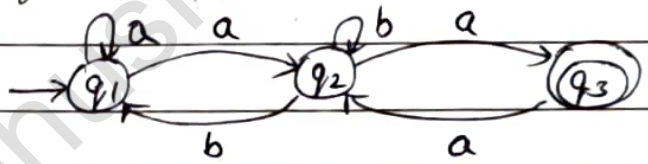
$$V_1 = V_1\alpha_{11} + V_2\alpha_{21} + \dots V_n\alpha_{n1} + \wedge$$

$$V_2 = V_1\alpha_{12} + V_2\alpha_{22} + \dots V_n\alpha_{n2}$$

$$\vdots$$

$$V_n = V_1\alpha_{1n} + V_2\alpha_{2n} + \dots V_n\alpha_{nn}$$

Q8- Consider the transitn system given below & find RE for that



$$q_1 = q_1a + q_2b + \wedge$$

$$q_2 = q_1a + q_2b + q_3a$$

$$q_3 = q_2a$$

Substituting value of q_3 in q_2

$$q_2 = q_1a + q_2b + q_2aa$$

$$q_2 = q_1a + q_2(b + aa)$$

$$\downarrow R \quad \downarrow Q \quad \downarrow R \quad \downarrow P$$

If $R = Q + RP$ then $R = QP^*$

$$q_2 = q_1a(b + aa)^*$$

Substituting this q_2 in q_1

$q_1 = q_1a + q_1a(b+aa)^*b + \Lambda$

$q_1 = \Lambda + q_1[a + a(b+aa)^*b]$

$q_1 = \Lambda \cdot [a+a(b+aa)^*b]^*$

$q_1 = [a+a(b+aa)^*b]^*$

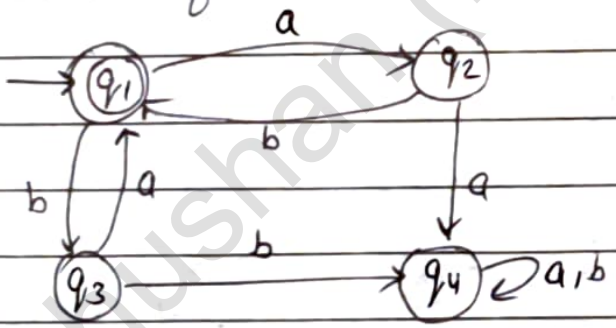
Substituting this q_1 to q_2

$q_2 = [a+a(b+aa)^*b]^* a(b+aa)^*$

Substituting this q_2 to q_3

$\boxed{q_3 = [a+a(b+aa)^*b]^* a(b+aa)^* a}$

Q9- Write RE from FA



Sol = Let us see the equations

$q_1 = q_2b + q_3a + \epsilon$

$q_2 = q_1a$

$q_3 = q_1b$

$q_4 = q_2a + q_3b + q_4a + q_4b$

$= q_2a + q_3b + q_4(a+b)$

Substituting the values of q_2 & q_3 in q_1

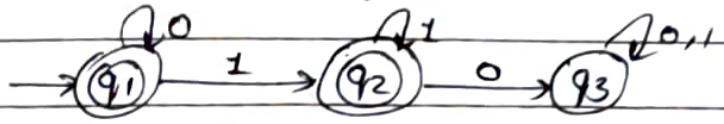
$q_1 = q_1ab + q_1ba + \epsilon$

$q_1 = \epsilon + q_1(ab + ba)$

$\boxed{q_1 = (ab+ba)^*}$

$[R = Q + RP]$
$[R = QP^*]$

Q10- Construct RE for given DFA



So= Let us build the RE for each state

$q_1 = q_1 0 + \epsilon$

$q_2 = q_1 1 + q_2 1$

$q_3 = q_2 0 + q_3 (0+1)$

Since final states are q_1 & q_2, so we are interested in solving q_1 & q_2 only.

$q_1 = q_1 0 + \epsilon$

$q_1 = \epsilon + q_1 0$

$q_1 = \epsilon \cdot 0^*$

$\boxed{q_1 = 0^*}$

Now, substituting this q_1 value into q_2

$q_2 = 0^* 1 + q_2 1$

$q_2 = 0^* 1 \cdot (1)^*$

So, $RE = q_1 + q_2$

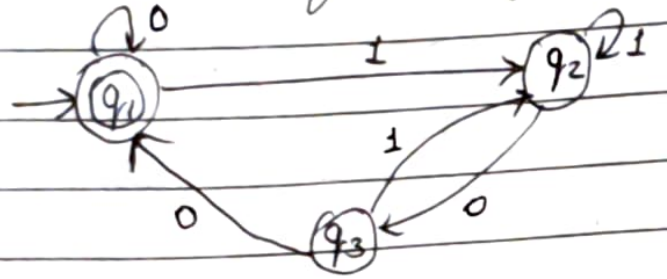
$\boxed{RE = 0^* + 0^* 1 \cdot 1^*}$

or $\boxed{RE = 0^* + 0^* 1^+}$

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

or $\boxed{RE = 0^* 1^*}$

Q11- Construct RE for the given transition diagram



Sol:-

$q_1 = q_1 0 + q_3 0 + \Lambda$

$q_2 = q_1 1 + q_2 1 + q_3 1$

$q_3 = q_2 0$

$q_2 = q_1 1 + q_2 1 + q_2 01$

$= q_1 1 + q_2 (1 + 01)$

$q_2 = q_1 1 (1+01)^*$

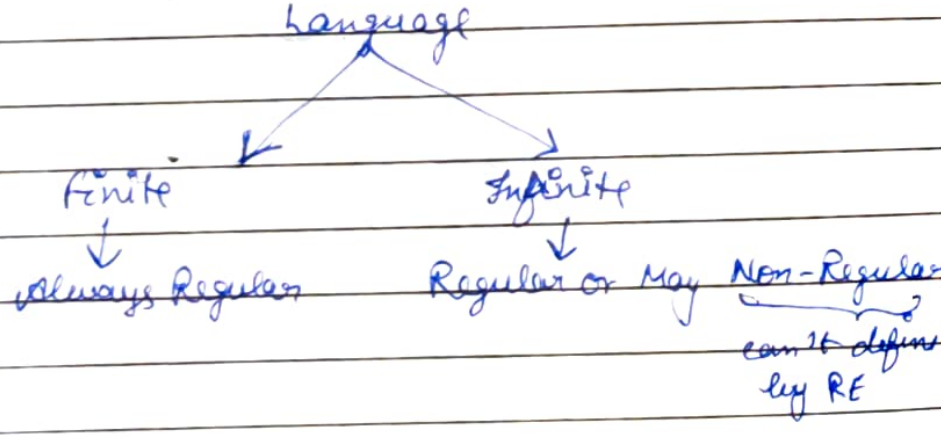
$q_1 = q_1 0 + q_1 1(1+01)^* 00 + \Lambda$

$= \Lambda + q_1 (0 + 1(1+01)^* 00)$

$q_1 = [0 + 1(1+01)^* 00]^*$

* Pumping Lemma for Regular Language

→ Pumping lemma is used to prove that a language is not regular.



Pumping lemma states that, If L is a regular language then the L has pumping length 'n' i.e., no. of states in FA, such that any string 'w' where $|w| \geq n$ may be divided in 3 parts $w = xyz$ such that the following condition must be true.

i) $xy^iz \in L$ for every $i \geq 0$

ii) $|y| > 0$

iii) $|xy| \leq n$

Q12- Show that $L = \{a^p / P \text{ is prime}\}$ is not regular

Sol- Suppose L is regular & let no. of states (Pumping length) be 7.

Step 1-

Step 2- $w = \underbrace{aa}_{x}\underbrace{aaa}_{y}\underbrace{aa}_{z}$ $|w| \geq 7$

$w = xyz$ $|xy| \leq 7$

where, $x = aa$ $y = aaa$ $z = aa$

Step 3- $w = xy^iz$

$w = aa\,aaa\,aa \in L$ for $i = 1$

$w = aa\,aaaaaa\,aa \notin L$ for $i = 2$

Because $|w| = 10 \neq$ Prime

Step 4- Hence, $xy^1z \in L$ is Regular But

$xy^2z \notin L$ is not regular

So, our assumption was wrong, therefore given language is not regular.

Note: When comparision is there ⇒ Non-Reg.
Bz FA does not have memory to store previous value of i

Q13- Show that $L = \{0^i 1^i \mid i \geq 1\}$ is not regular.

So = Step-1- Suppose L is regular and let no. of states be 5

Step2- $w = \underbrace{000}_{x}\underbrace{00}_{y}\underbrace{111}_{z}$ $|w| > 5$

$w = xyz$ $|xy| \leq 5$

where, $x = 0$, $y = 00$, $z = 111$

Step3- $w = xy^i z$

$w = 000\underbrace{00}_{y^2}111$ for $i = 2$

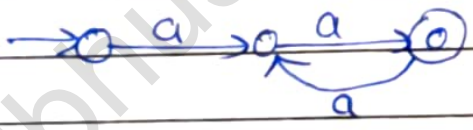
So, $w \notin L$

Step4- Our assumption was wrong, so the given language is not regular.

Q14 Show that $L = \{a^{2n} \mid n \geq 1\}$ is regular.

So = $L = \{aa, aaaa, aaaaaa, \ldots\}$

To show, Language is regular we have to make FA to accept this language.



* Closure properties of non-regular language -

1- Union of two regular language are regular.
2- Intersection of two RL are regular.
3- Complement of RL is regular.
4- Difference of RL is regular.
5- Concatenation of RL are regular.
6- Reverse of RL is Regular.
7- Closure of RL is regular.
8- Homomorphism of RL is regular.
9- Inverse Homomorphism of RL is regular.

* **Pigeonhole Principle -**

It states that if 'n' pigeon fly into m pigeonhole & $n > m$ then atleast one pigeonhole must contain two or more pigeons.

e.g.; Show that in a group of 50 students atleast 5 are born in same month.

There are $m = 12$ months (pigeonhole)

and $K_m + 1 = 50$

$K_{12} = 50 - 1$

$K = \frac{49}{12}$

$K = 4\frac{1}{12}$

Remainder 1, atleast 4+1 students are born in same month.

→ In automata theory, pigeons are the strings of a's, the pigeonhole are the states & correspondence associated each string w/ the state to which it goes when the string is i/p.

→ This principle is used to prove that certain infinite language are not regular.

* Decidability - Decision Properties of RL

To decide
- ↳ Emptiness problem (FA accepts empty or not)
- ↳ Finiteness
- ↳ Equivalence problem
- ↳ Membership

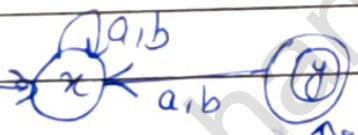
i) Emptiness Problem

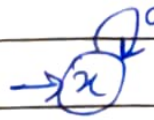
FA accepts empty lang. or not.

Algo: 1- Eliminate inaccessible states.

2- If resultant FA, has atleast one FA, it means it accepts non-empty language otherwise empty language.

e.g.,





- Since no final state
- Empty language

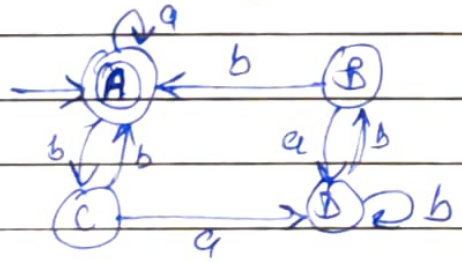
ii) Finiteness Problem

Checking whether language accepted by FA.

Algo: 1- Eliminate all inaccessible state.

2- Eliminate states from which final states are not reachable.

3- In resultant FA, there exists loop, then FA accepts infinite language.



iii) Equivalence problem -
Checking whether given two FA accept the same language, or not.

iv) Membership problem -
Checking whether the string s accepted by given DFA or not.

Unit - 3

REGULAR & NON-REGULAR GRAMMARS

Date
Page

* Every FA M accepts a language L which is represented By L(M).
. There are two ways for representing a language.
  - Using Finite Automata (FA)
  - Using Regular Expression (RE)

* If a language is non-regular, it can't be represented either using a above two ways. i.e., FA & RE.

* **Context Free Grammar -**

→ A set of product^n rule that describe all possible strings in a given formal lang.

→ CFG can be described as a combinat^n of four tuple -

$$G(V, T, P, S)$$

Where,

V → Set of Variable or Non-terminal (A, B, - - - - Y, Z)

T → Set of terminals (a, b, c, - - - - -, *, etc)

S → Starting Symbol

P → Set of product^ns.

