Automata Theory And Compiler Design:-

Automata:-

It is a "abstract machine (or) a model".

Automata Theory:-

It is a study of "abstract machine" (or) a "computing device".

History of Automata Theory:-

→ In 1930's, "Allen Turing" studied Abstract machine.

→ He studied this theory to know "what a computing device could do and what a computing device couldnot do".

→ In 1940's and 1950's, "Finite Automata" was studied by some scientists.

→ We use this study in "software and haldware", "computing device".

→ In late 1950's, "chomsky" studied, Grammer, these Grammers are used the "development of software".

→ In 1960's, "S. cook", he extended study of "Allen Turing" and he was able to seperate Problems that can be solved by computer and that

can't be solved by computer.

What was the reason to study Automata Theory:

→ It was used to design and develop the some softwares.

→ We use this in lexical analyser software, digital circuits software
  ↳ Breaks the Programs into logical units
    ↳ key words
      Identifiers
      Punchuations

to verify some systems,
to web scan systems
we use the automata Theory.

Introduction To Finite Automata Theory:-

Finite Automata:-

The Automata that exists in "finite" no. of states.

→ The Purpose of the "state" is to "remember certain portion of system history".

Advantage:-

→ We can develop the device/system in fixed no. of states/resources.

How to represent Finite Automata:-

→ To represent finite Automata, we use

(circle) ○ → To represent states

(start) ⟶ ⇒ To represent start state.

It is label to the start state. It will be there in arrow.

(double circle) ◎ → To represent the final states.
↓ Accepting states.

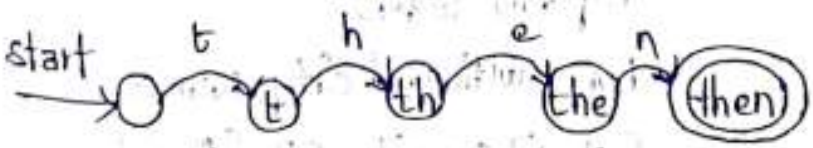
(arc) ↷ → To represent transistion. [movement].

It is labelled with the some input.

How To model a softwares-

→ The finite automaton for the part of lexical analyser that recognises the string "Then".

→ It requires 5 states. It refers to the Prefixes of the "then" [t, th, the, then, ○ (one empty state)].



Structural Representation:-

Grammars

→ used in the development of the processing the data with a recursive structure.

Regular Expressions

→ used for the structure or organise the data.

Eg:- UNIX style

[A-z] [a-z]*[ ]

[A-z].[A-Z]-

[A-Z] → any capital letter

[a-z] → any small letter

* → any no. of occurances

[ ] → a blank space

Eg:- Delhi IN

The given regular expression represent a capitalized word followed by blank space followed by 2 capital letters.

IN is a single name followed by two letters.

For multiword city Names

[A-Z] [a-z]*[ ] [A-Z]

West Bengal IN [a-z]*

[A-Z] [A-Z]

Automata & Complexity:-

It discusses limitations of computations.

→ What a computer can do at all?

This study is called decidability and problems that can be solved with sometimes are called decidable.

→ What a computer can do efficiently?

This study is called intractability and the problems that can be solved with no more time called "tractable (or) undecidable".

Central Concepts of Automata Theory:-

Alphabet: It is a finite non empty set of symbols.

It is denoted by "Σ".

Eg:- $\Sigma = \{0,1\}$ -- Binary Alphabet.

$\Sigma = \{a, \ldots, z\}$ -- Small letter Alphabet.

ASCII: character [A-65-ASCII-values]

String:- It is a finite sequence of input symbols chosen from an alphabet sigma.

Eg:- 0111 is a string: choose from a binary alphabet. $\Sigma = \{0,1\}$

It is represented by ω.

chosen from an alphabet Σ.

1) Empty string:- It is a string that contains zero occurances of input symbols.

→ It is represented as ϵ (Epsila).

→ It can be a string that can be any alphabet taken from

2) Length of the string:-

→ It can be no.of Positions of symbols in the string. → It is represented as "|w|"

Eg:- $w = 1010111$

The length of the string $|w| = 7$

3) Powers of the string:-

→ If Σ is an alphabet then Σ^k is a set of strings of length k.

Eg:- $\Sigma = \{a, b\}$

$\Sigma^0 = \phi$, $\Sigma^1 = \{a, b\}$, $\Sigma^2 = \{aa, ab, ba, bb\}$

$\Sigma^3 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$.

→ The set of all strings over some alphabet Σ is denoted by "Σ^*".

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

→ The set of all non empty strings is denoted by "Σ^+".

$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cdots\cdots$

The relation b/w the both strings is "$\Sigma^* = \Sigma^+ \cup \epsilon$".

4) Concatenation of strings:-

If x and y are two strings then the concatenation of x,y is represented by a "xy" where the string "x followed by y".

Eg:- $x = 10001$, $y = 00110$. } $\Sigma = \{0,1\}$

$xy = 1000100110$

→ $\epsilon x = x\epsilon = x$.

↳ empty string [identity of concatenation].

Language:- It is a set of strings choosen from the Σ^*.

Where Σ is some alphabet.

Eg:- $\Sigma = \{0,1\}$.

L = set of strings with equal no of 0's and equal no of 1's.

$L = \{01, 10, 0011, 1100, 1001, 0110, 0101, 1010 \ldots\}$

→ It is represented by "L".

→ If Σ is an alphabet and L is subset of Σ^*, then L is a language.

→ If L is a language over Σ then it is also a language over superset of Σ.

Eg:- Set of binary integers whose value is prime

L = {10, 11, 101, 111, ...}

→ Σ^* is also a language over the alphabet Σ.

→ ϕ is empty language $\phi \neq \{\epsilon\}$

↳ empty ↳ language with empty string ε

↳ empty language

Problem:-

It is a question of deciding whether a string belongs to some particular language or not.

Eg:- L = {0, 100, 10,000, 100, 010, 110, ...}

determine 000 is part of Given string or not (Yes)

→ It determine the membership of the string in a language.

→ If Σ is an alphabet and L is a language the question determining whether the string 'w' is there in language L or not is a "problem."

→ There are two types of finite Automata:

1) Deterministic Finite Automata:-

→ Deterministic refers that for each input symbol there is only one state to which finite automata can transition from its current state.

→ DFA is represented by "5 tuple" notation. It consits of 5 : $A = \{Q, \Sigma, \delta, q_0, F\}$

→ It consists of i) A set of states. It is represented by "Q".

ii) A set of input symbols, represented by "Σ".

iii) A transition function $\delta(q_0, 1)$ which takes two arguments "state and "input symbol" and give one output.

iv) starting state, represented as "q_0"

v) set of final states are (or) Accepting states, represented as "F".

Notations of DFA:-

→ There are two types of Notations:

1) Transition Diagram:-

It is a Graph that contains

i) For each state, there is a node

(ii) for each input symbol and for each state transition function δ is represented by an arc labelled with an input symbol.

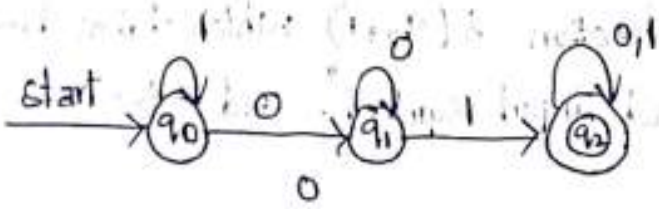
(iii) There is an arrow into the starting state labelled as start.

(iv) The final state or states must be in double circle ◎ and other states must be in single circle ○.

Question:- Construct DFA which accepts all strings of "0's and 1's" that have the sequence "01"

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

$L = \{01, 001, 0101, 011, 101, \ldots\}$



2) Transition Table:-

→ Transition Table is a tabular representation of function δ.

→ Rows corresponds to states.

→ columns corresponds to input symbols.

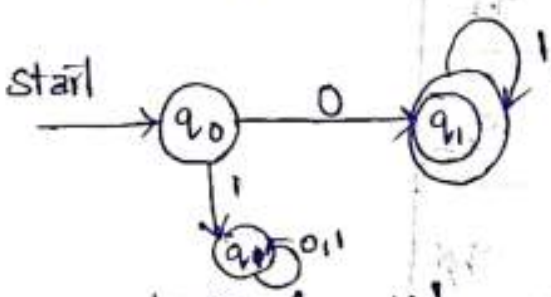
→ Entry corresponds to state which results from $\delta(q, a)$.

	0	1
→ q_0	q_1	q_0
q_1	q_1	q_2
* q_2	q_2	q_2

→ construct DFA with accepts all strings over alphabet $\Sigma = \{0,1\}$ starts with "0".

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

$L = \{0, 01, 000, 001, 010, 011, \ldots\}$



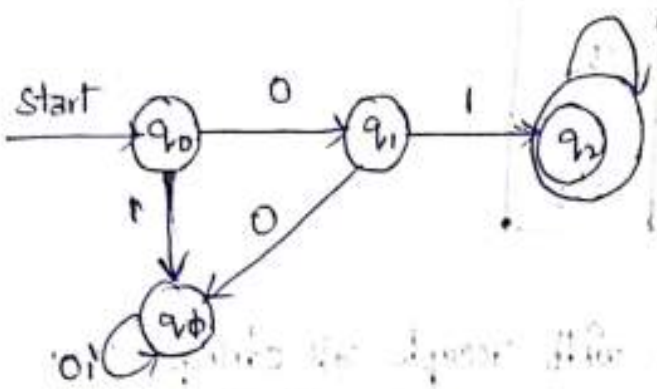
	0	1
→ q_0	q_0	q_1
* q_1	q_1	q_1

	0	1
→ q_0	q_1	q_ϕ
q_ϕ	q_ϕ	q_ϕ
* q_1	q_1	q_1

→ Construct DFA with accepts all string over alphabet $\Sigma = \{0,1\}$ starts with "01".

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

$L = \{01, 00, 001, 0101, 011, 0100, 0101, \ldots\}$

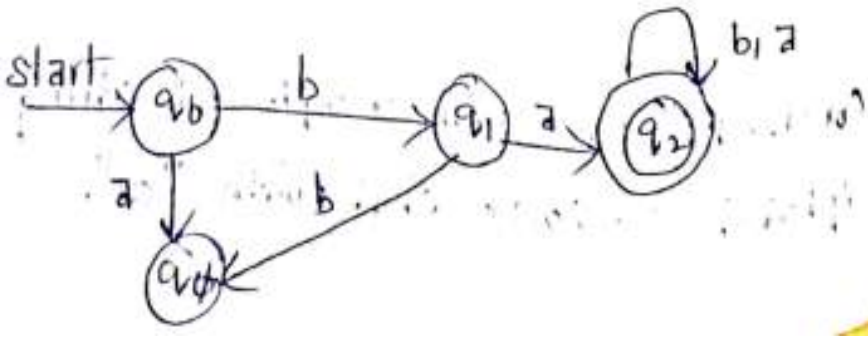


	0	1
$\rightarrow q_0$	q_1	q_ϕ
q_1	q_ϕ	q_2
$*q_2$	q_2	q_2
q_ϕ	q_0	q_ϕ

→ Construct DFA with accepts all string over alphabet $\Sigma = \{a, b\}$ starts with "ba".

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

$L = \{ba, bab, babb, baba, \ldots\}$



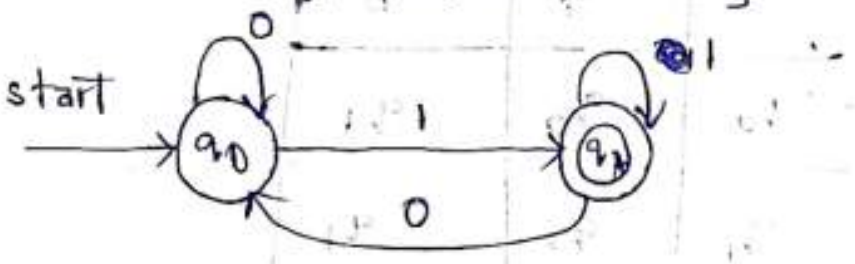
	a	b
$\rightarrow q_0$	q_ϕ	q_1
q_1	q_2	q_ϕ
$*q_2$	q_2	q_2
q_ϕ	q_ϕ	q_ϕ

for

→ Construct DFA, all strings over alphabet $\Sigma = \{0,1\}$ which ends with "1".