Note: Regular Language ⊂ Context Free Language

* Context Free Language -

If $G(V, T, P, S)$ is a CFG, the language of G, denoted by $L(G)$ is the set of terminal strings that have derivation from the start symbol. i.e.,

$$L(G) = \{w / w \in T^* \text{ and } S \xrightarrow[G]{*} w\}$$

* The derivation are represented into two forms - (↳ to check whether the string is accepted or not by CFG)
  - ↳ Sentential form
  - ↳ Parse Tree or Derivation Tree form

* Types of Derivation -

i) Left Most Derivation -

Always apply production rule to left most variable (Non-terminal). in every step.

ii) Right Most Derivation -

Always apply production rule to right most variable.

Note: Ambiguity occours when more than one derivation tree occurs, which we will discuss later.

Q1- Find CFG for the following languages -

i) $L = \{a^n b^n : n \geq 0\}$

Sol= $G(V, T, P, S)$

where,

$V = \{S\}$

$T = \{a, b\}$

$S = \{S\}$

P: $S \rightarrow aSb$

$S \rightarrow \epsilon$

ii) $L = \{a^n b^n : n > 0\}$ or $L = \{a^n b^n : n \geq 1\}$

Sol= $S \rightarrow aSb \mid ab$

iii) $L = \{a^n b^{n+2} : n \geq 0\}$

Sol= $S \rightarrow aSb \mid bb$

iv) $L = \{a^{2n} b^n : n \geq 0\}$ → Put value of n to get terminating condition

Sol= $S \rightarrow aaSb \mid \epsilon$

v) $L = \{a^{2n} b^n : n \geq 1\}$

Sol= $S \rightarrow aaSb \mid aab$

vi) $L = \{a^{2n+3} b^n : n \geq 0\}$

Sol= $S \rightarrow aaSb \mid aaa$

vii) $L = \{a^m b^n : m > n \ \& \ n \geq 0\}$

Sol $S \rightarrow AB$

$A \rightarrow aA \mid a$

$B \rightarrow aBb \mid \epsilon$

viii) $L = \{ w \mid n_a(w) = n_b(w) \}$

Sol= $S \rightarrow aSb \mid bSa \mid SS \mid \epsilon$

ix) $L = \{ a^m b^m c^n : m, n \geq 0 \}$

Sol= $S \rightarrow XY$

$X \rightarrow aXb \mid \epsilon$

$Y \rightarrow cY \mid \epsilon$

x) $L = \{ w \in \{a,b\}^* \mid w \text{ is a palindrome of odd length} \}$

Sol= $S \rightarrow aSa \mid bSb \mid a \mid b$

xi) $L = \{ w \in \{a,b\}^* \mid w \text{ is a palindrome of even length} \}$

Sol= $S \rightarrow aSa \mid bSb \mid \epsilon$

xii) $L = \{ w \in \{a,b\}^* \mid w \text{ is a palindrome of even length w/ |w| > 2} \}$

Sol= $S \rightarrow aSa \mid bSb \mid aa \mid bb$

xiii) $L = \{ w \in \{a,b\}^* \mid w \text{ is a palindrome of either odd or even length} \}$ or $L = \{ ww^R \cup w(a+b)w^R \}$

Sol= $S \rightarrow aSa \mid bSb \mid a \mid b \mid \epsilon$

xiv) $L = \{ wcw^R : w \in (a,b)^* \}$

Sol $S \rightarrow aSa \mid bSb \mid c$

xv) $L = \{ a^i b^j : i \leq j \leq 2i, \; i, j \geq 1 \}$

Sol= for $i = 1$, $j = 1, 2$

$w = ab, abb$

for $i = 2$, $j = 2, 3, 4$

$w = aabb, aabbb, aabbbb$

$S \rightarrow aSb \mid aSbb \mid abb$

xvi) $L = \{ a^i b^j c^k : i+j=k, \ i,j \geq 1 \}$

Sol:- For $i=1, \ j=1, \ k=2$

$w = abcc$

P: $S \rightarrow aSc \mid aXc$

$X \rightarrow bXc \mid bc$

xvii) $L = \{ a^i b^j c^k : j=i+k, \ i,j \geq 1 \}$

Sol:-

P: $S \rightarrow XY$

$X \rightarrow aXb \mid ab$

$Y \rightarrow bYc \mid bc \mid e$

Q2- Find CFL associated with CFG given below-

$S \rightarrow aB \mid bA$

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

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

Sol=(1) $S \rightarrow aB$

$\rightarrow ab$

(2) $S \rightarrow bA$

$\rightarrow ba$

(3) $S \rightarrow aB$

$\rightarrow abS$

$\rightarrow abaB$ or $abbA$

$\rightarrow abab$ or $abba$

(4) $S \rightarrow aB$

$\rightarrow aaBB$

$\rightarrow aabS$

$\rightarrow aaBbA$

$\rightarrow aabbba$

(5) $S \rightarrow bA$

$\rightarrow baS$

$\rightarrow babA$

$\rightarrow babbAA$

$\rightarrow babbaa$

$L = \{x : x$ containing equal no. of a's & b's$\}$

Q3- Find CFG for each RE.

i) (ab^*)

Sol =
$S \rightarrow aB$
$B \rightarrow bB \mid \epsilon$

ii) $(baa + abb)^*$

Sol =
$S \rightarrow AS \mid BS \mid \epsilon$
$A \rightarrow baa$
$B \rightarrow abb$

iii) a^*b^*

Sol =
$S \rightarrow AB$
$A \rightarrow aA \mid \epsilon$
$B \rightarrow bB \mid \epsilon$

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

Sol =
$S \rightarrow 0A01A1$
$A \rightarrow 0A \mid 1A \mid \epsilon$

Q4- Consider the grammar of the production. Compute the string aababbb with the left most & right most derivation. Draw parse tree.

$S \rightarrow aSS$

$S \rightarrow b$

So = Left most derivation :—

$S \rightarrow aSS$ ($S \rightarrow aSS$)

$\rightarrow aaSSS$ ($S \rightarrow b$)

$\rightarrow aabSS$ ($S \rightarrow aSS$)

$\rightarrow aababSS$ ($S \rightarrow b$)

$\rightarrow aababbS$ ($S \rightarrow b$)

$\rightarrow aababbb$ ($S \rightarrow b$)

Right Most derivation :

$S \rightarrow aSS$

$\rightarrow aSb$ ($S \rightarrow b$)

$\rightarrow aaSSb$ ($S \rightarrow aSS$)

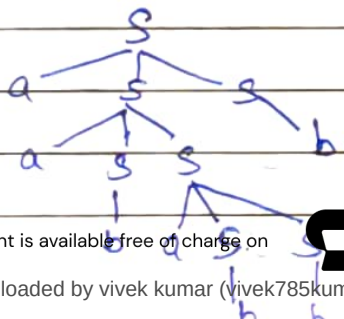
$\rightarrow aaSaSSb$ ($S \rightarrow aSS$)

$\rightarrow aaSaSbb$ ($S \rightarrow b$)

$\rightarrow aaSabbb$ ($S \rightarrow b$)

$\rightarrow aababbb$ ($S \rightarrow b$)

Derivation Tree (parse tree) -



Q.5- Find parse tree for the string abbcde considering the productions.

$S \rightarrow aAcBe$

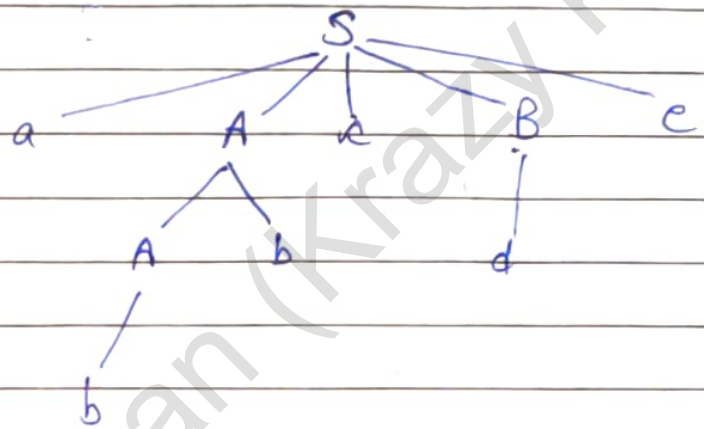
$A \rightarrow Ab$

$A \rightarrow b$

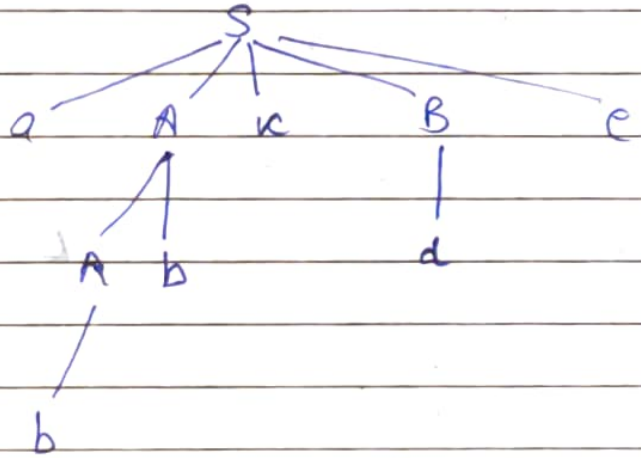
$B \rightarrow d$

Is this ambiguous? Justify

So = Left Most derivation Tree



Right Most derivation Tree



As, both the derivation tree are equal, So, the given grammer is not ambiguous

* Ambiguous Grammar -

→ A grammar is said to be ambiguous if the language generated by the grammar contains some string that has two or more different parse tree.

→ There is no general rule for removing ambiguity from CFG. It involves rewriting of grammar so that there is only one derivation tree for every string belonging to L(G) i.e., language generated by grammar G.

→ Inherently ambiguity -

- A CFL L(G) is said to be inherently ambiguous if all its grammar is ambiguous.
- Or we can say that it is a ambiguous CFG from which elimination of ambiguity is not possible. It is a property of language not grammar.

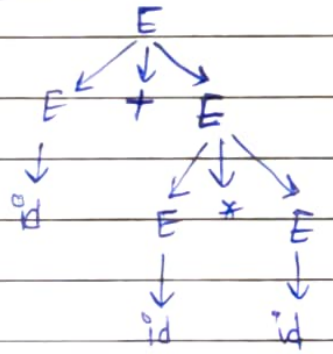
Q6- Remove ambiguity for given Grammar, for given string id+id*id

$E \rightarrow E + E$

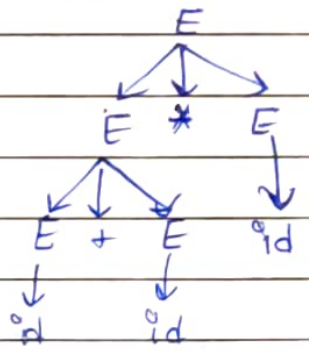
$E \rightarrow E * E$

$E \rightarrow id$

Sol = LMD Tree



RMD Tree



Removal of ambiguity -

Here,

Precedence of * is high, & Left associativity → Left Recursion

Precedence of + is lower, & LA → LR

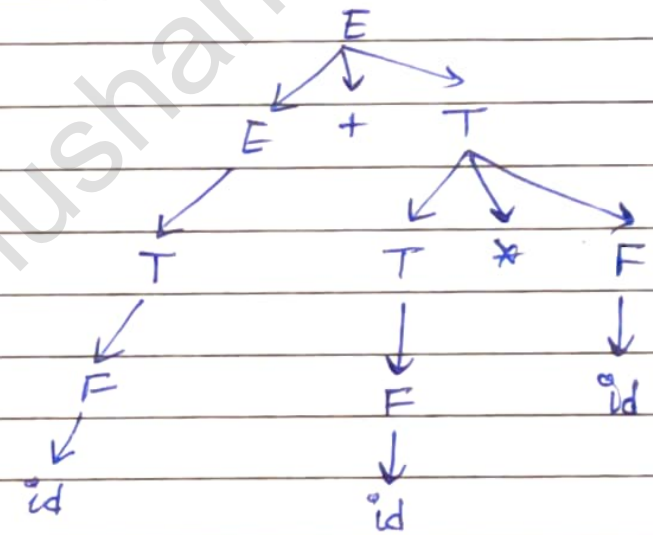
So, higher precedence should be at lowest level. i.e., * in this grammar.