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

$L = \{01, 11, 001, 011, 101, 111 \ldots\}$

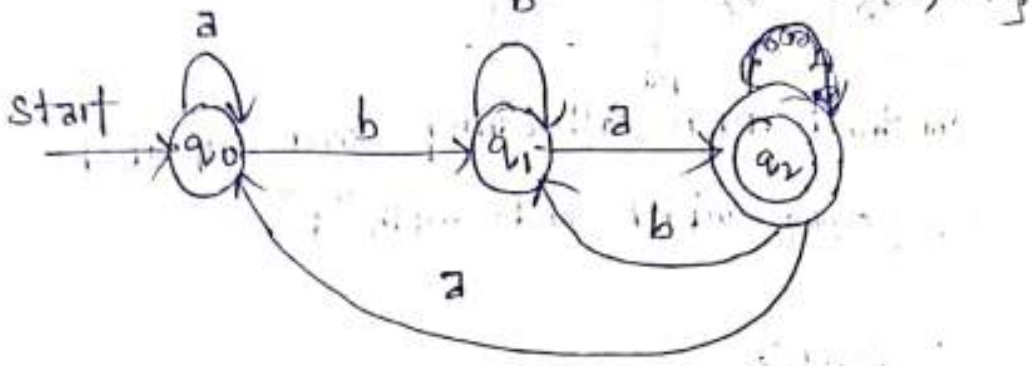


	0	1
$\rightarrow q_0$	q_0	q_1
$*q_1$	q_0	q_1

→ Construct DFA for all strings over the alphabet $\Sigma = \{a, b\}$ which ends with "ba"

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

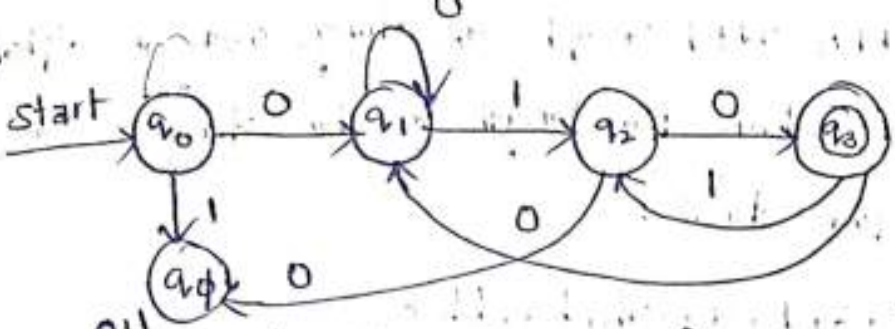
$L = \{ba, aba, bba, aaba, abba, baba, bbba, \ldots\}$



	a	b
→ q_0	q_0	q_1
q_1	q_2	q_1
* q_2	q_0	q_1

→ Construct DFA for all strings over the alphabet $\Sigma = \{0, 1\}$ which ends with 010

$L = \{010, 0010, 1010, \ldots\}$

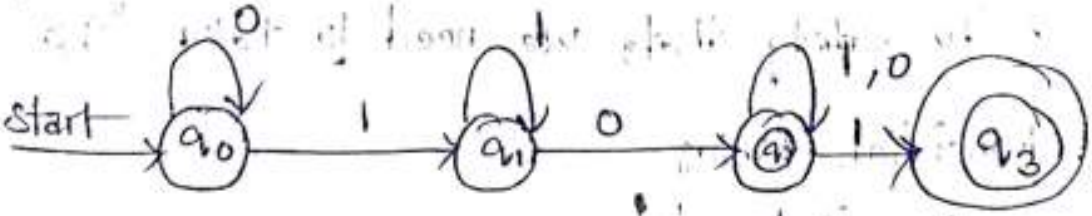


	0	1
→ q_0	q_1	q_ϕ
⊖ q_1	q_1	q_2
q_2	q_3	q_ϕ
* q_3	q_1	q_2
q_ϕ	q_ϕ	q_ϕ

→ construct DFA for all strings over the alphabet "011" which contains 10ϕ

010, 101 1101

$L = \{10, 10101, 1010, 1011, 0010, \ldots\}$



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

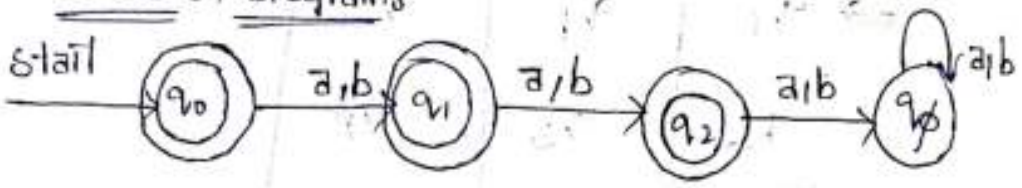
→ Construct DFA which accepts all strings over an alphabet $\Sigma = \{a, b\}$ where the length of the string is less than or equals to "2":

$L = \{\epsilon, a, b, aa, ab, ba, bb\}$

→ It depends on the length of strings not min "LOS".

→ The length of the string = 2 => 3 states

Transistion Diagram:-



→ This DFA satisfies all the strings of the length of the string is "≤ 2".

→ There is no transistion of q_2, if we take self transistion then we get more than 2 states.

→ But there is no numbers matching with the more than 2 in the language.

→ To satisty that, we need to take "q_ϕ".

Transistion Table:-

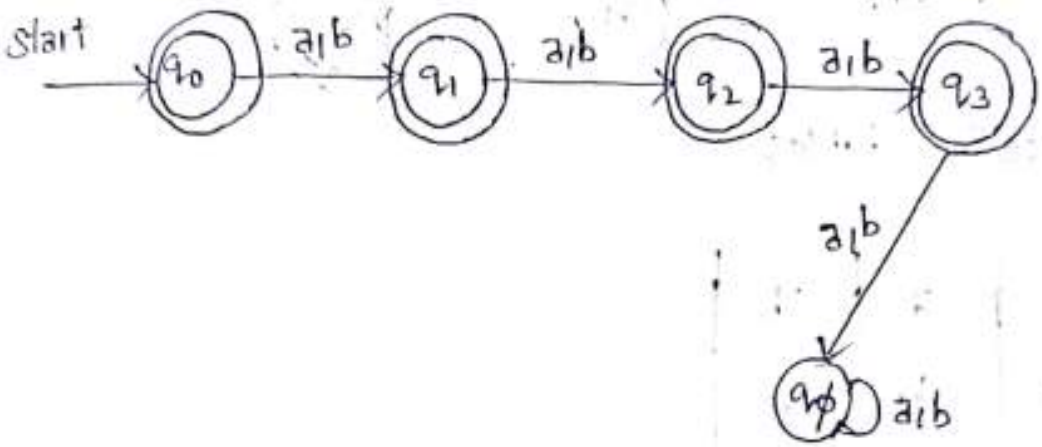
	a	b
→* q_0	q_1	q_2
* q_1	q_2	q_2
* q_2	q_ϕ	q_ϕ
q_ϕ	q_ϕ	q_ϕ

→ construct DFA which accepts all strings over an alphabet $\Sigma = \{a, b\}$ where the length of the string is less than or equals to "3".

$L = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb\}$

→ The length of the string is 3 ⇒ 4 states.

Transition Diagram:-



Transition Table:-

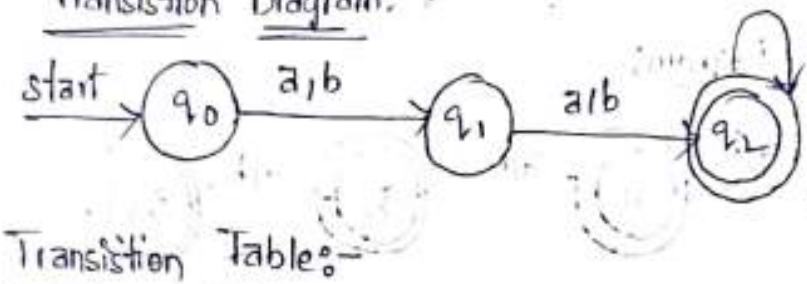
	a	b
→ * q_0	q_1	q_1
* q_1	q_2	q_2
* q_2	q_3	q_3
* q_3	q_ϕ	q_ϕ
q_ϕ	q_ϕ	q_ϕ

Ex:- → Construct DFA which accepts all the strings over the alphabet $\Sigma = \{a, b\}$ where the length of the string is "greater than or equal to" 2

$L = \{aa, ab, ba, bb, aaa, aab, aba, abb, \ldots\}$

→ The length of the string is 2 = 3 states

Transition Diagram:-



Transition Table:-

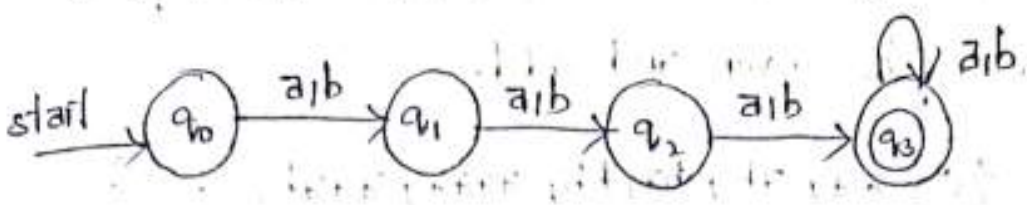
	a	b
→ q_0	q_1	q_1
q_1	q_2	q_2
* q_2	q_2	q_2

→ Construct DFA which accepts all the strings over the alphabet $\Sigma = \{a, b\}$ where the length of the string is "Greater than" or equal to "3"

$L = \{aaa, aab, aba, abb, aaaa, aaab, \ldots\}$

The length of the string is 3 = 4 states:

Transition Diagram:-



Transition Table:-

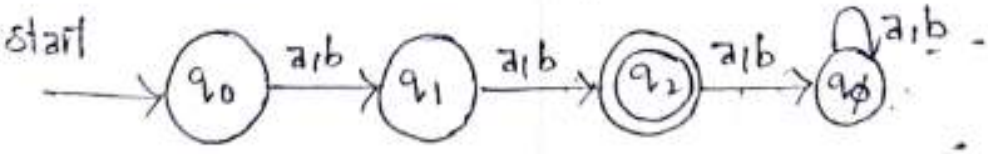
	a	b
→q_0	q_1	q_7
q_1	q_2	q_2
q_2	q_3	q_3
q_3	q_8	q_4

→ construct DFA which accepts all the strings over the alphabet $\Sigma = \{a|b\}$ where the "length" of the string is "2".

$L = \{aa, ab, ba, bb\}$

→ The length of the string is 2 = 3states

Transition Diagram:-



Transition Table:-

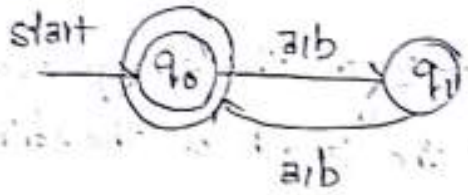
	a	b
→q_0	q_1	q_1
q_1	q_2	q_2
*q_2	q_ϕ	q_ϕ
q_ϕ	q_ϕ	q_ϕ

→ construct the DFA which accepts all the strings over an alphabet $\Sigma = \{a, b\}$ where the length of the string is even and odd.

$L = \{\epsilon, aa, ab, ba, bb, aaaa, aaab, \dots\}$

→ length of the string is = even and odd ⇒ Takes only 2 states.

Transition Diagram:- For even



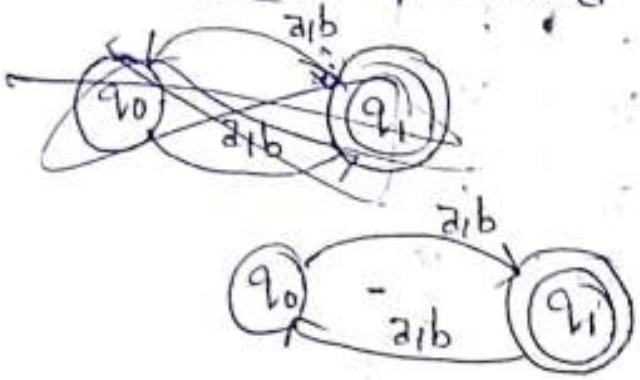
Even numbers [q_0 is final state]

Odd numbers [q_1 is final state]

Transition Table:-

	a	b
→* q_0	q_1	q_1
q_1	q_0	q_0

Transition Diagrams, For odd



Transistion Table:-

	a	b
→q_0	q_1	q_1
*q_1	q_0	q_0

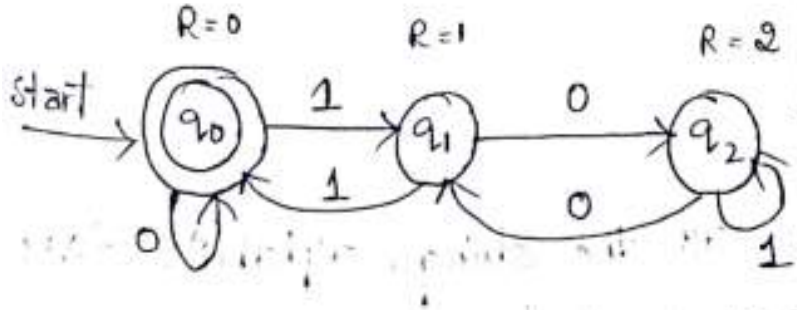
→ Construct DFA which accepts all the strings over an alphabet $\Sigma = \{0,1\}$ where the binary integers are divisible by 3.

↳ The lengths of the

$L = \{0, 11, 110, 1001, 1100, 1111, \ldots\}$

→ The length of string is depends on the "remainder of the number".

→ The length of the string is "3".