$E \rightarrow E + T \mid T$ (LR)

$T \rightarrow T * F \mid F$ (LR)

$F \rightarrow id$

LMD Tree -



Q7- $E \to E+E \mid E*E \mid E \uparrow E \mid id$ for id * id * id ↑ id

So = ③ + LA → LR

② * LA → LR

① ↑ RA → RR

So, $E \to E + T \mid T$ (LR)

$T \to T * F \mid F$ (LR)

$F \to G \uparrow F \mid G$ (RR)

$G \to id$

Q8- $E \to E+E \mid a$ for $w = a+a+a$

So = $E \to E+a \mid a$

Date
Page

* Simplificatⁿ of CFG

A CFG can be simplified by eliminating
- → Useless (Unreachable) symbols
- → Null (ϵ) productⁿ
- → Unit productⁿ

i) Elimination of Useless symbols -

Algo -

1- Identify all the variables that can termina
2- Find all the variables in productⁿ which can have path from start symbol (S)

Q9- Eliminate Useless symbol for CFG

$A \rightarrow xyz \mid Xyzz$
$X \rightarrow Xz \mid xYz$
$Y \rightarrow yYy \mid Xz$
$Z \rightarrow Zy \mid z$

Sol- (1) Since, $A \rightarrow xyz$
$Z \rightarrow z \mid Zy$
can terminate, so they are useful.
→ But X & Y do not lead to a string of terminals, therefore eliminate these productⁿs.

(2) Now, grammar is

$\boxed{A \rightarrow xyz}$

Because Z is not reachable from starting symbol A.

ii) Elimination of ϵ- Productions

Algo-

1- Find variable which lead to ϵ directly or indirectly & put them in V_N.

2- Replace value of each & every variable belongs to V_N by putting it as ϵ.
Add new productions rule in given grammar.

Q10- Find CFG without ϵ- productions equivalent to

$S \rightarrow ABaC$

$A \rightarrow BC$

$B \rightarrow b \mid \epsilon$

$C \rightarrow D \mid \epsilon$

$D \rightarrow d$

Sol- 1- Here, $V_N = \{A, B, C\}$

2- $S \rightarrow ABaC \mid BaC \mid AaC \mid ABa \mid aC \mid Aa \mid Ba \mid a$

$A \rightarrow BC \mid C \mid B$

$B \rightarrow b$

$C \rightarrow D$

$D \rightarrow d$

iii) Elimination of Unit Productions

Algo-

1- Find productions rule which are in form

$A \rightarrow B$

2- Add new productions using following

If $B \rightarrow y_1 \mid y_2 \mid y_3 - - -$

Then we add

$A \rightarrow y_1 \mid y_2 \mid y_3 - - -$

Q11- Remove Unit production for the given CFG.

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow C \mid b$

$C \rightarrow D$

$D \rightarrow E$

$E \rightarrow a$

Sol ⇒ ① $B \rightarrow C$ $C \rightarrow D$ $D \rightarrow E$

also, $B \rightarrow C \rightarrow D \rightarrow E$

② $S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow a \mid b$

$C \rightarrow a$

$D \rightarrow a$

$E \rightarrow a$

* Normal Forms.

→ In a CFG, the RHS of a production can be any string of terminals and non-terminals.

→ When we apply certain restrictions on the production then CFG is said to be in "Normal Form".

→ There are two normal forms.

- Chomsky Normal Form
- Greibanch Normal Form

1) Chomsky Normal Form

→ Grammar should be free from
- → Useless symbols
- → ϵ - Productions
- → Unit productions

→ Convert the grammar into CNF such that it has only two types of productions

$A \to a$ $(a \in T \ \& \ A \in V)$

$A \to BC$ $(A, B, C \in V)$

Q12- Convert the given grammar into CNF.

$S \to ABa \quad A \to aab \quad B \to Ac$

Sol: Step 1 - Consider the production $S \to ABa$

Let consider new productions

$X_a \to a \quad \& \quad D_1 \to AB$

So, production will be

$S \to D_1 X_a \quad X_a \to a \quad D_1 \to AB$

Step 2- Consider production $A \to aab$ & it is not CNF

So, let us consider new productions

$X_b \to b$

So, production will be

$A \to X_a X_a X_b$

still it is not CNF, so let us consider new productions $D_2 \to X_a X_b$

So, production will be

$A \to X_a D_2 \quad D_2 \to X_a X_b \quad X_b \to b$

Step 3- Consider the production $B \to Ac$, not CNF
Let us consider new production $X_c \to c$.
So production will be

$$B \to AX_c \qquad X_c \to c$$

Therefore, the productions for given grammar in CNF will be -

$$S \to D_1 X_a$$
$$X_a \to a$$
$$D_1 \to AB$$
$$A \to X_a D_2$$
$$D_2 \to X_a X_b$$
$$X_b \to b$$
$$B \to AX_c$$
$$X_c \to c$$

2) Greibach Normal Form (GNF)

$$NT \to T \cdot (NT - - - NT)$$
$$N \to T$$

Algo.
1- Eliminate null production, unit production & useless symbols.
2- Every production must be in form

$$A \to a\alpha$$

(a: Terminal, α: string of non-terminal)

3- Rename variable as $A_1, A_2, \dots A_N$.
4- If production $A_i \to A_j\alpha$ with $i > j$, then must generate production substituting for A_j.
5- Remove left Recursion, $A_k \to A_k\alpha$ by creating new production.

If $A \to A\alpha / \beta$ is in left recursion

then $A \to \beta X / \beta$

$X \to \alpha X / \alpha$

Q13- Convert the following grammar into GNF.

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

So step1- Eliminate null & unit productn & useless productn.

Convert into CNF.

Step2- Rename,

$S = A_1$ & $A = A_2$

So, $A_1 \to A_2A_2/a$

$A_2 \to A_1A_1/b$

Step3- A_1 productn :

$A_1 \to a$ (GNF)

$A_1 \to A_2A_2$ (Not GNF)

A_2 productn :

$A_2 \to b$ (GNF)

$A_2 \to A_1A_1$ (Not GNF)

So, Here, $A_i > j$

So, $A_2 \to aA_1 / A_2A_2A_1$ (GNF, Not GNF)

Step4- Eliminate left recursion for

$A_2 \to A_2A_2A_1 / aA_1 / b$

(α = A_2A_1, β_1 = aA_1, β_2 = b)

Adding new productn Z, & A_2 productn

$A_2 \to aA_1 / b$ ---

$A_2 \to aA_1Z / bZ$

(β_1 = aA_1, β_2 = b)

Z productn -

$Z \to A_2A_1Z / A_2A_1$

(α = A_2A_1, α = A_2A_1)

step 5- A_2 production:

$$A_2 \rightarrow aA_1 | b | aA_1Z | bZ |$$

A_1 production:

$$A_1 \rightarrow aA_1A_2 | bA_2 | aA_1ZA_2 | bZA_2$$

Z production:

$$Z \rightarrow aA_1A_1Z | bA_1Z | aA_1ZA_1Z | bZA_1Z$$

$$Z \rightarrow aA_1A | aA_1B | bA_1 | bZA_1$$

Regular Grammar

The language accepted by finite automata can described using a set of production known as regular grammar. The production of a regular grammar are of the following form -

$$A \rightarrow a \quad A \rightarrow aB \quad A \rightarrow Ba \quad A \rightarrow \epsilon$$

Where, $a \in T$ & $A, B \in V$

A language generated by Regular Grammar is known as regular language.

A regular grammar could be written in two forms.

i) Right linear form $(A \rightarrow aB)$

ii) Left linear form $(A \rightarrow Ba)$

* DFA to Right Linear Regular Grammar -

Every DFA can be described using set of productn with these steps -

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

Corresponding RL, $G = (V, T, P, S)$

1- Rename $q_0 \in Q$ as $S \in V$.

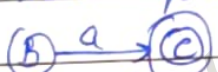
2- Rename states $\in Q$ as $A, B, C, D, \dots \in V$

3- Create productn rule as

a) If q_0 is final state then add productn $S \to \epsilon$ to P

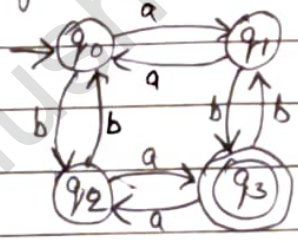
b) If 

then make productn $B \to aC$

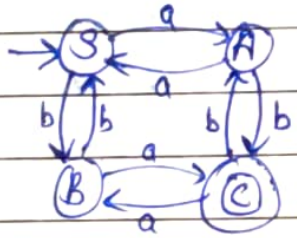
c) If 

then make productn $B \to aC/a$

Q14- Give Right Linear Grammar for DFA



So Step1 Rename States,



Step 2- Set of productions are given by

P: $S \to aA / bB$

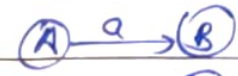
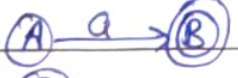
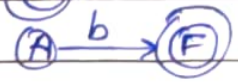
$A \to aS / bC / b$

$B \to bS / aC / a$

$C \to aB / bA$

* Right Linear Grammer to DFA -

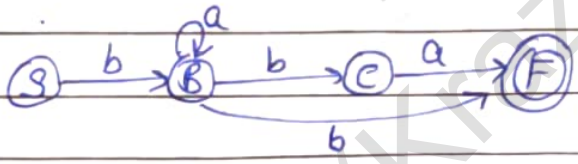
Every right linear grammar can be represented using DFA

- $A \to aB \Rightarrow$ 
- $A \to aB/a \Rightarrow$ 
- $A \to \epsilon \Rightarrow$ 
- $A \to b \Rightarrow$  where F is new state

Q15- Convert the following RLG to FA.

$S \to bB,\ B \to bC\ \ B \to aB\ \ C \to a\ \ B \to b$

Sol=

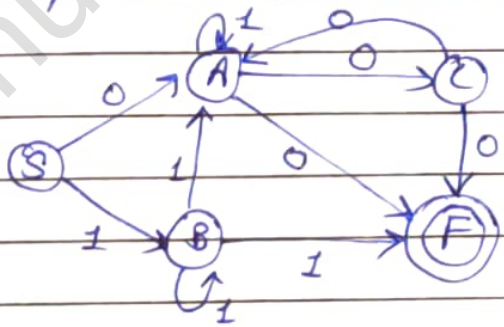


Q16- Convert RG to DFA

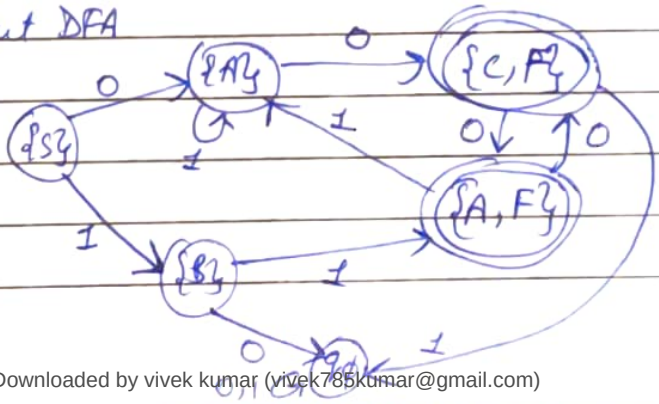
$S \to 0A/1B\ \ A \to 0C/1A/0\ \ B \to 1B/1A/1\ \ C \to 0/0A$

Sol= Let introduce new state F as a final state for productions like $A \to 0,\ B \to 1,\ C \to 0$

Step1-



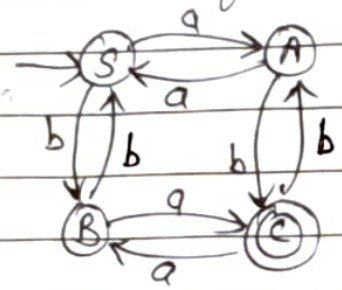
Step2- Equivalut DFA



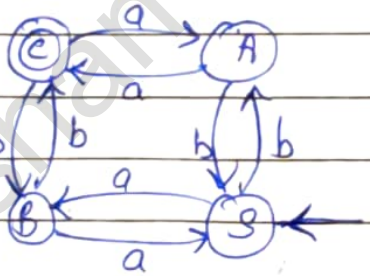
* DFA to Left Linear Grammar

1- Interchange starting state & final state.
2- Reverse the direction of all transitions.
3- Write the grammar from transition graph in left linear form.

Q17- Give RLG for DFA



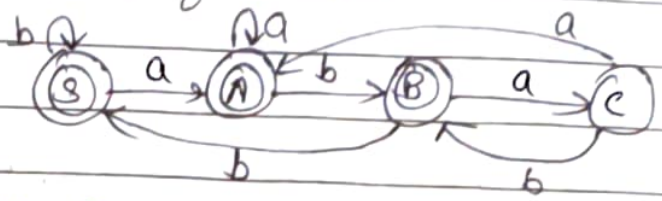
So Step1 Interchange start state & final state and reverse the direction of transition.



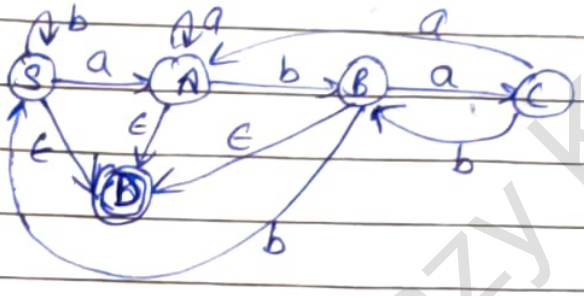
Step2- Write Eq. LLG.

$S \rightarrow Ba/Ab$
$A \rightarrow Sb/Ca/a$
$B \rightarrow Sa/Cb/b$
$C \rightarrow Bb/Aa$

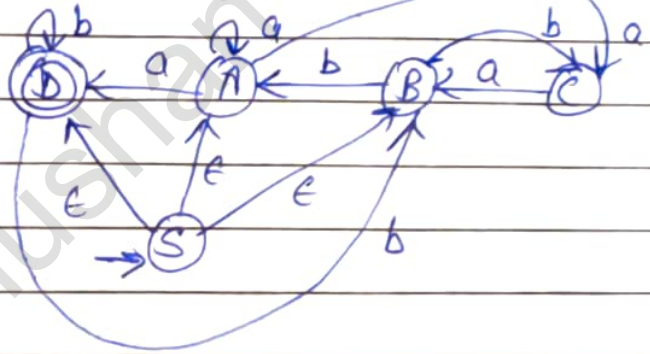
Q18- Give LLG for DFA



Sol- Step1- It has 3 final states, make them a single final state.



Step 2- Interchange start & final state and reverse the direction of transition



Step 3- Equivalent LLG

$S \rightarrow D/A/B$

$A \rightarrow Aa/Da/Ca/a$

$B \rightarrow Cb/Ab$

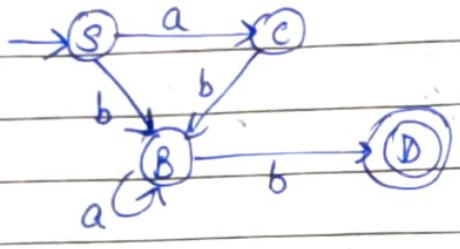
$C \rightarrow Ba$

$D \rightarrow Bb/Db/b$

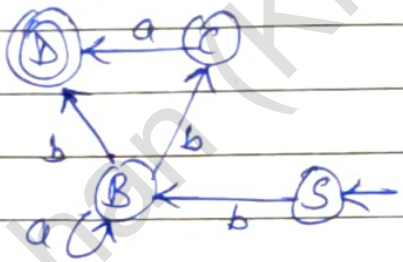
* LLG to DFA -

Q19- Construct DFA for RL by LLG
$S \to Ca/Bb \quad C \to Bb \quad B \to Ba/b$

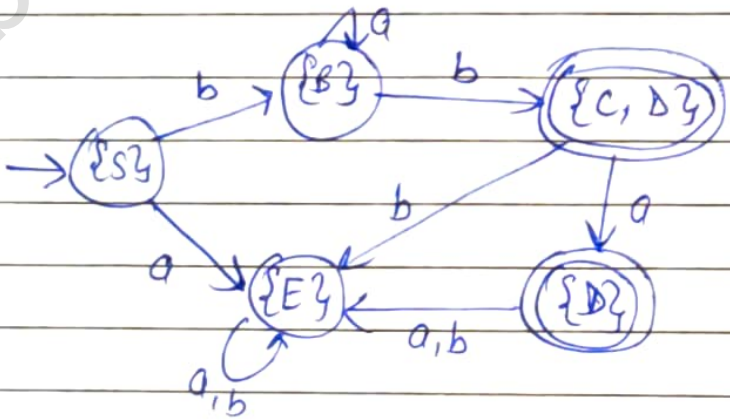
So Step 1 - Draw transition graph.



Step 2 Reverse the direction of transition & interchange start & final state



Step 3- Convert FA to DFA



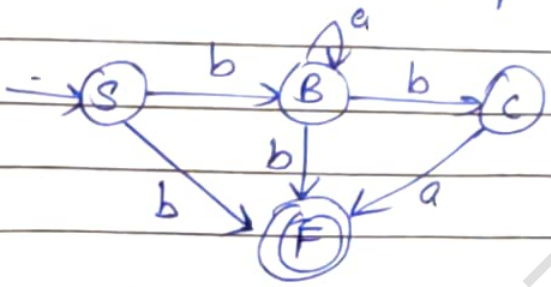
* RLG to LLG

RLG → Transit^n Graph → LLG

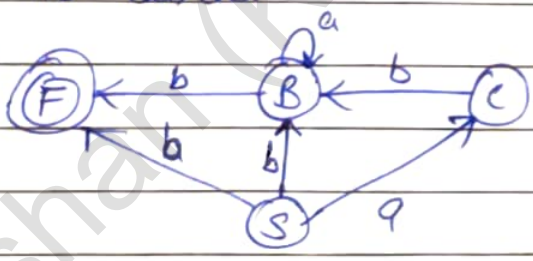
Q20- Convert RLG to LLG

S → bB/b B → bC B → aB C → a B → b

Step 1- RLG to Transit^n Graph



Step 2- Interchang Final state w/ Start state & reverse the direct^n



Step-3- Write LLG

S → b/Bb/Ca

B → Ba/b

C → Bb

* LLG to RLG

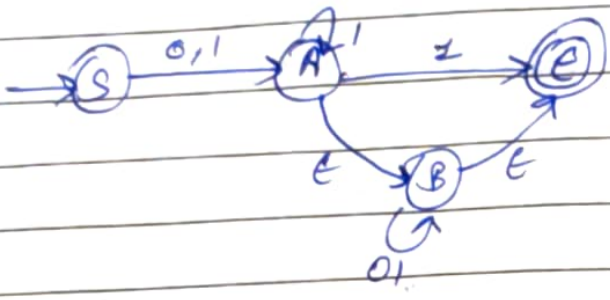
LLG → Transit^n Graph → RRL

Same procedure as RLG to LLG.

Q2- Construct RLG for RE -

$R = (0+1)1^* (1+(01)^*)$

Sol=



$S \rightarrow 0A / 1A$

$A \rightarrow 1A / 1 / B$

$B \rightarrow 01B / 01$

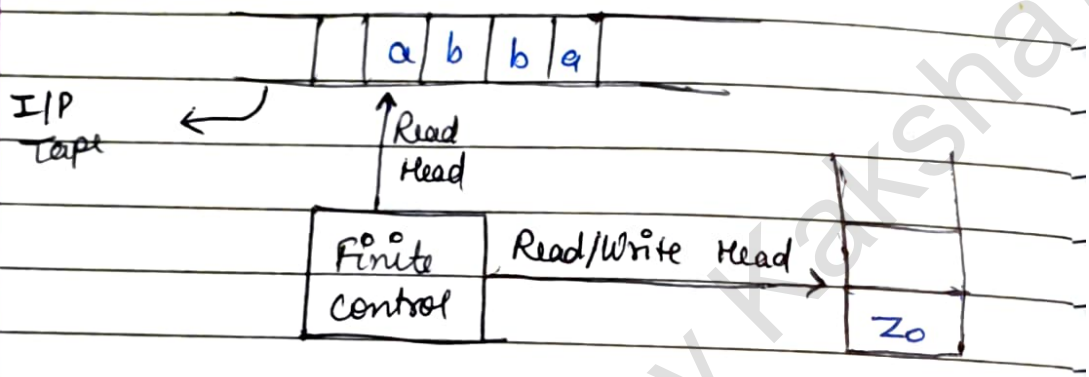
UNIT - 4

PDA & Properties of CFL

Date
Page

* Push Down Automata (PDA)-

→ Push down automata is a finite automata with memory called stack which helps PDA to recognize context free language (CFL).



→ A pushdown automata M is defined as 7-tuples.

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

Where,

Q: Finite non-empty set of states.
Σ: Finite non-empty set of input symbols.
Γ: Finite non-empty set of pushdown symbol
q_0: Initial state. ($q_0 \in Q$)
Z_0: special stack symbol called initial stack symbol
F: Set of final state ($F \subseteq Q$)
δ: Transition function

$$Q \times (\Sigma \cup \epsilon) \times \Gamma \longrightarrow Q \times \Gamma^*$$

$$\delta(q_0, a, b) \vdash (q_2, b)$$

- q_0: Current State
- a: Input Symbol
- b: Top Symbol of the stack
- q_2: Next State
- b: top symbol of the stack

* Operations –

i) Pop operation ⇒ $\delta(q_1, a, b) \vdash (q_2, \epsilon)$

ii) Push operation ⇒ $\delta(q_1, a, b) \vdash (q_2, ab)$

iii) NOp (NO operation / Skip) ⇒ $\delta(q_1, a, b) \vdash (q_2, b)$

* DPDA & NPDA

→ Deterministic Pushdown Automata is one for which every input string has unique path through initial state in the m/c.

→ Non-Deterministic PDA is one for which at a certain time it has to select a particular path among possible paths.

→ NDFA also has 7-tuples as DPDA has but transition function δ is defined as

$$Q \times \{\Sigma \cup \epsilon\} \times \Gamma \vdash 2^{Q} \times \Gamma^{*}$$

* Language of PDA

A language L can be accepted by PDA in two ways –

i) Through final state

ii) Through Empty stack.

Q1- Construct a PDA for accepting the language $L = \{a^n b^n \mid n \geq 1\}$. Give set rule & transit diagram.

Sol = (1) Using set of Equations -

$\delta(q_0, a, z_0) \vdash (q_0, az_0)$

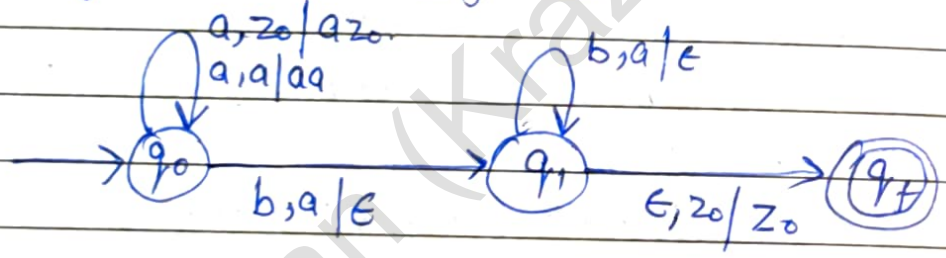
$\delta(q_0, a, a) \vdash (q_0, aa)$

$\delta(q_0, b, a) \vdash (q_1, \epsilon)$

$\delta(q_1, b, a) \vdash (q_1, \epsilon)$

$\delta(q_1, \epsilon, z_0) \vdash (q_f, z_0)$ ⇒ Through final state

(2) Using Transition Diagram



Q2- Design a PDA for $L = \{a^n b^n \mid n \geq 1\}$ through Empty stack.