Transistion Table:-

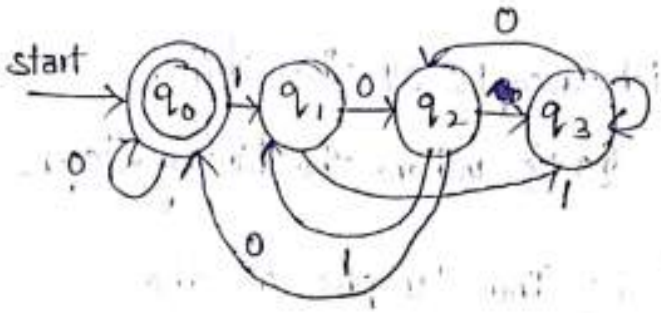
	0	1
→* q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_1	q_2

Decimal number	Binary equivalent	Rem ain der
0	0	0
1	1	1
2	10	2
3	11	0
4	100	1
5	101	2

→ Construct DFA which accepts all the strings over alphabet $\Sigma = \{0,1\}$ where binary integers, are divisible by "4".

$$L = \{\overset{0}{0}, \overset{4}{100}, \overset{8}{1000}, \overset{12}{1100}, \overset{16}{1110}, \ldots\}$$

remainder R = 0,1,2,3



Transition Table:

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

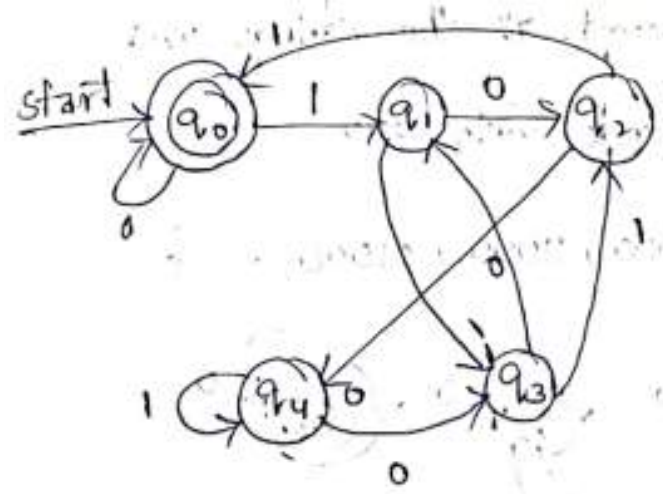
- Decimal binary remainder

0	0	0
1	1	1
2	10	2
3	11	3
4	100	0
5	101	1
6	110	2
7	111	3

→ Construct DFA all the strings alphabet $\Sigma = \{0,1\}$ where are divisible by "5".

$$L = \{\overset{0}{0}, \overset{5}{101}, \overset{10}{1010}, \overset{15}{1111}, \ldots\}$$

R = 0,1,2,3,4



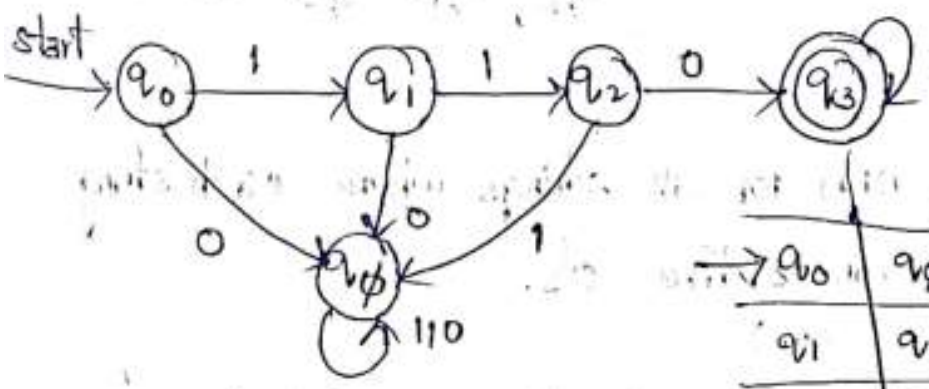
Decimal	binary	remainder
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	0
6	110	1
7	111	2
8	1000	3
9	1001	4

Transistion Table:-

	0	1
→ * q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_4	q_0
q_3	q_1	q_2
q_4	q_3	q_4

→ construct DFA which accepts all the strings over an alphabet $\Sigma = \{0,1\}$ starts with 110.

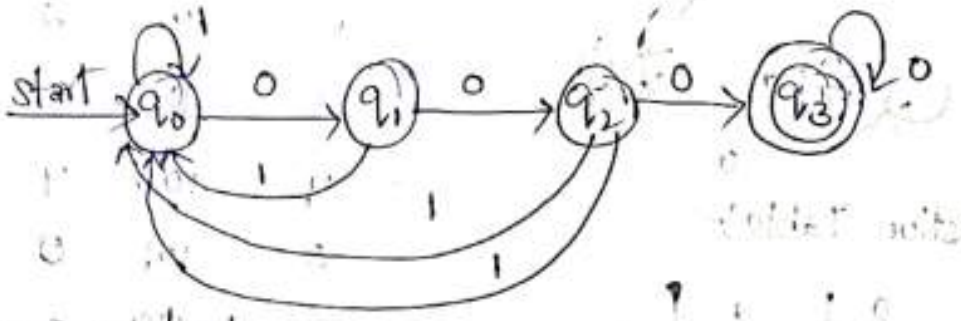
$L = \{110, 1101, 1100, 11000, 11001, 11010, \ldots\}$



	0	1
→ q_0	q_ϕ	q_1
q_1	q_ϕ	q_2
q_2	q_3	q_ϕ
q_3	q_3	q_3
q_ϕ	q_ϕ	q_ϕ

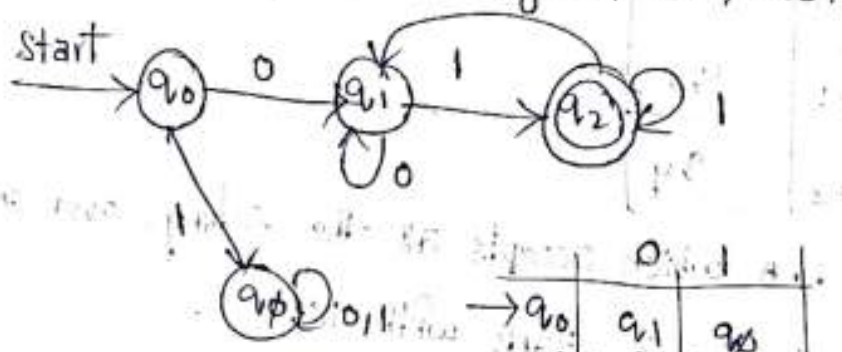
→ construct DFA which accepts all the strings over an alphabet $\Sigma = \{0,1\}$ ends with 000

$L = \{000, 1000, 0000, 11000, 01000, \ldots\}$



→ construct DFA $\Sigma = \{0,1\}$ starts with 0 and ends with 1.

$L = \{01, 001, 011, 0001, 0101, 0111, 0001, \ldots\}$



	0	1
→ q_0	q_1	q_ϕ
q_1	q_1	q_2
q_2	q_1	q_2

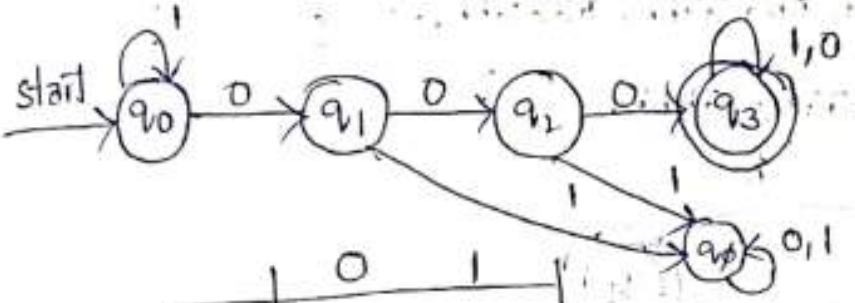
→ Construct DFA for all strings where each string has three consecutive 0's.

(or)

Construct DFA for all strings which have/ contains "000".

$L = \{000, 1000, 0001, 0010, 00010, 10001, \ldots\}$

length of the string = 3 then n+1 states = 4

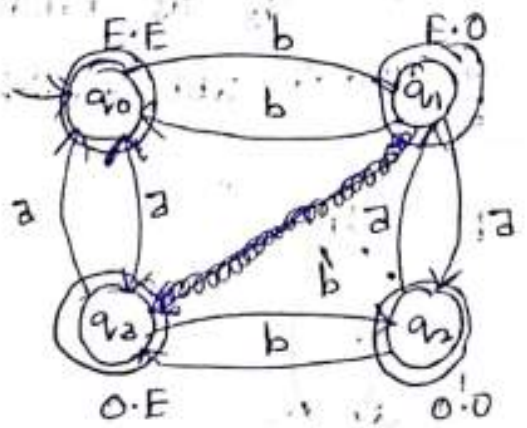


	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_2	q_ϕ
q_2	q_3	q_ϕ
$*q_3$	q_3	q_3

→ Construct DFA's for:

1) Even number of a's and even number of b's.
2) Even number of b's and odd number of a's.
3) Even number of a's and even number of b's.
4) odd number of a's and odd number of b's.

1) Even number of a's and odd number of b's.



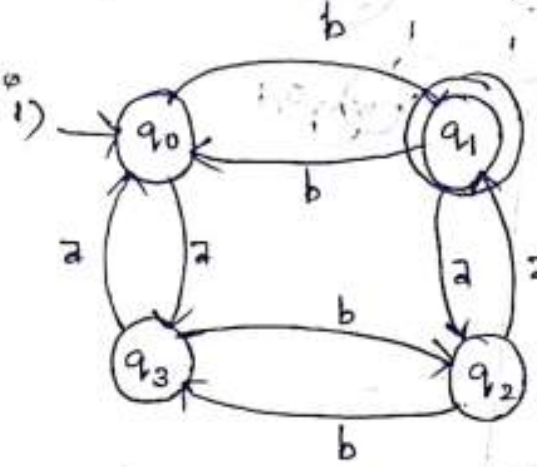
i) $L = \{b, bbb, abb, aabbb, \ldots\}$

ii) $L = \{a, aaa, abb, aaabb, \ldots\}$

iii) $L = \{\epsilon, aa, bb, aabb, aaaabb, \ldots\}$

iv) $L = \{ab, aaab, abbb, \ldots\}$

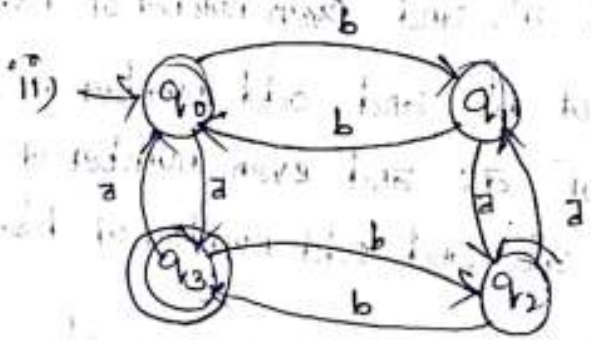
i)



here we take "q_1" as final state.

$L = \{b, bbb, aab, aabbb, \ldots\}$

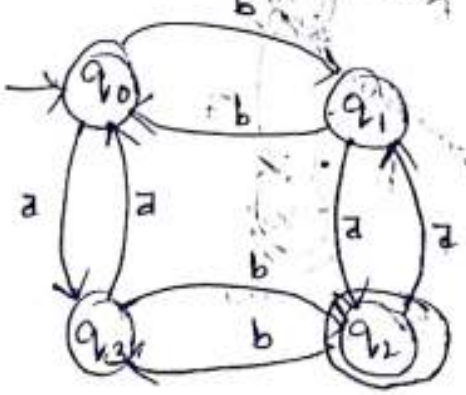
ii)



here we take "q_3" as final state

$L = \{a, aaa, abb, aaabb, \ldots\}$

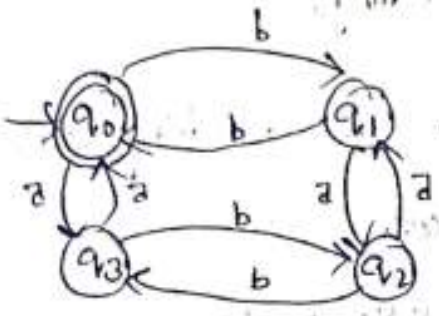
iii)



here we take "q_2" as final state

$L = \{\epsilon, aa, bb, aabb, aaaabb, \ldots\}$

iv)



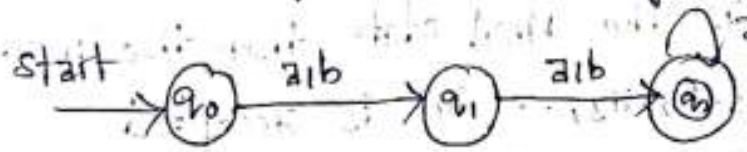
Here we take "q_0" as final state

$L = \{ab, aaab, abbb, \ldots\}$

⇒ Design a DFA that $L(M) = \{x / x$ is a string of 0s and 1s and $|x| \geq 2\}$

$L = \{aa, ab, ba, bb, aaa, aab, aba, abb \ldots\}$

→ The length of the string is 2 = 3 states



	a	b
→ q_0	q_1	q_1
q_1	q_2	q_2
q_2	q_2	q_2

How DFA process a string:

Let $a_1, a_2, a_3, \ldots, a_n$ is a string (or) some input sequences.

→ start DFA from initial state (q_0)

→ consult transistion from $\delta(q_0, a_1)$ to find state to which transistion takes place from q_0 on input symbol a_1.

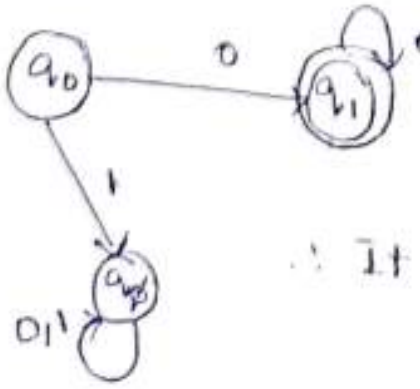
→ for the next input symbol evaluate $\delta(q_1, a_2)$ to find next state (q_2).

→ continue this process to find $q_3, q_4, \ldots q_n$

→ If q_n is the final state then the input sequence $a_1, a_2, a_3, \ldots, a_n$ is accepted otherwise it is rejected.

Example:-

0010010



∴ It satisfies the data.

Extended Transistion Function:-

An extended transistion function describes, what happens when we start from any state and follow some input sequence.

→ Extended Transistion function is denoted by $\hat{\delta}$.

→ Extended Transistion function we give state and input sequence.

$$\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$$

Where $w = xa$.

Language of DFA:-

The language of DFA is denoted by $L(A)$.

→ It is defined as

$$L(A) = \{ w / \hat{\delta}(q_0, w) \text{ is in } F \}.$$

→ If L is a language of DFA ($L(A)$) for some DFA "A" then L is called a "regular language".

Non-Deterministic finite Automata:- (NFA)

⇒ In which the finite automata can exists in any number of states on input from a state.

→ NFA is also a "5 tuple" notation. It consists of set of $(Q, \Sigma, \delta, q_0, F)$.

→ Q: set of states

Σ: set of input symbols

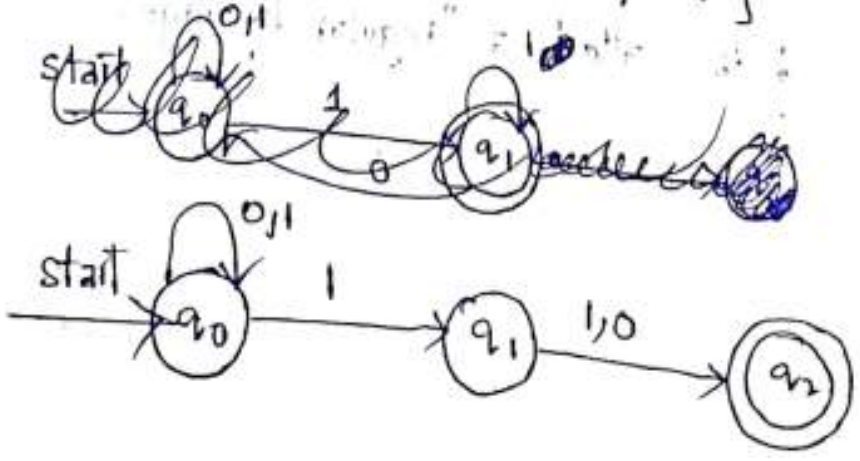
δ: A transistion function $\delta(q_0, 1)$, which takes two arguments "state and input symbol" and give one output.

q_0: starting state.

F: set of final states (or) Accepting states.

→ Construct NFA which accepts all strings whose second last symbol is "1".

$L = \{10, 11, 010, 011, 111, 110, \ldots\}$



	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_1\}$
q_1	q_2	q_2
$*q_2$	ϕ	ϕ

Extended Transition Function:-

The Language of NFA:-

It is also denoted by $L(A)$ and it is defined as $L(A) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \phi\}$

Extended Transistion function:-

$$\hat{\delta}(q_0, w) = \{r_1, r_2, \ldots r_n\}$$

Where, $r_1, r_2, \ldots, r_n$ are final states.

Finite Automata with epsilan Transistions:-

→ It is represented as "NFA-ϵ" (or) "ϵ-NFA".

→ It is a "5 tuple" notation. where

$A = (Q, \Sigma, \delta, q_0, F)$ transistion.

Q = set of states

$\Sigma = \Sigma \cup \{\epsilon\}$

δ = A transition function. $\delta(q_0, 1)$, which takes two arguments "state and input symbol" and give one output.

q_0 = starting state

F = set of Final states/Accepting states.

Usec of Epsilan:-

1) Each epsilan is along the path is "Invisible", i.e, it contributes nothing to string along the path.

→ Construct epsilan NFA which accepts decimal numbers consisting of i) an optional "+ or −"

ii) A string of digits

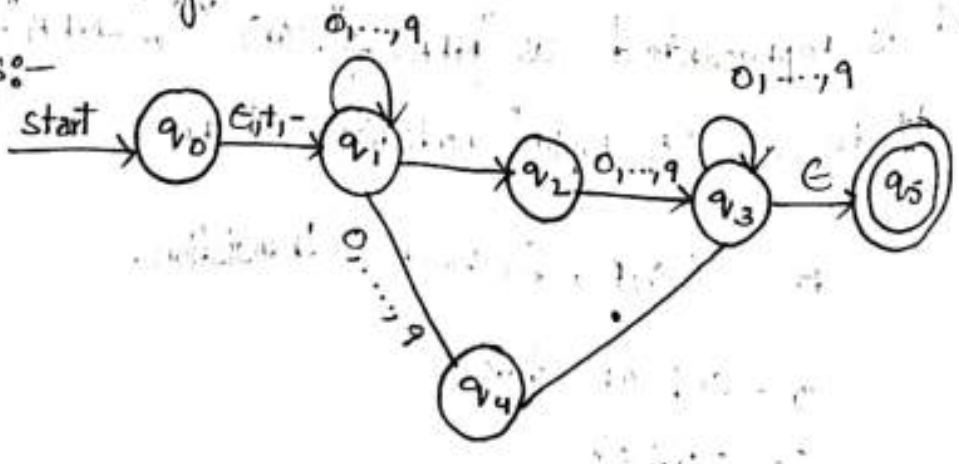
iii) A decimal point

iv) Another string of digits.

v) Either these string of digits can be empty but

vi) Atleast one of string of digits must be non empty.

Ans:-



Extended Transition Function:-

$$\hat{\delta}(q_0, w) = \bigcup_{j=1}^{m} (\epsilon\text{-closure}(r_j))$$

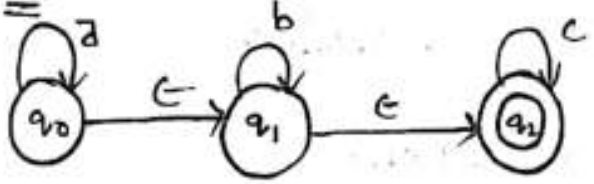
$$= \epsilon\text{-closure}(r_1) \cup \epsilon(\text{closure}(r_2) \cup \ldots$$

$$\cup \epsilon\text{-closure}(m)$$

Epsilan Closure:-

ϵ-closure of a state Q consists of every state that can be reached from Q along a path with "ϵ".

Example:-



ϵ

ϵ – closure (q_0) :- There is an "invisible transition to the "itself. " $\{q_0, q_1, q_2\}$ "

ϵ - closure (q_1) : " $\{q_1, q_2\}$ "

ϵ - closure (q_2) : " $\{q_2\}$ "

Eliminating ϵ - Transistions:-

Conversion of "NFA - ϵ" (or) "ϵ NFA" to "NFA without Transistions of ϵ":-

→ Find ϵ-closures for all states of the given ϵ-NFA.

→ Find Transistions($\hat{\delta}$) for each state / input symbol.

→ ~~construct~~ $\hat{\delta}(q, a) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q), a))$

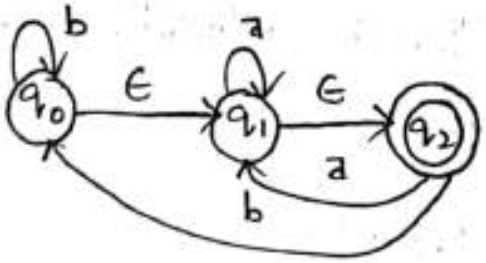
→ construct NFA without epsilan Transistions based on the computations of Transistion function.

→ Decide Final state as (F') as

i) $F' = F \cup \{q_0\}$ if ϵ-closure (q_0) is having F as a member.

ii) $F' = F$, otherwise

→



Step 1:- Find ϵ-closures for all states

ϵ-closure $(q_0) = \{q_0, q_1, q_2\}$

ϵ-closure $(q_1) = \{q_1, q_2\}$

ϵ-closure $(q_2) = \{q_2\}$

Step 2:- find the Transistions for each state on input symbol.

$\Rightarrow \hat{\delta}(q_1, a) = \epsilon\left(\text{closure}\left(\delta\left(\epsilon\text{-closure}(q_0), a\right)\right)\right)$

$= \epsilon\left(\text{closure}\left(\delta((q_0, q_1, q_2), a)\right)\right)$

$= \epsilon\left(\text{closure}\left(\delta((q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a)\right)\right)$

$= \epsilon\left(\text{closure}\left(\phi \cup q_1 \cup q_1\right)\right)$

$= \epsilon\left(\text{closure}(q_1)\right)$

$= \{q_1, q_2\}$

Step 3:-

$\Rightarrow \hat{\delta}(q_1, b) = \epsilon\left(\text{closure}\left(\delta\left(\epsilon\text{-closure}(q_0), b\right)\right)\right)$

$= \epsilon\left(\text{closure}\left(\delta((q_0, q_1, q_2), b)\right)\right)$

$= \epsilon\left(\text{closure}\left(\delta((q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b)\right)\right)$

$= \epsilon\left(\text{closure}(q_0 \cup \phi \cup q_0)\right)$

$= \epsilon\left(\text{closure}(q_0)\right)$

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

$\Rightarrow \hat{\delta}(q_1, a) = \epsilon\left(\text{closure}\left(\delta\left(\epsilon\text{-closure}(q_1), a\right)\right)\right)$

$= \epsilon\left(\text{closure}\left(\delta((q_1, q_2), a)\right)\right)$

$= \epsilon\left(\text{closure}\left(\delta(q_1, a) \cup \delta(q_2, a)\right)\right)$

$= \epsilon\left(\text{closure}(q_1 \cup q_1)\right)$

$= \epsilon(\text{closure}(q_1))$

$= \{q_1, q_2\}$

$\Rightarrow \hat{\delta}(q_1, b) = \epsilon(\text{closure}(\delta(\epsilon\text{-closure}(q_1), b))$

$= \epsilon(\text{closure}(\delta(q_1, q_2), b))$

$= \epsilon(\text{closure}(\delta(q_1, b) \cup \delta(q_2, b)))$

$= \epsilon(\text{closure}(\phi \cup q_0))$

$= \epsilon(\text{closure}(q_0))$

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

$\Rightarrow \hat{\delta}(q_2, a) = \epsilon(\text{closure}(\delta(\epsilon\text{-closure}(q_2), a))$

$= \epsilon(\text{closure}(\delta(q_2), a))$

$= \epsilon(\text{closure}(\delta(q_2, a)))$

$= \epsilon(\text{closure}(q_1))$

$= \{q_1, q_2\}$

$\Rightarrow \hat{\delta}(q_2, b) = \epsilon(\text{closure}(\delta(\epsilon\text{-closure}(q_2), b))$

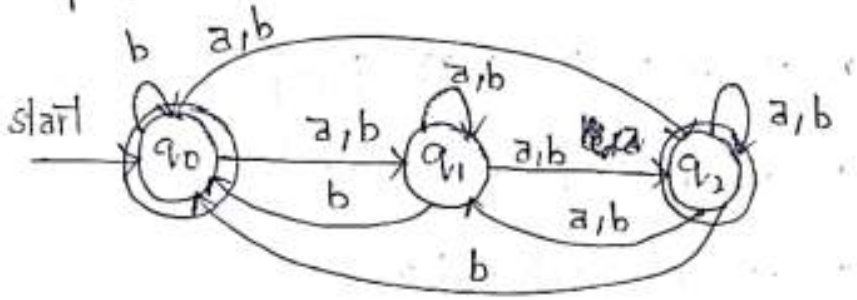
$= \epsilon(\text{closure}(\delta(q_2, b))$

$= \epsilon(\text{closure}(\delta(q_2, b))$

$= \epsilon(\text{closure}(q_0))$

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

Step 3:-



Step 4:-

$F' = F \cup \{q_0\}$ if ϵ-closure(q_0) is having F as a member.

Otherwise $F' = F$.

$F = \{q_2\}$

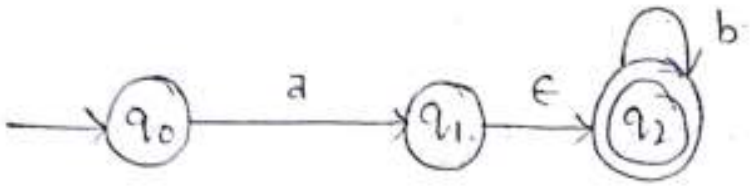
ϵ-closure$(q_0) = \{q_0, q_1, q_2\}$

$F' = F \cup \{q_0\}$

$= q_2 \cup \{q_0\}$

$= \{q_2, q_0\}$

→ convert following ε-NFA - ε to the NFA without ε.



Ans:-

Step 1:- Find ε closures for all states

$$\epsilon\text{-closure}(q_0) = \{q_0\}$$

$$\epsilon\text{-closure}(q_1) = \{q_1, q_2\}$$

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

Step 2:- Find the Transitions for each state on input symbol.

$$\Rightarrow \hat{\delta}(q_0, a) = \epsilon\text{-(closure}(\delta(\epsilon\text{-closure}(q_0), a))$$

$$= \epsilon\text{-(closure}(\delta(\{q_0\}, a)))$$

$$= \epsilon\text{-(closure}(q_1))$$

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

$$\Rightarrow \hat{\delta}(q_0, b) = \epsilon\text{(closure}(\delta(\epsilon\text{-closure}(q_0), b))$$

$$= \epsilon\text{-(closure}(\delta(q_0, b)))$$

$$= \epsilon\text{-closure}(\phi)$$

$$= \phi$$

$\Rightarrow \hat{\delta}(q_1, a) = \epsilon(\text{closure}(\delta(\epsilon\text{-closure}(q_1), a))$

$= \epsilon\text{-}(\text{closure}(\delta(\epsilon\text{-closure}(q_1, a))))$

$= \phi$

$\Rightarrow \hat{\delta}(q_1, a) = \epsilon(\text{closure}(\delta(\epsilon\text{-closure}(q_1), b))$

$= \epsilon\text{-}(\text{closure}(\delta(\epsilon\text{-closure}(q_1, b))))$

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

$= q_2 \phi q_2$

$\Rightarrow \hat{\delta}(q_2, a) = \epsilon(\text{closure}(\delta(\epsilon\text{-closure}(q_2), a))$

$= \epsilon(\text{closure}(\delta(\epsilon\text{-closure}(q_2, a))))$

$= \phi\ q_2\ \phi$

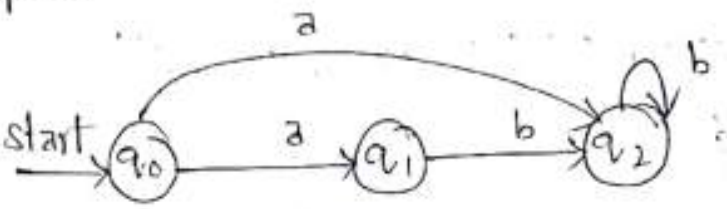
$\Rightarrow \hat{\delta}(q_2, b) = \epsilon(\text{closure}(\delta(\epsilon\text{-closure}(q_2), b))$

$= \epsilon(\text{closure}(\delta(\epsilon\text{-closure}(q_2, b))))$

$= \epsilon(\text{closure}(q_2))$

$= q_2\ \phi\ q_2$

Step 3:-



Step 4:-

$F' = F \cup \{q_0\}$ if ϵ-closure (q_0) is having F as a member

Otherwise $F' = F$

$F = \{q_2\}$ ϵ-closure $(q_0) = q_0$

Since F is not a member of ϵ-closure (q_0) so, $F' = F$

$F' = \{q_2\}$.

Conversion of NFA to DFA:-

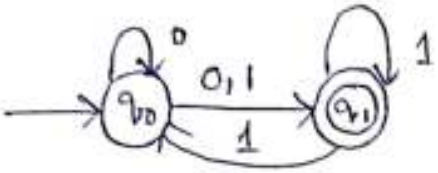
Step 1: start with initial state and find its transitions.