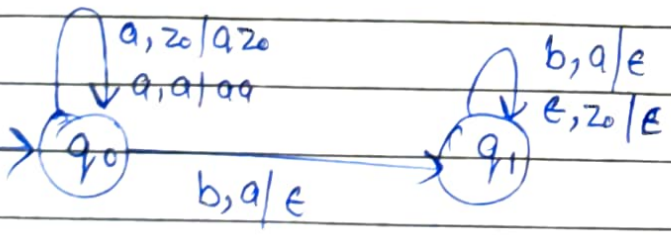
$\delta(q_0, a, z_0) \vdash (q_0, az_0)$

$\delta(q_0, a, a) \vdash (q_0, aa)$

$\delta(q_0, b, a) \vdash (q_1, \epsilon)$

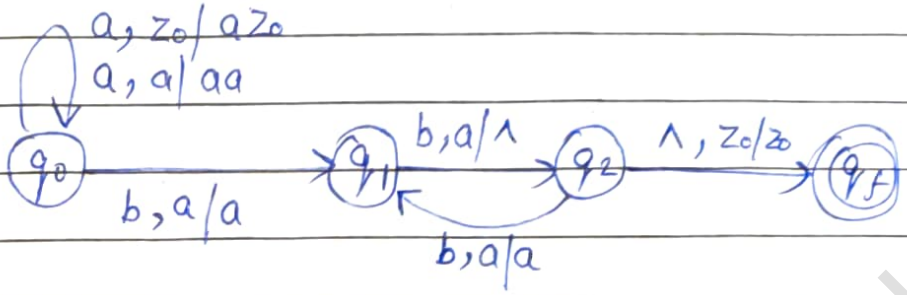
$\delta(q_1, b, a) \vdash (q_1, \epsilon)$

$\delta(q_1, \epsilon, z_0) \vdash (q_1, \epsilon)$



Q3- Design a PDA for $L = \{a^n b^{2n} / n \geq 1\}$

Sol =

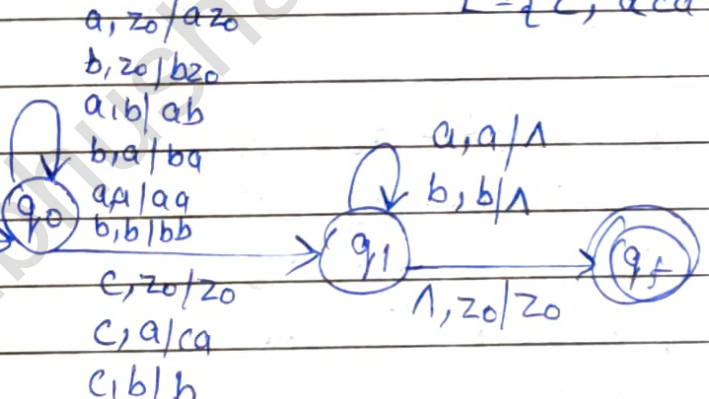


$\delta(q_0, a, z_0) \vdash (q_0, az_0)$
$\delta(q_0, a, a) \vdash (q_0, aa)$
$\delta(q_0, b, a) \vdash (q_1, a)$
$\delta(q_1, b, a) \vdash (q_2, \Lambda)$
$\delta(q_2, b, a) \vdash (q, a)$
$\delta(q_2, \Lambda, z_0) \vdash (q_f, z_0)$

Q4- Design a PDA for accepting a language
$L = \{wcw^r / w \in \{a,b\}^*\}$

Sol =

$L = \{c, aca, bcb, abcba, \ldots\}$

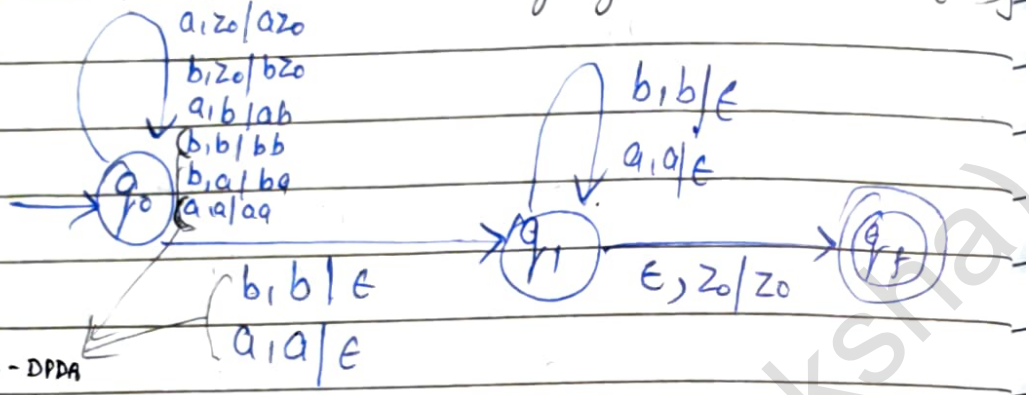


$\delta(q_0, a, z_0) \vdash (q_0, az_0)$
$\delta(q_0, b, z_0) \vdash (q_0, bz_0)$
$\delta(q_0, a, b) \vdash (q_0, ab)$
$\delta(q_0, b, a) \vdash (q_0, ba)$
$\delta(q_0, a, a) \vdash (q_0, aa)$
$\delta(q_0, b, b) \vdash (q_0, bb)$

$\delta(q_0, c, a) \vdash (q_1, a)$
$\delta(q_0, c, b) \vdash (q_1, b)$
$\delta(q_0, c, z_0) \vdash (q, z_0)$
$\delta(q_1, a, a) \vdash (q, \Lambda)$
$\delta(q, b, b) \vdash (q, \Lambda)$
$\delta(q, \Lambda, z_0) \vdash (q_f, z_0)$

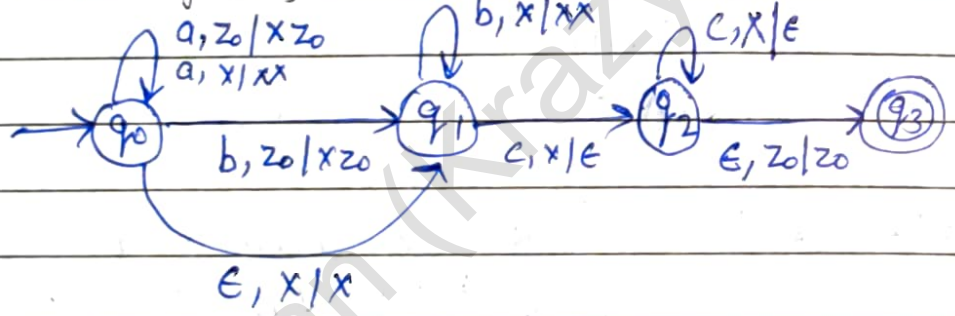
Q5- Construct a PDA for language $L = \{ww^r / w \in (a,b)^+\}$

Sol =



Q5.1- Find PDA for language $L = \{a^i b^j c^k / i,j,k \geq 0$ and $i+j=k\}$

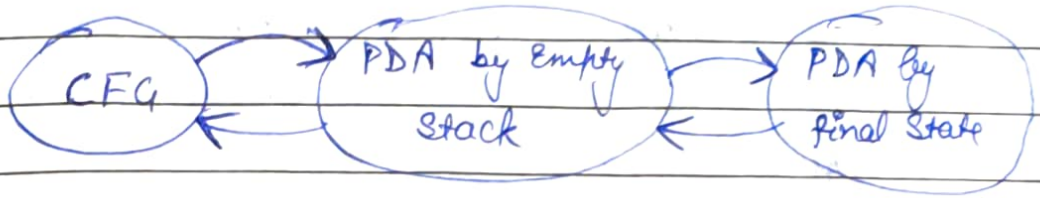
Sol -



String = { bc, ac, abcc, aacc, aabbcccc, bbcc, ... }

$\delta(q_0, a, Z_0) \vdash (q_0, XZ_0)$
$\delta(q_0, a, X) \vdash (q_0, XX)$
$\delta(q_0, b, Z_0) \vdash (q_1, XZ_0)$
$\delta(q_1, b, X) \vdash (q_1, XX)$
$\delta(q_0, \epsilon, X) \vdash (q_1, X)$
$\delta(q_1, c, X) \vdash (q_2, \epsilon)$
$\delta(q_2, c, X) \vdash (q_2, \epsilon)$
$\delta(q_2, \epsilon, Z_0) \vdash (q_3, Z_0)$

* PDA & CFL



* Construct of PDA from CFG.

→ From a given CFG $G = (V, T, P, S)$ we can construct PDA, that stimulates leftmost derivation of G.

→ The PDA accepts $L(G)$ by empty stack is given by —

$$M = (\{q\}, T, V \cup T, \delta, q, S, \phi)$$

Where δ is defined by

1- For each variable $A \in V$, include a transition
$\delta(q, \epsilon, A) \vdash \{(q, \alpha) \mid A \to \alpha$ is a production in G$\}$

2- For each terminal $a \in T$, include a transition
$\delta(q, a, a) \vdash (q, \epsilon)$

Q6- Construct PDA for the following grammar

$S \to aABB / aAA \quad A \to aBB / a \quad B \to bBB / A$

Sol= $P = (\{q\}, \{a, b\}, \{S, A, B, a, b\}, \delta, q, S, \phi)$

δ is defined as

$\delta(q, \Lambda, S) \vdash \{(q, aABB), (q, aAA)\}$
$\delta(q, \Lambda, A) \vdash \{(q, aBB), (q, a)\}$
$\delta(q, \Lambda, B) \vdash \{(q, bBB), (q, A)\}$
$\delta(q, a, a) \vdash (q, \Lambda)$
$\delta(q, b, b) \vdash (q, \Lambda)$

Q7- Construct a PDA for grammar
$S \to aAA$, $A \to aS/bS/a$
Also check whether the string 'abaaaa' is in PDA or not.

So- Let the PDA for given CFG be
$M = \{Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\}$
Where,
$Q = \{q\}$
$\Sigma = \{a, b\}$
$\Gamma = \{S, A, a, b\}$
$q_0 = \{q\}$
$Z_0 = \{S\}$
$F = \{q\}$
δ will defined as:
$\delta(q, \epsilon, S) \vdash \{q, aAA)$
$\delta(q, \epsilon, A) \vdash \{(q, aS), (q, bS), (q, a)\}$
$\delta(q, a, a) \vdash (q, \epsilon)$
$\delta(q, b, b) \vdash (q, \epsilon)$

Checking the string 'abaaaa' on the above PDA
$(q, abaaaa, S) \vdash (q, abaaaa, aAA)$
$\vdash (q, baaaa, AA)$
$\vdash (q, baaaa, bSA)$
$\vdash (q, aaaa, SA)$
$\vdash (q, aaaa, aAAA)$
$\vdash (q, aaa, AAA)$
$\vdash (q, aaa, aAA)$
$\vdash (q, aa, AA)$
$\vdash (q, aa, aA)$
$\vdash (q, a, A)$
$\vdash (q, a, a) \vdash (q, \epsilon, \epsilon)$ ⇒ Accepted

* Construction of CFG from PDA

If $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ is a PDA then CFG will be –

$$G = (V, \Sigma, P, S)$$

Where,

① V = Set of Non-Terminals

$$V = \{S\} \cup \{[q, z, q'] \mid q, q' \in Q, z \in \Gamma\}$$

② Productions P ⇒

i) S production –

$$S \to [q_0, z_0, q] \quad \text{for every } q \in Q$$

ii) For Pop Operation –

for $\delta(q, a, z) \vdash (q', \Lambda)$

$$[q, z, q'] \to a$$

iii) For Push & Nop –

for $\delta(q, a, z) \vdash (q_1, z_1 z_2 z_3 \cdots z_m)$

$$[q, z, q'] \to a\,[q_1, z_1, q_2][q_2, z_2, q_3] \cdots [q_m, z_m, q']$$

Q8- Construct a CFG equivalent to PDA

$P = (\{q_0, q_1\}, \{a, b\}, \{z_0, z\}, \delta, q_0, z_0, \phi)$

δ :

$\delta(q_0, b, z_0) \vdash (q_0, zz_0)$

$\delta(q_0, \Lambda, z_0) \vdash (q_0, \Lambda)$

$\delta(q_0, b, z) \vdash (q_0, zz)$

$\delta(q_0, a, z) \vdash (q_1, z)$

$\delta(q_1, b, z) \vdash (q_1, \Lambda)$

$\delta(q_1, a, z_0) \vdash (q_0, z_0)$

Sol = Let $G = (V, \{a, b\}, P, S)$

Where,

$$V = \{ S, [q_0, z_0, q_0], [q_0, z_0, q_1], [q_0, z, q_0], [q_0, z, q_1], [q_1, z_0, q_0], [q_1, z_0, q_1], [q_1, z, q_0], [q_1, z, q_1] \}$$

Productions ⇒

S-Production -

$P_1 : S \to [q_0, z_0, q_0]$

$P_2 : S \to [q_0, z_0, q_1]$

$\delta(q_0, b, z_0) \vdash (q_0, z z_0)$

$P_3 : [q_0, z_0, q_0] \to b[q_0, z, q_0][q_0, z_0, q_0]$

$P_4 : [q_0, z_0, q_0] \to b[q_0, z, q_1][q_1, z_0, q_0]$

$P_5 : [q_0, z_0, q_1] \to b[q_0, z, q_0][q_0, z_0, q_1]$

$P_6 : [q_0, z_0, q_1] \to b[q_0, z, q_1][q_1, z_0, q_1]$

$\delta(q_0, \Lambda, z_0) \vdash (q_0, \Lambda)$

$P_7 : [q_0, z_0, q_0] \to \Lambda$

$\delta(q_0, b, z) \vdash (q_0, zz)$

$P_8 : [q_0, z, q_0] \to b[q_0, z, q_0][q_0, z, q_0]$

$P_9 : [q_0, z, q_0] \to b[q_0, z, q_1][q_1, z, q_0]$

$P_{10} : [q_0, z, q_1] \to b[q_0, z, q_0][q_0, z, q_1]$

$P_{11} : [q_0, z, q_1] \to b[q_0, z, q_1][q_1, z, q_1]$

$\delta(q_0, a, z) \vdash (q_1, z)$

$P_{12}: [q_0, z, q_0] \rightarrow a[q_1, z, q_0]$

$P_{13}: [q_0, z, q_1] \rightarrow a[q_1, z, q_1]$

$\delta(q_1, b, z) \vdash (q_1, \Lambda)$

$P_{14}: [q_1, z, q_1] \rightarrow b$

$\delta(q_1, a, z_0) \vdash (q_0, z_0)$

$P_{15}: [q_1, z_0, q_0] \rightarrow a[q_0, z_0, q_0]$

$P_{16}: [q_1, z_0, q_1] \rightarrow a[q_0, z_0, q_1]$

From P_1 to P_{16} are the productions.

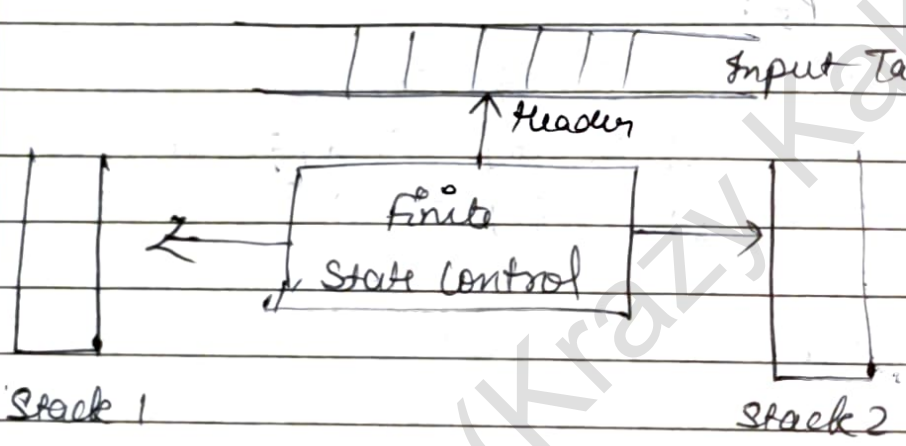
* Two Stack PDA -

→ A two stack PDA is similar to a PDA but it has two stack instead of one.

→ A two stack PDA has six tuples.

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

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \times \Gamma \rightarrow Q \times \Gamma^* \times \Gamma^*$$

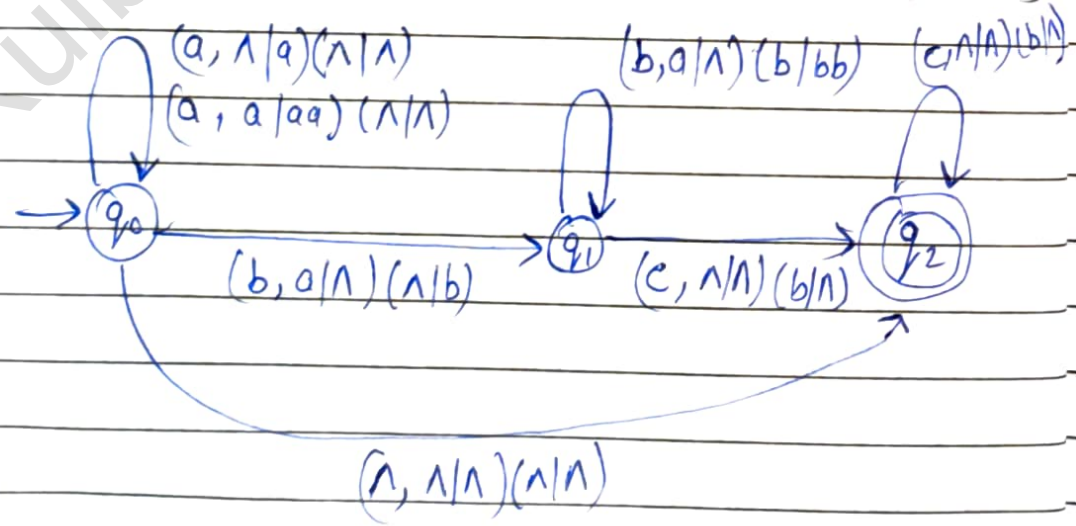


$$\delta(q_0, a, Z_0, Z_1) \vdash (q_1, Z_1, Z_2)$$

Current State, Current I/P, Stack 1, Stack 2, Next State, Stack 1, Stack 2

Q- Design two stack PDA for $L = \{a^n b^n c^n \mid n \geq 0\}$

Sol =



$\delta(q_0, a, \Lambda, \Lambda) \vdash (q_0, a, \Lambda)$

$\delta(q_0, a, a, \Lambda) \vdash (q_0, aa, \Lambda)$

$\delta(q_0, b, a, \Lambda) \vdash (q_1, \Lambda, b)$

$\delta(q_1, b, a, b) \vdash (q_1, \Lambda, bb)$

$\delta(q_1, c, \Lambda, b) \vdash (q_2, \Lambda, \Lambda)$

$\delta(q_2, c, \Lambda, b) \vdash (q_2, \Lambda, \Lambda)$

$\delta(q_0, \Lambda, \Lambda, \Lambda) \vdash (q_2, \Lambda, \Lambda)$

* Pumping Lemma for CFL.

→ Pumping lemma is used to prove that certain languages are not CFL.

→ Let L(G) is CFL then following condition must be satisfied.

i) Every $z \in L(G)$ with $|z| \geq n$, where n is natural number.
And $z = uvwxy$ for some string

ii) $|vx| \geq 1$

iii) $|vwx| \leq n$

iv) $uv^k wx^k y \in L$ for all $k \geq 0$

Q10- Prove that the language $L = \{a^i b^i c^i \mid i \geq 1\}$ is not CFL.

Sol= Assume that the language L is CFL and n is the natural number.

$L = \{abc, aabbcc, aaabbb, ccc, aaaabbbb, ccccc \ldots\}$

Let $n = 10$

So, taking $z = aaaa\,bbbb\,cccc$

Let $z = uvwxy$

$z = \underbrace{aaaa}_{u}\,\underbrace{bbbb}_{v}\,\underbrace{cccc}_{w}\,\underbrace{}_{x}\,\underbrace{}_{y}$

$uv^k wx^k y \in L$ for all $k \geq 1$

for $k=1$, $aaaabbbbcccc \in L$

for $k=2$, $aaaaaabbbbcccccc \notin L$

So, given CFL Grammar is not CFL.

UNIT - 5

TURING MACHINES & RECURSIVE THEORY

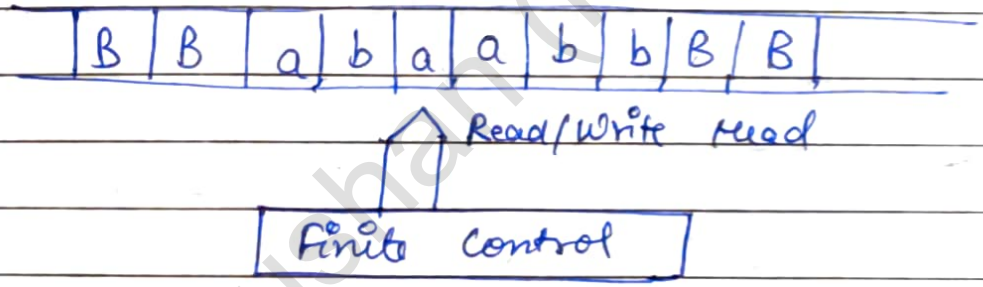
Date
Page

→ Turing m/c is the most powerful m/c.
→ Made by Alan Turing, in 1936.
→ TM is used to accept Recursively Enumerable languages (Type-0 Grammar).

Features of TM

i) It is able to remember long sequence of I/P, because of infinite memory.
ii) Input head can move left or right
iii) can be used to generate output based on certain input.

Basic Model of TM



→ The TM has finite control connected to R/W head.
→ It has one tape which is divided into a number of cells.
→ Each cell can store only one symbol
R/W head can examine one cell at a time.

* Formal Definition of Turing Machine -

TM is defined by using 7 - tuple

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

Where,

Q: Finite set of states of finite control.

Σ: Finite set of I/P symbols.

Γ: The complete set of tape symbols, including Blank symbol (B).

δ: Transit function.

$$Q \times \Gamma \longrightarrow Q \times \Gamma \times (L, R, N)$$

q_0: Initial State.

B: Blank Symbol (also #).

F: Set of final state (Halt State)

Representation of TM

→ Instantaneous Descripter

$\delta(q_0, a) = (q, b, R)$

q_0 → Current State, a → Current I/P, q → Next State, b → Rewrite Symbol, R → Movement

→ Transition Diagram

→ Transition Diagram

Q1- Design a TM that accept the language
$L = \{a^n b^n / n \geq 1\}$

So = Each cycle will consist of the following steps -

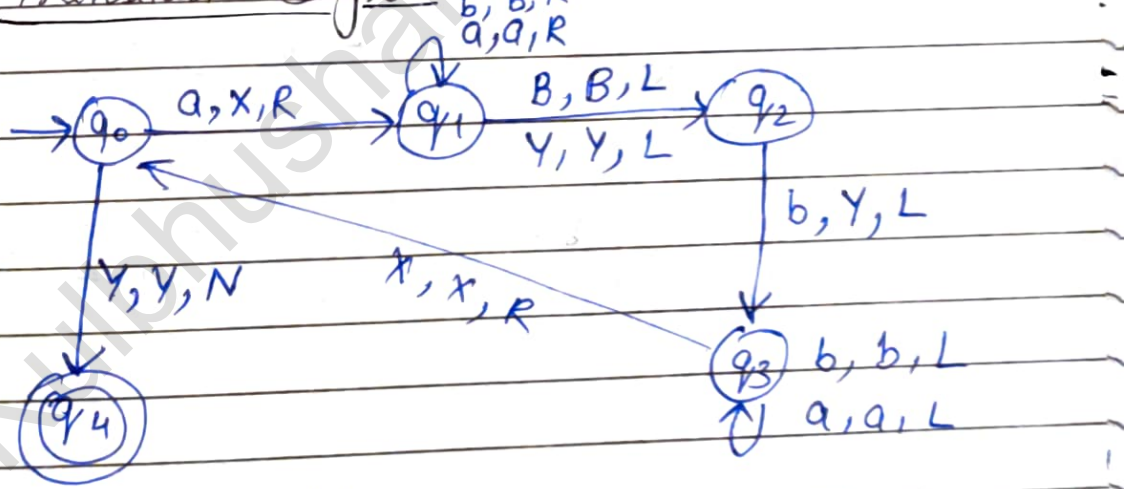
1- leftmost 'a' changes to X.
2- Rightmost 'b' changes to Y.
3- Had comeback to first 'a'.