Step 2: If there are any new states find transitions for each of them.

Step3:- Continue step 2 until there are no new transistions.

Step4:- Mark all the states which has the final state of NFA as final state of DFA.

1) convert the following NFA into DFA / Find the equivalent DFA for the given NFA.



Step 1:- Find the transistions which starts with initial state.

	0	1
→ q_0	$[q_0, q_1]$	q_1
$[q_0, q_1]$	$[q_0, q_1]$	q_1
q_1	ϕ	$[q_0, q_1]$

Step 2:- Find the transistions for the new states occured

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

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

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

$= [q_0 q_1] \cup \phi$

$= \{q_0, q_1\}$

$\delta([q_0 q_1], 1) = \delta(q_0, 1) = \delta(q_0, 1) \cup \delta(q_1, 1)$

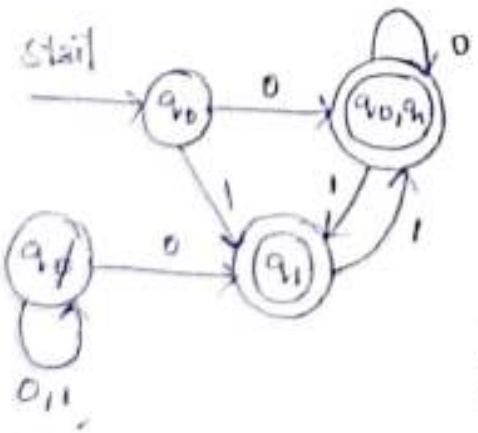
$= q_1 \cup q_1$

$= q_1$

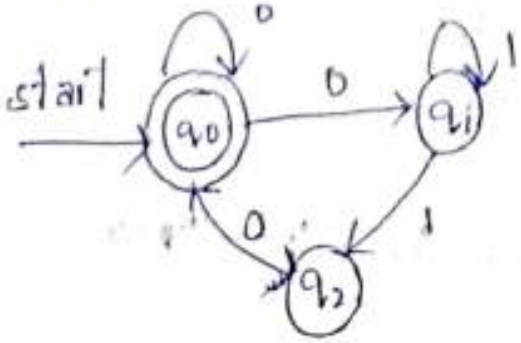
$\delta(q_1, 0) = \phi$

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

Step 3:- Continue the previous step if there are no new states, after that we need draw the transition diagram from the table.



2) Determine the equivalent DFA to the given NFA.



Step 1:-

	0	1
$\rightarrow q_0$	$[q_0 q_1]$	ϕ
$[q_0 q_1]$	$[q_0 q_1]$	$[q_1 q_2]$
$[q_1 q_2]$	q_0	$[q_1 q_2]$
q_ϕ	q_ϕ	q_ϕ

Step 2:-

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

$\delta(q_0, 1) = \phi$

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

$= q_0 \cup \phi \; q_1 \; [q_0 q_1] \cup \phi$

$= q_0 \; \{q_0, q_1\}$

$\delta([q_0 q_1], 1) = \delta(q_0, 1) \cup \delta(q_1, 1)$

$= \phi \cup [q_1, q_2]$

$= [q_1 q_2]$

$\delta(q_1, q_2)$ =

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

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

$\delta(q_2, 0) = q_0$

$\delta(q_2, 1) = \phi$

$\delta((q_1, q_2), 0) = \delta(q_1, 0) \cup \delta(q_2, 0)$

$= (\phi \cup \phi) \cup q_0$

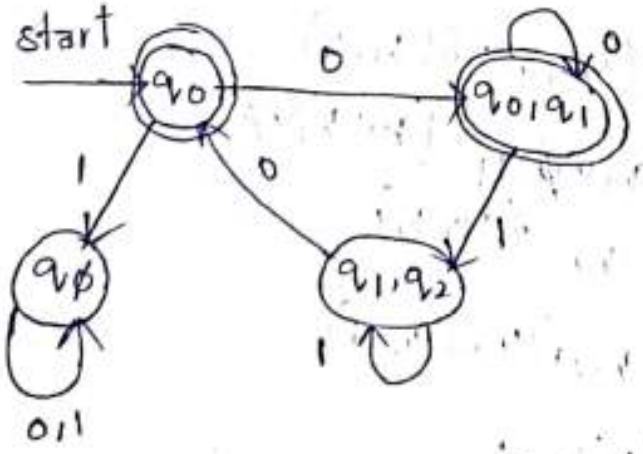
$= q_0$

$\delta((q_1, q_2), 1) = \delta(q_1, 1) \cup \delta(q_2, 1)$

$= (q_1, q_2) \cup \phi$

$= q_1, q_2$

step 3:-



Conversion of ϵ-NFA to DFA:-

Step1: Take epsilon-closure for the starting state of NFA as the starting state of DFA

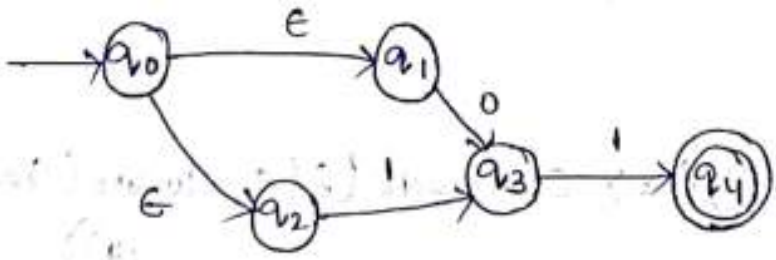
Step2: Find the states for each input from the current state.

Step3: If we find a new state, take it as a current state repeat step2.

Step4: & repeat step 2 and step 3 until there is no new state.

Step5: Mark the states of DFA as final states which contains the final state of NFA.

1) Convert the given ϵ-NFA into DFA



Step1: finding the ϵ-closures for the starting state of NFA as the starting state of DFA

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

	0	1
$\rightarrow[q_0, q_1, q_2]$	$\{q_3\}$	$\{q_3\}$
q_3	ϕ	q_4
q_4	ϕ	ϕ
q_ϕ	ϕ	ϕ

$\hat{\delta}([q_0, q_1, q_2], 0) \Rightarrow \epsilon$- closure $(\delta(\epsilon\text{-closure}(q_0, q_1, q_2), 0))$

$\Rightarrow \epsilon$- closure $(\delta(\epsilon$-closure $q_0 q_1 q_2), 0))$

$\Rightarrow \epsilon$ - closure $(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0))$

$\Rightarrow \epsilon$ - closure $(\phi \cup q_3 \cup \phi)$

$\Rightarrow \epsilon$ - closure (q_3)

$\Rightarrow q_3$

$\hat{\delta}([q_0, q_1, q_2], 1) \Rightarrow \epsilon$- closure $(\delta(\epsilon$- closure $(q_0, q_1, q_2, 1))$

$\Rightarrow \epsilon$ - closure $(\delta(q_0 q_1 q_2), 1))$

$\Rightarrow \epsilon$ - closure $(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1))$

$\Rightarrow \epsilon$ - closure $(\phi \cup \phi \cup q_3)$

$\Rightarrow \epsilon$ - closure (q_3)

$\Rightarrow q_3$

$\hat{\delta}([q_3],0) \Rightarrow \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_3),0))$

$\Rightarrow \epsilon\text{-closure}(\delta(q_3),0))$

$\Rightarrow \epsilon\text{-closure}\delta(q_3,0)$

$\Rightarrow \phi$

$\hat{\delta}([q_3],1) \Rightarrow \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_3),1))$

$\Rightarrow \epsilon\text{-closure}(\delta(q_3),1))$

$\Rightarrow \epsilon\text{-closure}\delta(q_3,1)$

$\Rightarrow q_4$

$\hat{\delta}([q_4],0) \Rightarrow \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_4),0))$

$\Rightarrow \epsilon\text{-closure}(\delta(q_4),0))$

$\Rightarrow \epsilon\text{-closure}(\delta(q_4,0))$

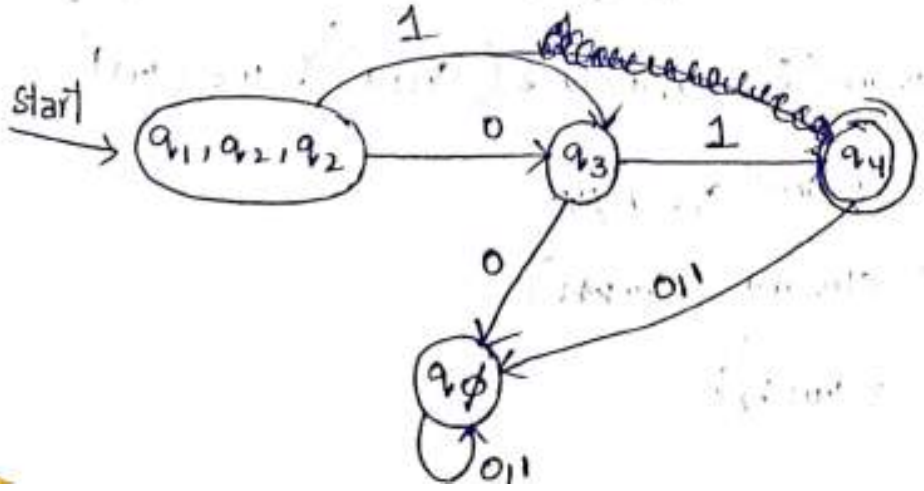
$\Rightarrow \phi$

$\hat{\delta}[q_4,1] \Rightarrow \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_4),1)$

$\Rightarrow \epsilon\text{-closure}(\delta(q_4),1))$

$\Rightarrow \epsilon\text{-closure}(\delta(q_4,1))$

$\Rightarrow \phi$



→ Determine the equivalent DFA for the given ∈-NFA.

i) ∈ - closure $(q_0) = \{q_0, q_1\}$

ii) ∈ - closure $(q_1) = \{q_1, q_2\}$

iii) ∈ - closure $(q_2) = \{q_2\}$

	a	b	c
→ $\{q_0, q_1\}^{q_2}$	$\{q_0, q_1\}^{q_2}$	$\{q_1, q_2\}$	$\{q_2\}$
$\{q_1, q_2\}$	ϕ	$\{q_1, q_2\}$	

$\hat{\delta}([q_0 q_1], a) \Rightarrow$ ∈-closure $(\delta($∈-closure $(q_0 q_1), a))$

$\Rightarrow$ ∈- closure $(\delta(q_0 q_1), a))$

$\Rightarrow$ ∈- closure $(\delta[q_0, a) \cup \delta(q_1, a)\} \cup \delta(q_2, a))$

$\Rightarrow$ ∈- closure $(q_0 \cup \phi)$

$\Rightarrow$ ∈- closure $(q_0 q_1)$

$\Rightarrow \{q_0, q_1\}^{q_2}$

$\hat{\delta}([q_0 q_1], b) \Rightarrow \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0 q_1), b))$

$\Rightarrow \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b)$

$\Rightarrow \epsilon\text{-closure}(\phi \cup q_1 \cup \phi)$

$\Rightarrow \epsilon\text{-closure}(q_1)$

$\Rightarrow \{q_1, q_2\}$

$\hat{\delta}([q_0 q_1 q_2], c) \Rightarrow \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0, q_1, q_2), c))$

$\Rightarrow \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0, c) \cup \delta(q_1, c) \cup \delta(q_2, c)$

$\Rightarrow \epsilon\text{-closure}(\phi \cup \phi \cup q_2)$

$\Rightarrow \{q_2\}$

$\hat{\delta}([q_1, q_2], a) \Rightarrow \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1, q_2), a))$

$\Rightarrow \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1, a) \cup \delta(q_2, a))$

$\Rightarrow \epsilon\text{-closure}(\delta \phi \cup \phi)$

$\Rightarrow \phi$

$\hat{\delta}[(q_1, q_2), b] \Rightarrow \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1, q_2), b))$

$\Rightarrow \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1, b) \cup \delta(q_2, b))$

$\Rightarrow \epsilon\text{-closure}(q_1 \cup \phi)$

$\Rightarrow \{q_1, q_2\}$

NFA - Applications: Text search.

→ It is used for web search (www). and "finding information from text".

Finding strings in Text:-

1) inverted indices — maintain some set of words & occurances
   • It requires more memory to traveller to search indices

2) Finite Automata
   ↓
   It is NFA
   → Thats why it is used.

1) Take a transistion on an intial state upon each input symbol.

2) To accept the string $a_1, a_2, \ldots a_k$ Take transistion from q_0 to q_1 on a_1

1) Take Transistion from q_1 to q_2 on a_2 and so on ... Then q_k becomes the final state

→ This is not suitable for where documents require rapid change.

= UNIT - II =

Regular Expression: It is a simple expression to describe a language that the DFA accept.

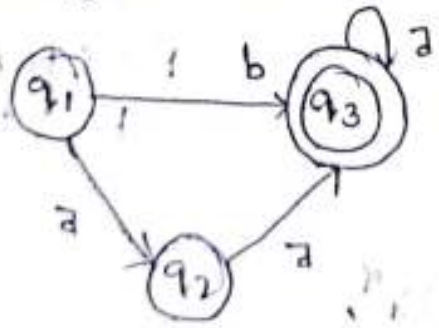
Finite Automata:-

Step 1: Take equation for each state based on incoming edges.

Step 2: Add ε to eqn of initial state.

Step 3: Simplify the eqns. using Arden's method and find regular expression for the final state.

1) Convert given DFA into regular expression.