I | B a a a b b b B | B X a a b b Y B |

II | B X a a b b Y B | B X X a b Y Y B |

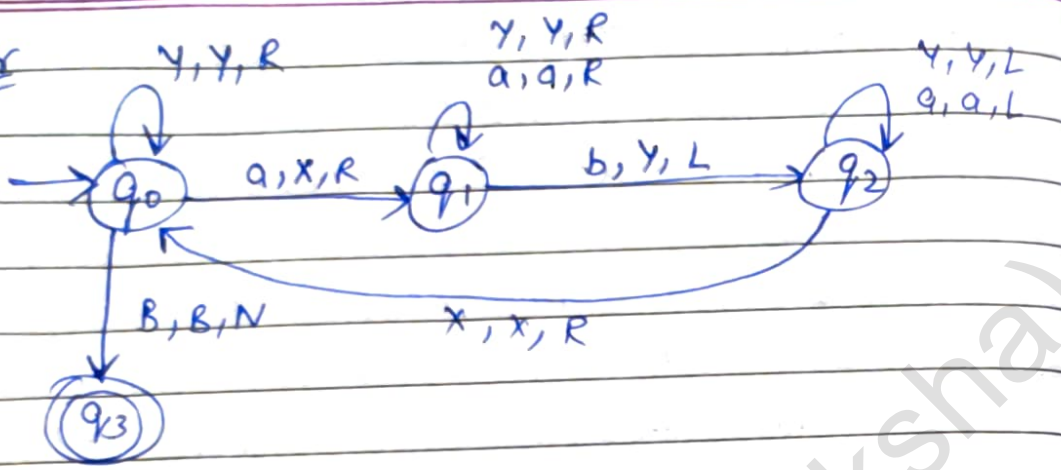
III | B X X a b Y Y B | B X X X Y Y Y B |

Transition Diagram -



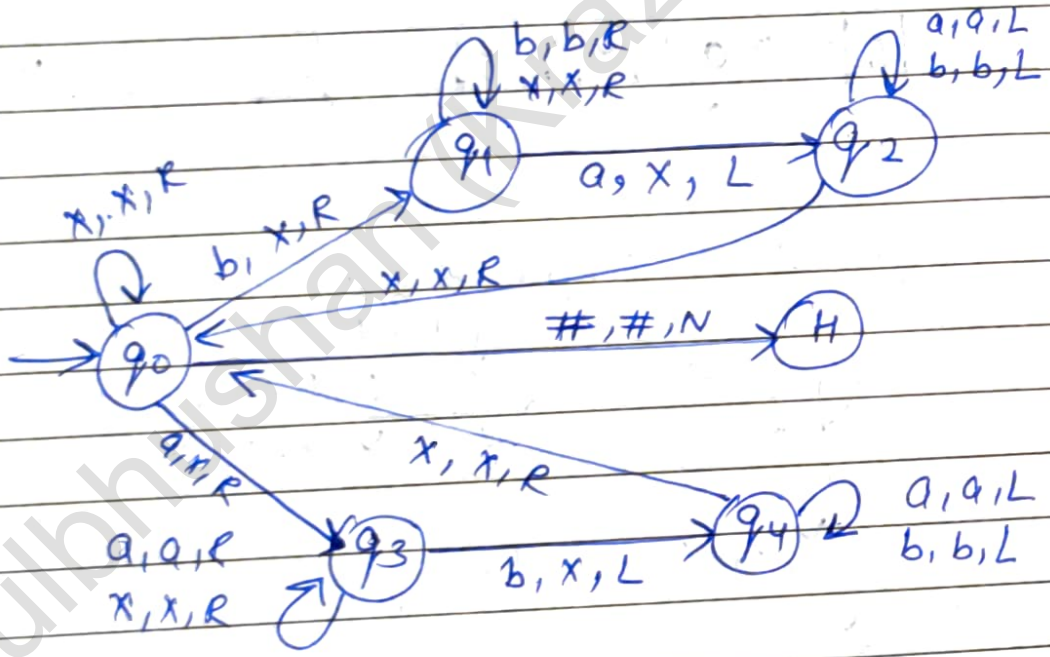
Transition Table -

Q \ Γ	a	b	X	Y	B
q_0	(q_1, X, R)	–	–	(q_4, Y, N)	–
q_1	(q_1, a, R)	(q_1, b, R)	–	(q_2, Y, L)	(q_2, B, L)
q_2	–	(q_3, Y, L)	–	–	–
q_3	(q_3, a, L)	(q_3, b, L)	(q_0, X, R)	–	–
q_4	q_4	q_4	q_4	q_4	q_4 ← Halt State



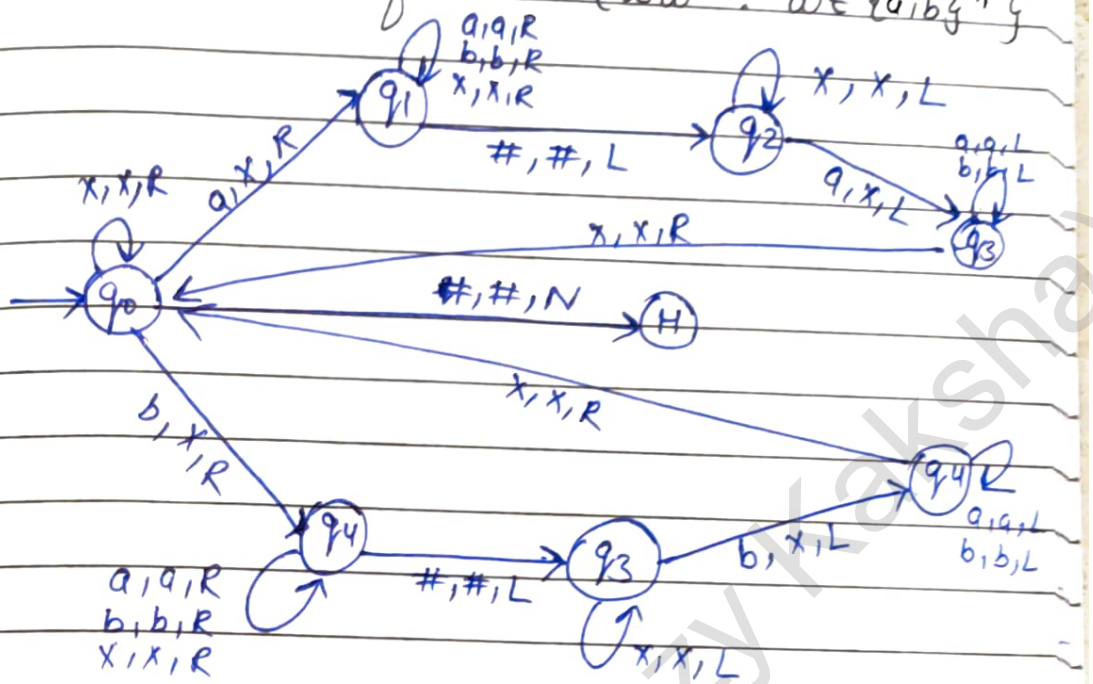
Q2- Construct a TM for language
L = { Set of strings consists of equal no. of a & b }

So =



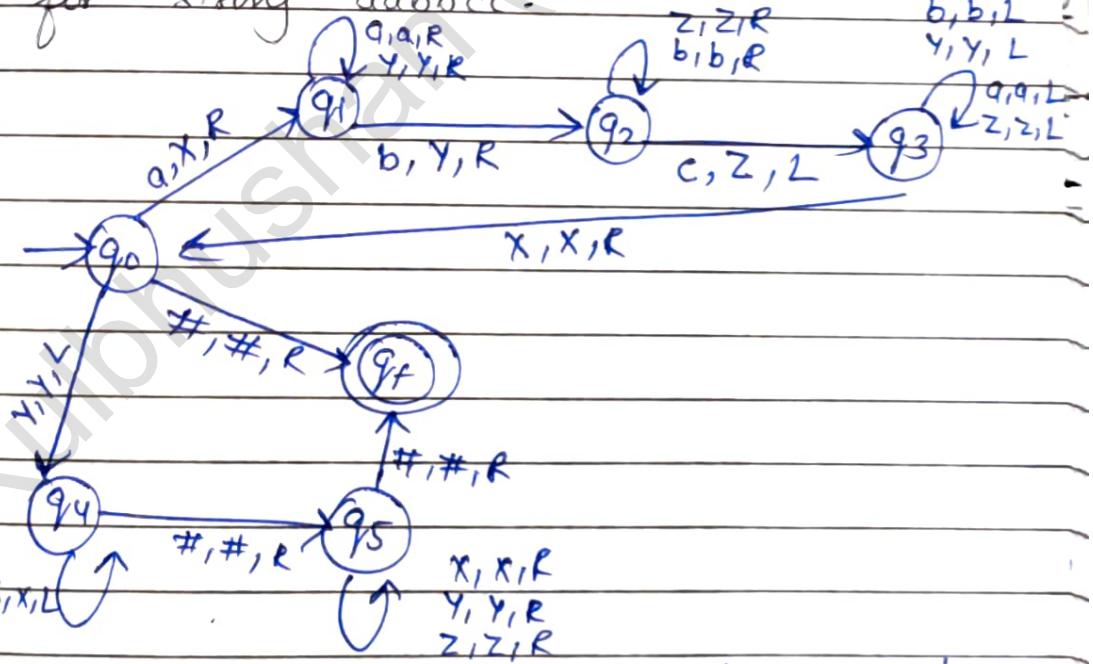
Q3- Construct a TM for $L = \{ww^R : w \in \{a,b\}^*\}$

Sol=



Q4- Construct a TM for $L = \{a^n b^n c^n \mid n \geq 0\}$ & check for string aabbcc.

Sol=



Testing - $\underset{q_0}{\downarrow}aabbcc \vdash X\underset{q_1}{\downarrow}abbcc \vdash Xa\underset{q_1}{\downarrow}bbcc \vdash XaY\underset{q_2}{\downarrow}bcc$

$\vdash XaYb\underset{q_2}{\downarrow}cc \vdash XaYbZ\underset{q_3}{\downarrow}c \vdash XaY\underset{q_3}{\downarrow}bZc$

$\vdash X\underset{q_3}{\downarrow}aYbZc \vdash X\underset{q_3}{\downarrow}aYbZc \vdash X\underset{q_0}{\downarrow}aYbZc \vdash$

$XXYYZZ$ q_f

* Variants of Turing Machine

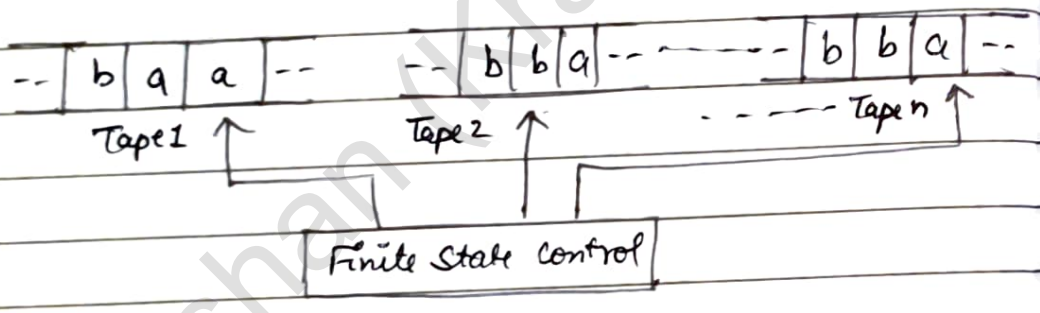
1) Multitape TM-

A multitape TM is a TM with several tapes, each tape has its own independent controlled R/W head.

Definition -

An n-tape TM $M = (Q, \Sigma, \Gamma, q_0, \delta, B, F)$ where $Q, \Sigma, \Gamma, q_0, F$ are same as in standard TM but δ is

$$\delta : Q \times \Gamma^n \rightarrow Q \times \Gamma^n \times \{L, R\}^n$$



ii) Non-Deterministic TM -

A non-deterministic TM is an automata as given by standard definition of TM except that δ is now a function

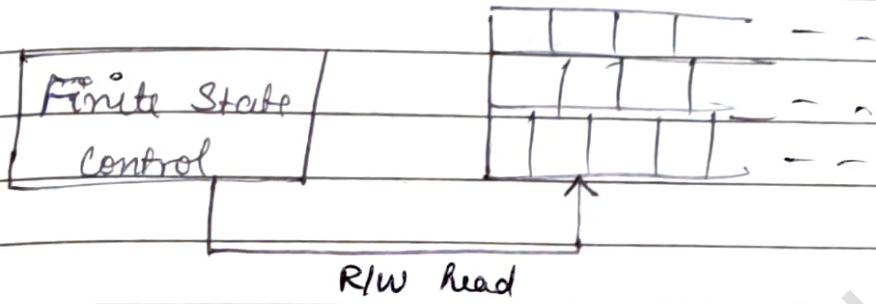
$$\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, R\}}$$

e.g.,

$$\delta(q, x) \rightarrow \{(q_1, y_1, L), (q_2, y_2, L) \dots (q_n, y_n, R)\}$$

iii) **Multidimension TM -**

Tapes extended infinitely in more than one direction.



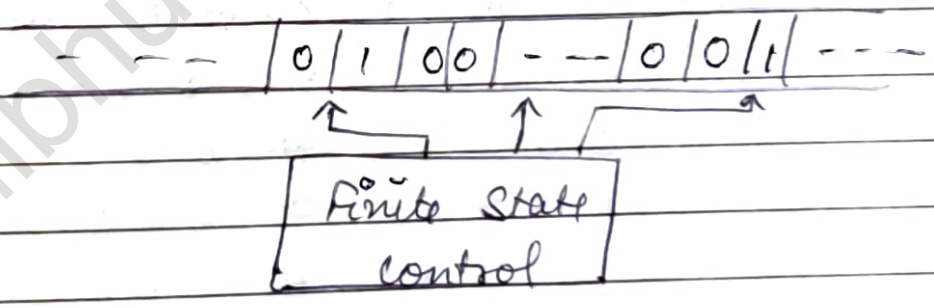
$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, U, D\}$$

iv) **Multihead TM -**

A multihead TM can be defined as TM with a single tape & a single finite state control but with multiple independent R/W heads & defined as

$$\delta : Q \times \Gamma_T \rightarrow Q \times \Gamma_T \times \{L, R\}^n$$

where, $\Gamma_T = \Gamma \times \Gamma \times \Gamma \times \dots \Gamma$ & n is the no. of R/W heads.