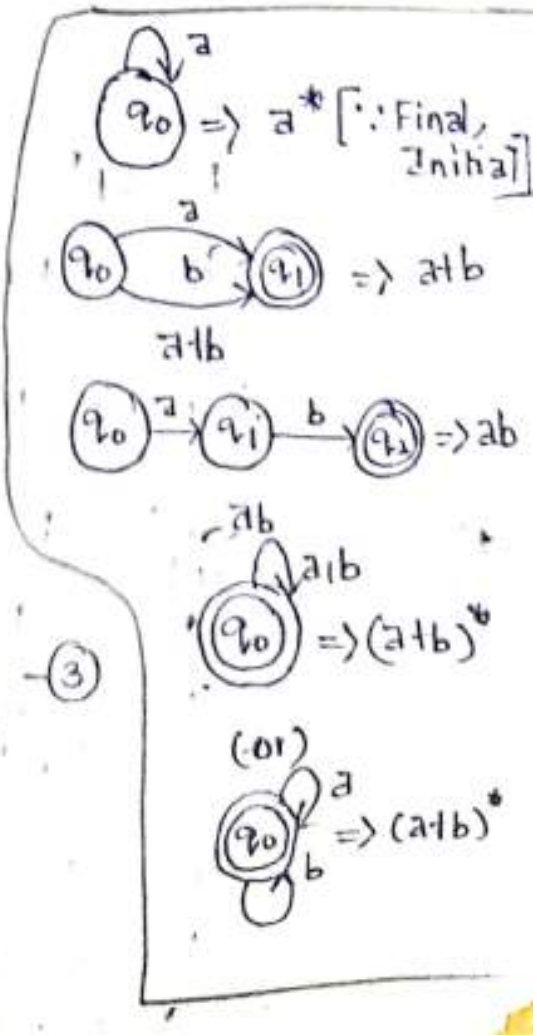
Step 1: Incoming edges

1) $q_1 = 0 \quad -(1)$

2) $q_2 = q_1 a \quad -(2)$

3) $q_3 = q_1 b + q_2 a + q_3 a \quad -(3)$

Step 2:- $q_1 = \varepsilon \quad -(1)$



Step 3:- $R = Q + RP \Rightarrow R = QP^*$

Subsitute eq ① in eq ②

$q_2 = q_1 a$

$= \varepsilon a$

$q_2 = a$ — ④

Subsitute eq ④ & ③ in eq ③

$q_3 = \varepsilon b + q_2 a + q_3 a$

$= \varepsilon b + aa + q_3 a$

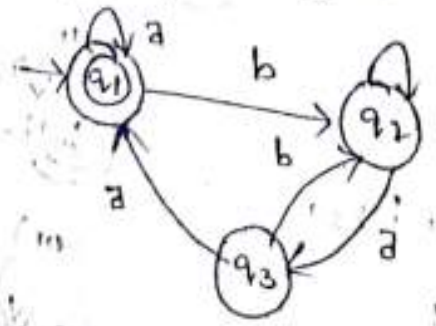
$\frac{q_3}{R} = \frac{b + aa}{Q} + \frac{q_3}{R} \cdot \frac{a}{P}$ [This is in the form of $R = Q + RP$, $R = Q + RP$]

$R = QP^*$

$q_3 = (b + aa)a^*$ //

∴ This is regular expression.

②



1) $q_1 = q_1 a + q_3 a$ — ①

2) $q_2 = q_2 b + q_1 b + q_3 b$ — ②

3) $q_3 = q_2 a$ — ③

step2: $q_1 = q_1 a + \varepsilon$

step3: $R = Q + RP$

Substitute eq ③ in eq ②.

$q_2 = q_2 b + q_1 b + q_3 b$

$q_2 = q_2 b + q_1 b + q_2 ab$

$q_2 = q_1 b + q_2 b + q_2 ab$

$q_2 = q_1 b + q_2 (b + ab)$

$[\because R = Q + RP]$

$\boxed{q_2 = (q_1 b)(b + ab)^*}$ ⇓ $\because R = QP^*$

Substitute ④ in ③

$q_3 = q_2 a$

$q_3 = (q_1 b)(b + ab)^* a \quad -⑤$

Substitute eq ⑤ in eq ①

$q_1 = q_1 a + q_3 a + \varepsilon$

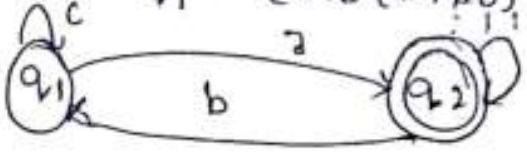
$q_1 = q_1 a + (q_1 b)(b + ab) a + \varepsilon$

$q_1 = \varepsilon + q_1 (a + b(b + ab)^* aa$

$q_1 = \varepsilon (a + b(b + ab)^* aa^*)$

$q_1 = (a + b(b + ab)^* aa)^*$

⑧



Sol:- Step 1: $q_1 = q_2 b + q_1 c$ — (1)

$q_2 = q_1 a + q_2 d$ — (2)

Step 2: $q_1 = q_2 b + q_1 c + \varepsilon$ — (3)

Step 3: $q_1 = (q_2 b + \varepsilon) + q_1 c$

$q_1 = (q_2 b + \varepsilon) c^*$ $\Rightarrow RP = R = QP^*$

subsitute eq (3) in eq (2)

$q_2 = q_2 d + q_1 a$

$q_2 = q_2 d + (q_2 b + \varepsilon) c^* a$

$q_2 = q_2 d + q_2 b c^* a + c^* a$

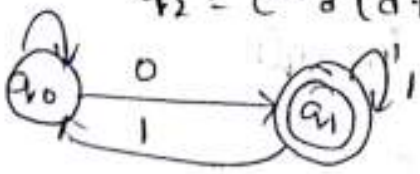
$q_2 = q_2 (d + b c^* a) + c^* a$

$\therefore [R = RP + Q]$

$\therefore$ Now write this in $R = QP^*$ form

$q_2 = c^* a (d + b c^* a)^*$

(4)



Step 1: $q_0 = q_0 1 + 0 + q_1 1 + 1$ — (1)

$q_1 = q_1 1 + q_0 0$ — (1)

Step 2: $q_0 = q_0 0 + q_1 1 + \varepsilon$

$R = RP + Q$

$q_0 = (q_1 1 + \epsilon) 0^*$ — ③

Substitute eq ③ in eq ②

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

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

$q_1 = q_1 (1 + 1\, 0^* 0) + 0^* 0$

$= q_1 = (0^* 0)(1 + 10^* 0)^*$

DFA to regular expression by state elemination method:-

Step 1:- Initial state should not have incoming edges, if exists create a new state make it as initial state.

Step 2:- Finite automata should have only one final state, if there are multiple final states then create a new state and make it as final.

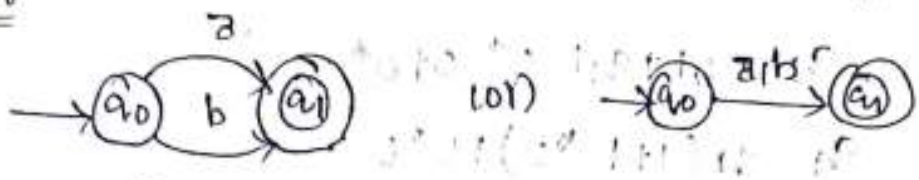
Step 3:- Final state should not have any outgoing edges if exists create a new state and make it as final state.

Step 4:- Eliminate every state except initial and final states.

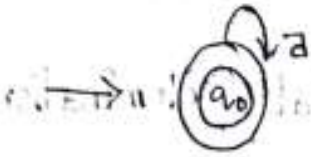
Step 5: Finally FA should have only intial state and final states.

Step 6: The label from initial state to final state becoming the regular expression.

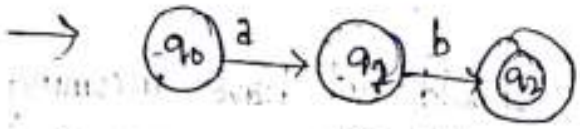
Q:-



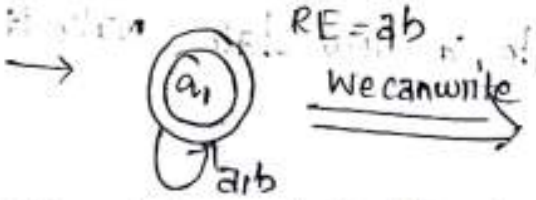
↳ RE = a+b



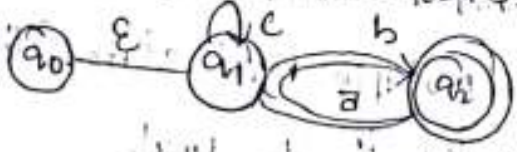
$\Rightarrow RE = a^*$



RE = ab

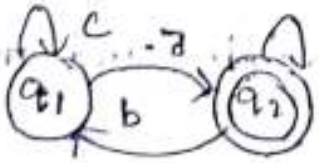


$RE = (a+b)^*$



$\varepsilon \cdot (c^*(b+ab)) = c^*b + c^*ab$

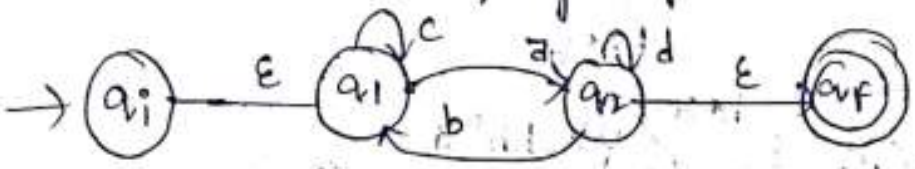
(2)



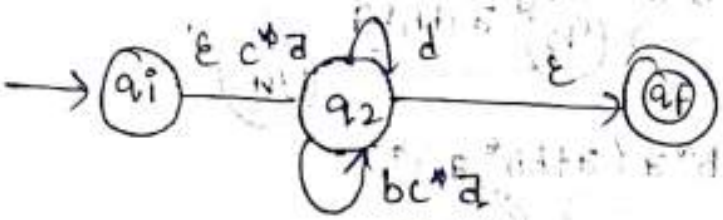
Since there is an incoming edge for q_1 take a new state as initial state.

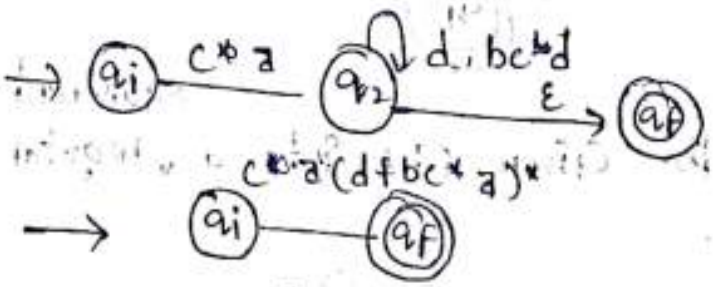
step3:-

since there are outgoing edges from



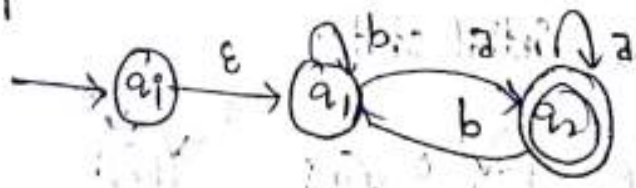
eliminate q_1 then DFA



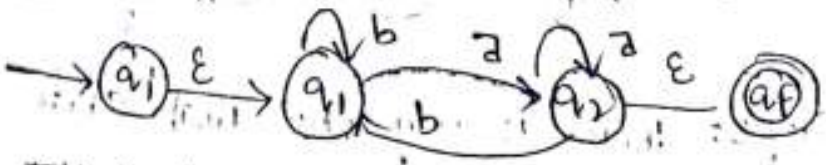


=> Write the regular expression:

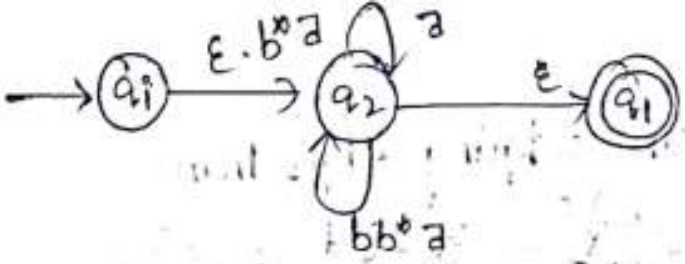
step1:

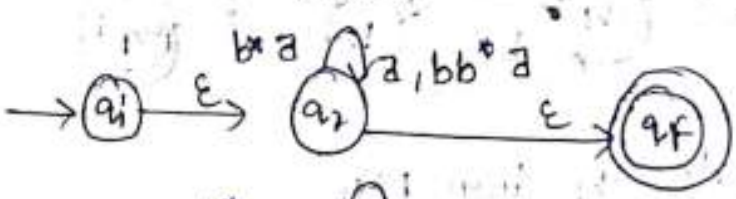


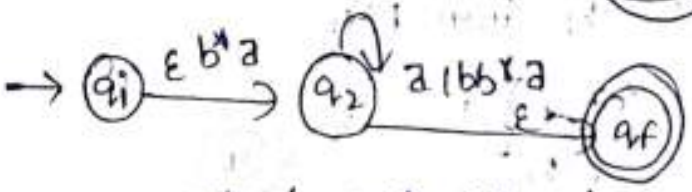
step2:

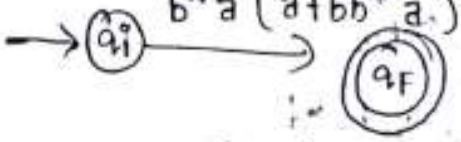


Step3: Eliminate the 'q1' state.

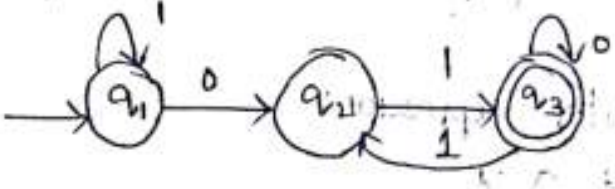




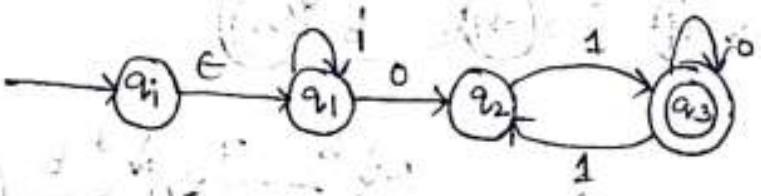




⇒ Convert the Given DFA into a equivalent regular expression.

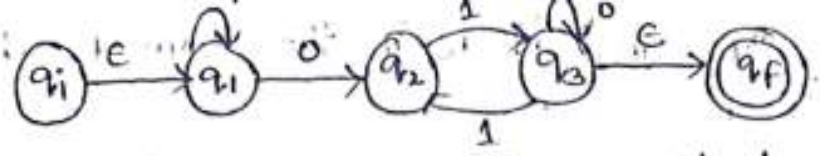


Step1: Since there is an incoming edge to intial, so create a new state (q_i) and make it as intial state.

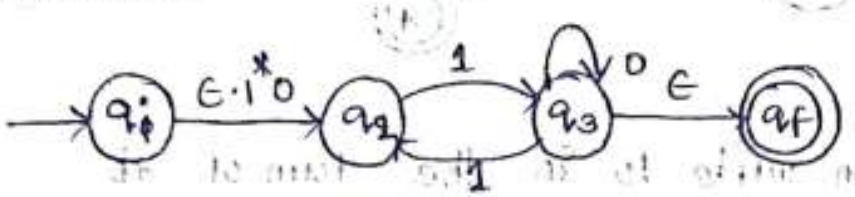


Step2: There is only one final state.

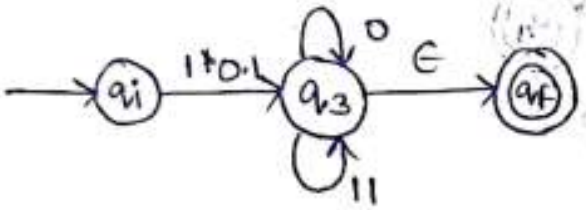
Step3: Since there are outgoing edges to final state, create a new state and make it as final state.

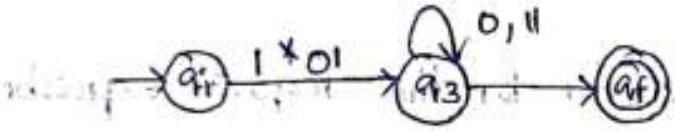


Step4: Eliminate q_1 then the resultant DFA becomes

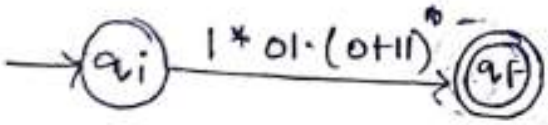


Eliminate q_2 then the resultant DFA becomes





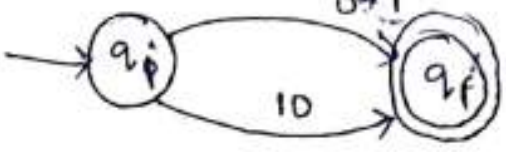
Eliminate q_3 then the resultant DFA becomes



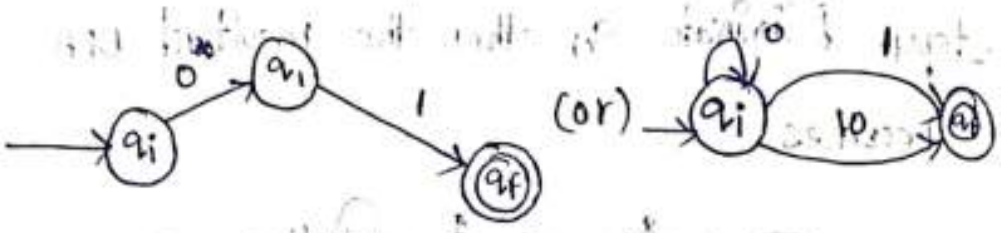
Regular Expression (RE) = $1^*01(0+11)^*$

⇒ convert given regular expression into its equivalent finite Automato.

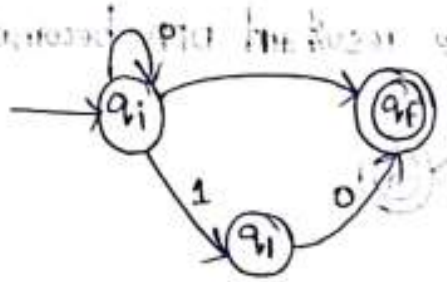
$$\underbrace{0^*1}_{a} + \underbrace{10}_{b}$$



a is $\underbrace{0^*}_{a}\underbrace{1}_{b}$ and it is in the form of ab

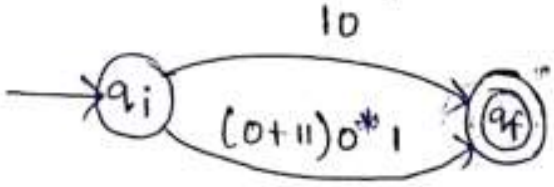


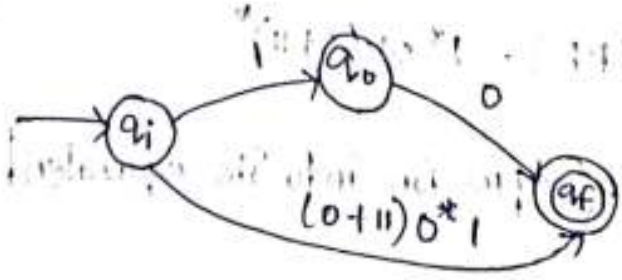
We can write 10 in the form of ab

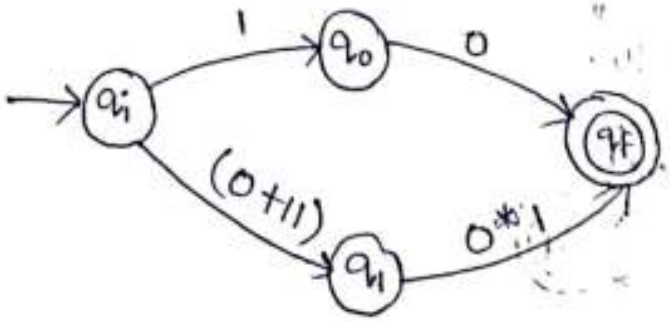


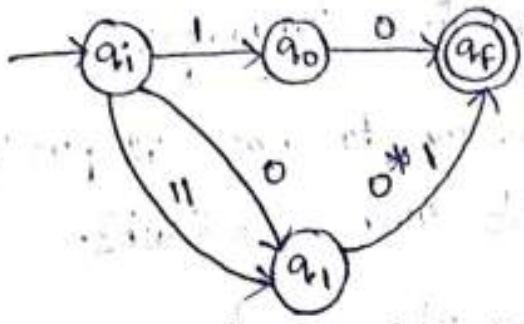
⇒ Find Finite Automata for the regular expression

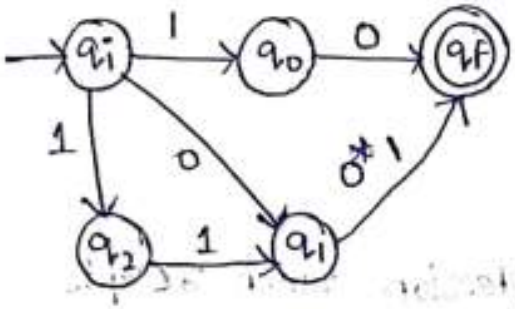
$$\underbrace{10}_{a} + \underbrace{(0+11)0^*1}_{b}$$

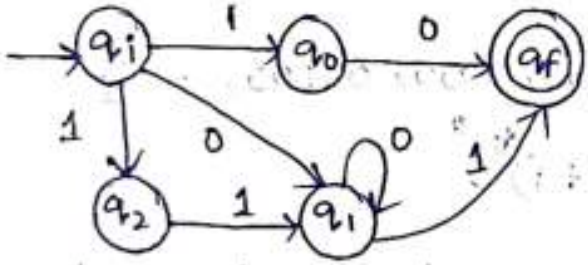






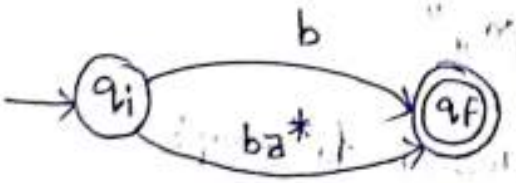


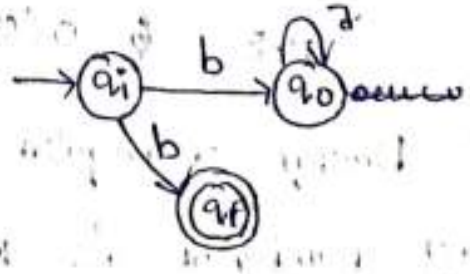


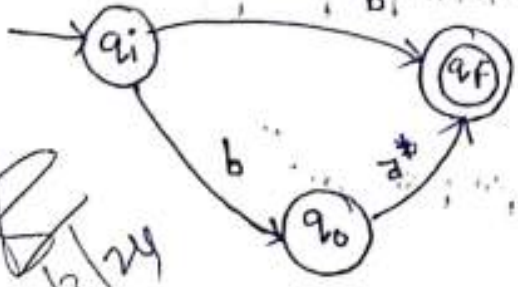


⇒ Find finite Automato for the regular expression

$$\frac{b}{a} + \frac{ba^*}{b}$$







b|3|24

Regular Expression for the Given languages:-

i) Design the regular expression to access all possible combinations of a's and b's over $\Sigma = \{a, b\}$.

Sol:- $L = \{\epsilon, a, b, ab, aab, baa, aba, \ldots\}$

"$R.E = (a+b)^*$"

Transition diagram:-

ii) Design a regular expression which accepts all strings which ends with "00".

Sol:- $L = \{00, 100, 1100, 0100, 01100, \ldots\}$

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

iii) Design a regular expression for to accepts all strings which starts with "0" and ends with "1"

Sol:- $L = \{0101, 01, 011, 01011, \ldots\}$

"$R.E = 0(0+1)^*1$"

iv) Design a regular expression to accepts any number of a's following by any no. of b's and any no. of c's.

Sol:- "$R.E = (a^* b^* c^*)$"

⇒ v) atleast one b's, one a's, one c's

" R.E = $(aa^* bb^* cc^*)[or](a^+ b^+ c^+)$ "

vi) begins with "oo (or) 11" and ends with "oo(or)11"

sol:- " R.E = $(oo+11)(o+1)^* (oo+11)$ "

vii) third character from the right end is "a"

sol:- " R.E = $(a+b)^* a (a+b)(a+b)$ "

viii) design R.E for atleast 2 b's.

Sol:- " RE = $(a+b)^* b (a+b)^* b (a+b)^*$ "

ix) " exactly two "b's.

R.E = $a^* b a^* b a^*$

x) Even length strings over $\Sigma = (0)$

" R.E = $(oo)^*$ "

$L = \{\epsilon, oo, oooo, oooooo, \ldots\}$

xi) odd length strings over $\Sigma = (1)$

" R.E = $1(11)^*$

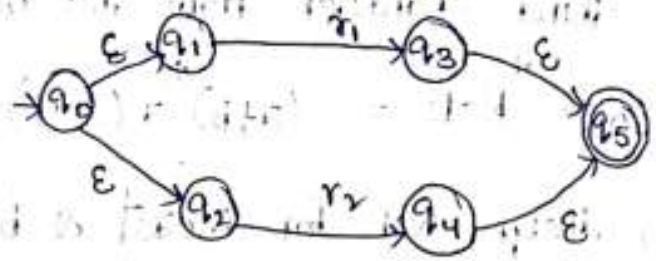
$L = \{\epsilon, 111, 1, 11111, 1111111, \ldots\}$

Regular Expression to Finite Automata using Subset method:

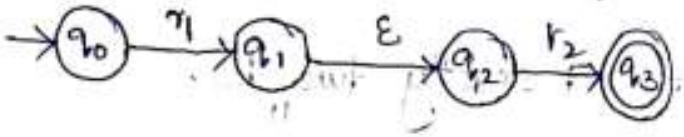
Step1: Convert the given expression into NFA with ε

Step2: Convert NFA with ε into DFA.