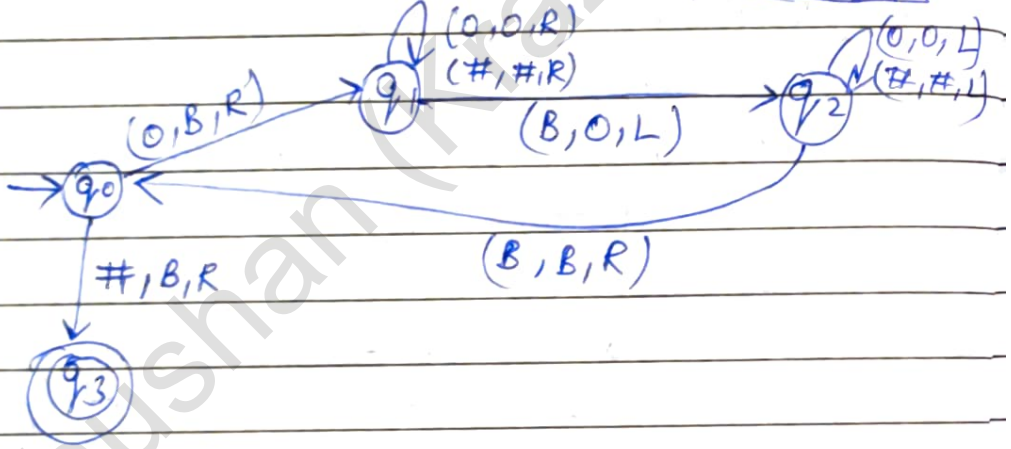
TM for computation

Q1- Design a TM that computes a function $f(m,n) = m+n$.

Sol= Let $m=5$ & $n=3$

$m+n = 8$

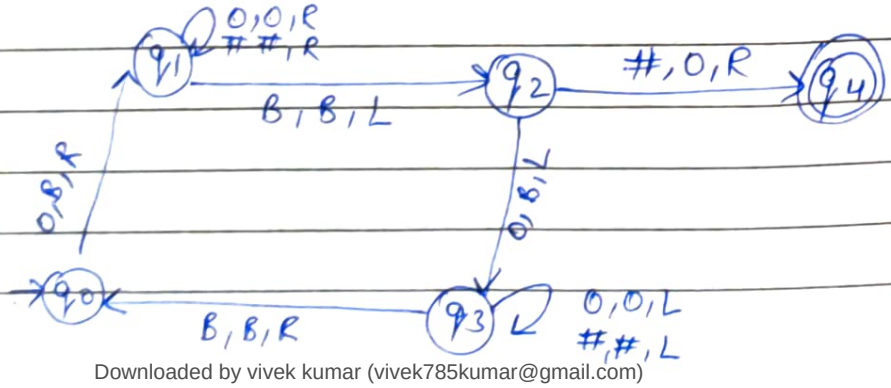
	0	0	0	0	0	#	0	0	0	B
B	0	0	0	0	#	0	0	0	0	B
B	B	0	0	0	#	0	0	0	0	0	B



Q2- Design a TM to compute $m-n$.

Sol= Let $m=5$ & $n=3$ ⇒ $5-3=2$

0	0	0	0	0	#	0	0	0
B	0	0	0	0	#	0	0	B
B	B	0	0	0	#	0	B	B
B	B	B	0	0	#	B	B	B



Q3- Design a TM to compute m * n

So:- Let m = 2, n = 3 ⇒ 2 * 3 = 6

0	0	#	0	0	0	#	B	B	B	...

B 0 # X 0 0 # 0 B B B ...

B 0 # X X 0 # 0 0 B B ...

B 0 # X X X # 0 0 0 B ...

B 0 # 0 0 0 # 0 0 0 B ...

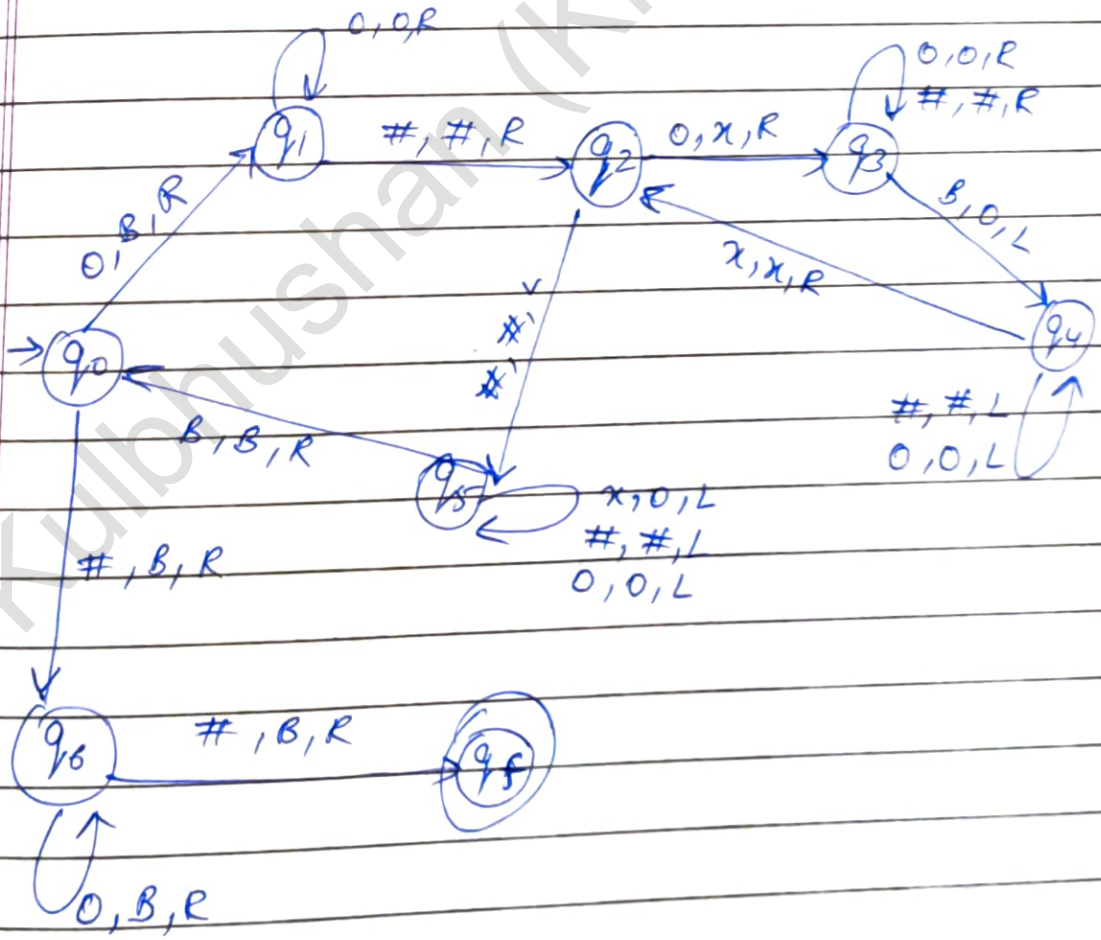
B B # X 0 0 # 0 0 0 0 B B B

B B # X X 0 # 0 0 0 0 0 B B

B B # X X X # 0 0 0 0 0 0 B

B B # 0 0 0 # 0 0 0 0 0 0 B ...

B B B B B B B 0 0 0 0 0 0 0



* Universal Turing Machine

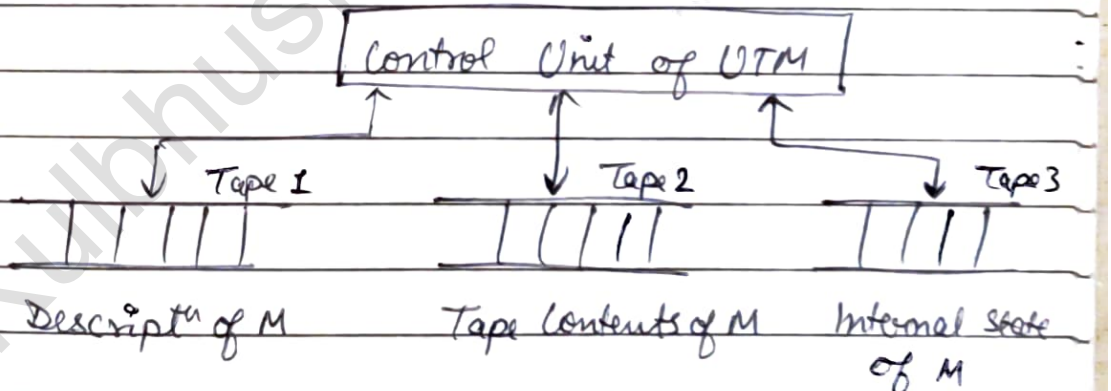
→ A general purpose computer can be programmed to solve different types of problem.

→ A TM can also behave like a general purpose computer.

→ UTM is a kind of TM which can simulate any other TM. In other words, UTM is a single m/c used to compute any computable sequence.

→ A UTM M_u is an automaton given as i/p the description of any TM M and a string w, can simulate the computation of M on w.

Model of UTM →



Encoding Scheme defined As-

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be TM

$Q = \{q_1, q_2 \dots q_n\}$

$F = q_2$ then

Encode State, $q_1 = 1$

$q_2 = 11$

$q_3 = 111$ & so on

Similarly,

$a_1 = 1$, $a_2 = 11$ $a_3 = 111$ ...

Moving direction -

$L = 1$ $R = 11$

'0' → Separator

Let

$\delta(q_1, a_2) = (q_2, a_3, L)$

then encoded as

10110110111101

⇒ UTM first looks at tape 2 & 3 then check for decision to be made in tape 1.

* Linear Bounded Automata (LBA)

→ It is a multitrack Non-Deterministic TM with a tape of some bounded finite length.

→ Input alphabet contains 2 special symbols which serves as End Markers.

→ LBA is a 8-tuple machine.

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

Where, M_L → Left Marker

M_R → Right Marker

$$\delta : (Q \times \Gamma) \longrightarrow Q \times \Gamma \times \{L, R\}$$

* Church's Thesis -

This is proposed in 1986 by two mathematician.

→ Church thesis states that any algorithm that can be performed on any computing machine can be performed on a Turing machine as well.

→ No one has yet been able to suggest a problem solvable by algorithm for which a TM program cannot be written.

→ Alternating model have been proposed for mechanical computation like λ-Calculus But none of them are more powerful than Turing Machine.

* Recursive & Recursively Enumerable Language

Recursive Language (REC)

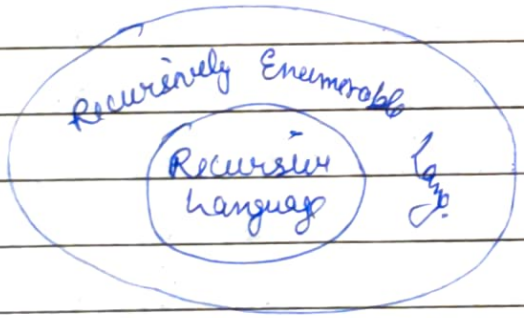
→ Language is REC if there exists a TM which accept all strings $w \in L$ & reject $w \notin L$.

→ TM will halt every time for every input string.

→ A language L is decidable if it is Recursive language.

Recursive Enumerable Language (RE)

→ Language is RE if there exists a TM which will accept for all I/P strings $w \in L$, but may or may not halt for all $w \notin L$.



* Halting Problem

When a TM M is given, given an input w to the m/c M.

So we can not design a generalized algorithm which can appropriately determine, whether the given program will halt or not.

* Post's Correspondance Problem (PCP)

→ PCP is an undecidable problem that turns out to be a very helpful tool for proving problems in logic or in formal language theory to be undecidable.

→ Let A & B be two non-empty list of strings over Σ.

$$A = \{x_1, x_2, x_3 \dots x_k\}$$
$$B = \{y_1, y_2, y_3 \dots y_k\}$$

→ We say, there is a Post Correspondence b/w A & B if there is a sequence of one or more integers i, j, k ... m such that the string

$$x_i x_j \dots x_m = y_i y_j \dots y_m$$

Note: MPCP (Modified PCP) ⇒ in which $i = 1$ always.

Q1 Determine the Solution for the following instance of PCP

i	List A	List B	
1	1	111	$\frac{1}{111}$
2	10111	10	$\frac{10111}{10}$
3	10	0	$\frac{10}{0}$

So ∴ Sequence = 2, 1, 1, 3

A: 10111110

B: 10111110