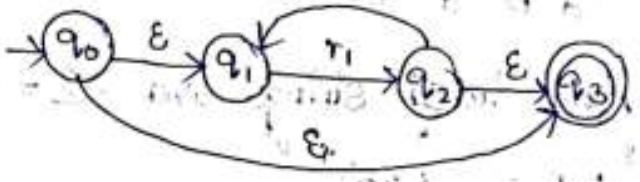
⇒ $r_1 + r_2$ →



⇒ $r_1 r_2$ →



⇒ r_1^* →



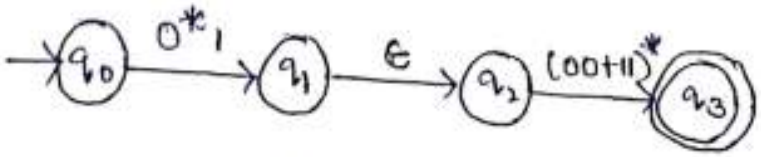
Q:- Convert the Given expression into the finite Automata using "subset" method.

$$\underbrace{0^*1}_{r_1}\underbrace{(00+11)^*}_{r_2}$$

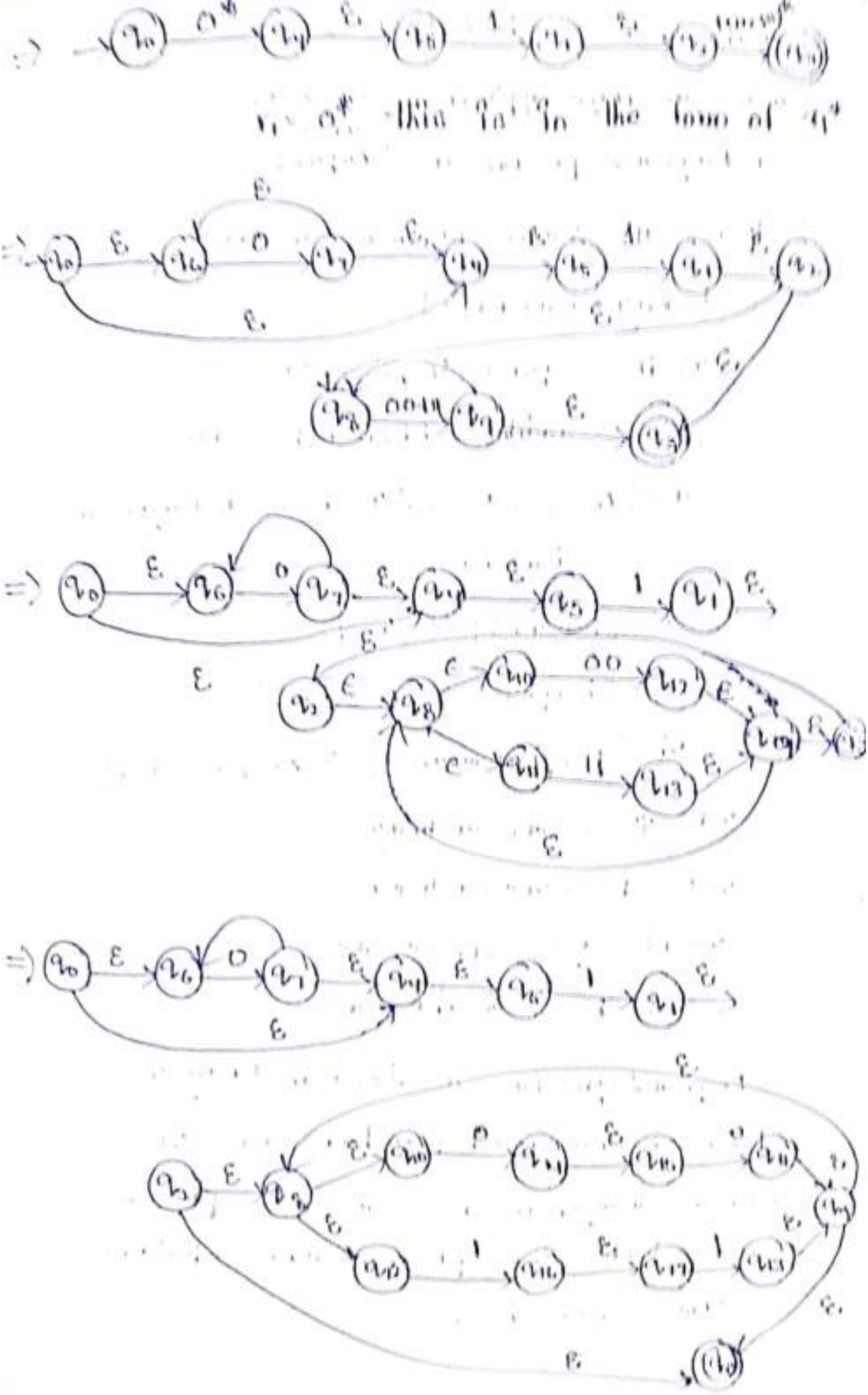
A:- This is in the form of "$r_1 r_2$"

where $r_1 = 0^*1$, $r_2 = (00+11)^*$

⇒



$r_1 = 0^*1$ Again this is in the form of $r_1 r_2 \Rightarrow r_1 = 0^*$, $r_2 = 1$



Applications of Regular Expressions:-

=> There are three applications of R.E

1) Regular Expressions in Unix:-

→ There are various notations in unix

i) Any character [.]

ii) The sequence [1,2,.....10]

=> addition - 1+2+3+4+.....+10

iii) [A-Z], [a-z], [0-9] → A range of character

iv) [:digit:] → [0-9]

v) [:alpha:] → [A-Z, a-z]

vi) [:alphanumeric:] → [A-Z, a-z, 0-9]

vii) * → zero (or) many

viii) + → one (or) many

ix) ? → zero (or) one

x) | → concatenation

2) Regular Expressions in Lexical Analyser:-

→ One of the most important phase of Lexical analyser is to scan the program and divide the programs into individual units called tokens.

→ semantic analyser provides some tasks to the token.

→ First phase of the compiler it pre divides the program into tokens.

→ In tokens, there are identifier and keyword

→ In unix, there are lex and flex they decide what how the lexical analyser do in unix operating system.

8) Regular Expressions in finding patterns in the string:

→ To find the patterns in the string we use RE.

123A Main st:

[0-9]+[A-Z]?[A-Z][a-z]*([A-Z][a-z]*)*

(Street|st\.|Road|Rd\.):

Algebraic laws of Regular Expression:

Associative laws for regular Expression RegEx

$A+(B+C) = (A+B)+C$ and $A(B.C) = (A.B)C$

Commutative laws for Regular Expression RegEx

$A+B = B+A$ however $A \cdot B \neq B \cdot A$

Identity for Regular Expression: RegEx

$\phi + A = A + \phi = A$

$\epsilon \cdot A = A \cdot \epsilon = A$

Distributivity for Regular Expression

$A(B+C) = AB + AC$ [left]

$(B+C)A = BA + CA$ [right]

→ ϕ is an anclilator

Idempotent $A + A = A$

closure laws for RegularExpression: RegEx

$(A^*)^* = A^*$, $\phi^* = \epsilon$, $\epsilon^* = \epsilon$, $A+ = AA^* = A^*A$, and

$A^* = A + \epsilon$.

Demorgon's Law:

$(L+B)^* = (L^*B^*)^*$.

Test for the Regular Expression whether $E = F$

1) convert the E and F into concreate R.E, C & D
2) Test $L(C) = L(D)$ th. If Language of C is equals to the Language of D then we can say $E = F$ is TRUE otherwise FALSE.

pumping lemma:- → To prove the language is regular expression or not.

→ repetion [Pumping] lemma[word] → It is entry point, using that we can ~~create~~ derive other words.

→ Repetition of that word, by that word can derive the otherwords.

→ There are two types of pumping lemma:

1) Regular Expression for pumping lemma
2) context free languages

Theorem:-

let "L" be the Regular language, Then there exists a constant "c" such that for each and every string "w" in "L".

$|w| \geq c$

We can break the w in three strings; 'x, y, z'

- $|y| > 0$
- $|xy| \leq c$
- For all $k \geq 0$, the string xy^kz is also in L.

procedure to prove a language is not regular using pumping lemmas:

1) consider L is a regular language

2) select constant 'c' such that $|w| \geq c$, for every string (w) in L.

3) Take a string (w) from L and divide w into x, y and z.

such that

$|y| > 0$

$|xy| \leq c$

for $k \geq 0$, $xy^k z$ is in L.

4) If not, say L is a inregular.

problem:-

prove $L = \{ a^m b^n \mid gcd(m,n) = 1 \}$ is not a regular language.

Sol:- $L = \{aabbb, aaabbbb, aaaabbbbb, \ldots\}$

1. let L is a regular language

2. Constant $c = 5$, where $|w| \geq c$

3. Let $w = \underbrace{aa}_{x}\underbrace{abb}_{y}\underbrace{bbb}_{z}$

divide w into x, y, z such that $|y| > 0$ & $|xy| \leq c$

then $x = aa$, $y = ab$, $z = bbb$

for every $k \geq 0$, compute $xy^k z$

let $k = 0$.

$xy^0 z \Rightarrow x \cdot y^0 \cdot z = xz = aabbb$, this is in L

let $k = 1$.

$\Rightarrow xy^1 z \Rightarrow x \cdot y^1 \cdot z = xyz = aaabbbb$ is in L

let $k = 2$.

$\Rightarrow xy^2 z \Rightarrow x \cdot y^2 \cdot z = xyyz = aaabaabbbb$ is not in L

∴ The given language L is irregular not regular

Q: prove $L = \{0^n \mid n \text{ is a perfect square}\}$ is not a regular language.

Ans: $L = \{0, 0000, 000000000, 0000000000000000\}$

1) Let L is a regular language.

2) constant $c = 1$, where $|w| \geq c$

3) select $w = 0000$,

divide w in xyz such that

$|y| > 0$ & $|xy| \leq c$ then

$x = \epsilon$, $y = 0$, $z = 000$

for every $k \geq 0$, compute $xy^k z$

let $k = 0$,

$\Rightarrow xy^k z \Rightarrow xz = \epsilon \cdot 000 \Rightarrow 000$ is not in L

let $k=1$,

$\Rightarrow xy^kz \Rightarrow xyz \Rightarrow \epsilon \cdot 0 \cdot 000 \Rightarrow 0000$ is in L.

let $k=2$,

$\Rightarrow xy^kz = xyyz \Rightarrow \epsilon \cdot 0 \cdot 0 \cdot 0000 \Rightarrow 000000$ is not in L

∴ The given language L is not regular.

Context-free Grammers:- (CFG) set of rules.

→ Grammer is a Quadruple Notations

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

V → set of non-terminals (or) variables, (uppercase letters)

T → set of terminals (lowercase letters)

P → set of productions $(\alpha \to \beta)$ - these are constructed to terminals & Non-Terminals $(\alpha = aA, \beta = bBa)$

$\Rightarrow \alpha, \beta \in (V \cup T)^*$

S → start symbol

Types of Grammers:- 4 types of Grammers

1) Type 0:- It is a unrestricted Grammer.

→ The language Generated by this type Grammer is "Recursively Enumerable language (REL)".

→ To represent this we use "Turing Machine (TM)".

Produ
⇒ $\alpha \to \beta$

where α has atleast one non-terminal.

Context free Grammer (CFG):-

A context free grammer (G) is a 4 tuple notation. where $G = (V, T, P, S)$

Where
- V → set of non Terminals or Variables
  - → represented in Upper case letters.
- T → set of Terminals.
  - → represented in lowercase letters.
- S → start Symbol
- P → set of productions of the form $(\alpha \to \beta)$

where $\alpha \in V$, $|\alpha| = 1$ & (β can be a terminal or terminal followed by non-Terminal or non-terminal followed by a terminal)

Derivations:-

→ productions of the Grammer are used to infer the strings of the language.

→ There are two approaches
1) Recursive Inference
2) Derivations

1) Recursive Inference Approach:-

→ This is a bottom up approach (body to head).

→ We start with known symbols (terminals).

Grammar is:

1. E → I
2. E → E+E
3. E → E*E
4. E → (E)
5. I → b
6. I → a
7. I → Ib
8. I → Ia
9. I → Ia
10. I → Ib

→ The string to be derived is a*(a+boo)

a*(a+boo)	E*(E+Ioo)	E*E
I*(a+boo)	E*(E+Io)	E
E*(a+boo)	E*(E+I)	
E*(I+boo)	E*(E+E)	
E*(E+boo)	E*(E\	

S.No	string Infered	for language of	Productions used	steps used
1)	a	I	6	-
2)	a	E	1	1)
3)	b	I	5	-
4)	bo	I	7	3)
5)	boo	I	7	4)

6) boo E 1 5)

7) a+boo E 2 2),6)

8) (a+boo) E 4 7)

9) a*(a+boo) E 3 2),8)

2) Derivation Approach:-

→ This is a topdown approach.

→ Let "G" be a "CFG",

$$\alpha A \beta \in (V \cup T)^*$$

Where, A is a non terminal and $A \to \gamma$ then

$$\alpha A \beta \overset{*}{\underset{G}{\Rightarrow}} \alpha \gamma \beta$$

$\Rightarrow$ a * (a+boo)

Grammer is:

1. E → I
2. E → E+E
3. E → E*E
4. E → (E)
5. I → b
6. I → a
7. I → Ia
8. I → I1
9. I → Ia
10. I → Ib

E → E*E → I*E → a*E → a*(E) → a*(E+E)

→ a*(I+E) → a*(a+E) → a*(a+I) → a*(a+I0)

→ a*(a+I0) → a*(a+I00) → "a*(a+boo)".

Left Most Derivation And Right Most Derivation:

1) Left Most Derivation:

In which At each step of derivation

If we replace left most variable with its production body.

→ We represent left most derivation by $\overset{*}{\underset{lm}{\Rightarrow}}$

2) Right most Derivation:

In which, at each step of derivation, we replace rightmost variable with its production body.

→ We represent leftmost derivation by $\overset{*}{\underset{rm}{\Rightarrow}}$

Examples:-

→ E
→ E*E
→ E*(E)
→ E*(E+E)
→ E*(E+I)
→ E*(E+I0)
→ E*(E+I00)
→ E*(E+b00)

→ E*(I+b00)
→ E*(a+b00)
→ I*(a+b00)
→ a*(a+b00)

→ Derive "abababa" from the given Grammer and leftmost derivation approach.

$S \rightarrow SbS \mid a$

case i)

→ S

→ SbS {S → SbS}

→ SbSbS {S → SbS}

→ SbSbSbS {S → SbS}

→ abSbSbS {S → a}

→ ababSbS {S → a}

→ abababS {S → a}

→ "abababa" {S → a}

case ii) → S

→ SbS {S → SbS}

→ abS {S → a}

→ abSbS {S → SbS}

→ abSbSbS {S → SbS}

→ ababSbS {S → a}

→ abababS {S → a}

→ abababa {S → a}

Language of Context free Grammer:

Let "G" be a context free Grammer, then its Language is denoted by $L(G)$ and it is defined as $L(G) = \{w \text{ in } T^* / S \overset{*}{\underset{G}{\Rightarrow}} w\}$.

→ The Language of CFG is known as "context free Language".

Sentential Form:-

let "G" be a "CFG", $\alpha \in (V \cup T)^*$, S is a start Symbol. If $S \overset{*}{\underset{G}{\Rightarrow}} \alpha$ then α is, in "sentential Form".

→ There are two types:

1) left most sentential form:

If $S \overset{*}{\underset{lm}{\Rightarrow}} \alpha$ then your α is left most sentential form.

2) Rightmost Sentential form:

If $S \overset{*}{\underset{rm}{\Rightarrow}} \alpha$ then your α is rightmost sentential form.

Parse tree / derivation tree / syntax tree:-

A parse tree is a tree which is constructed with the following conditions:

1) root node must be a start symbol.

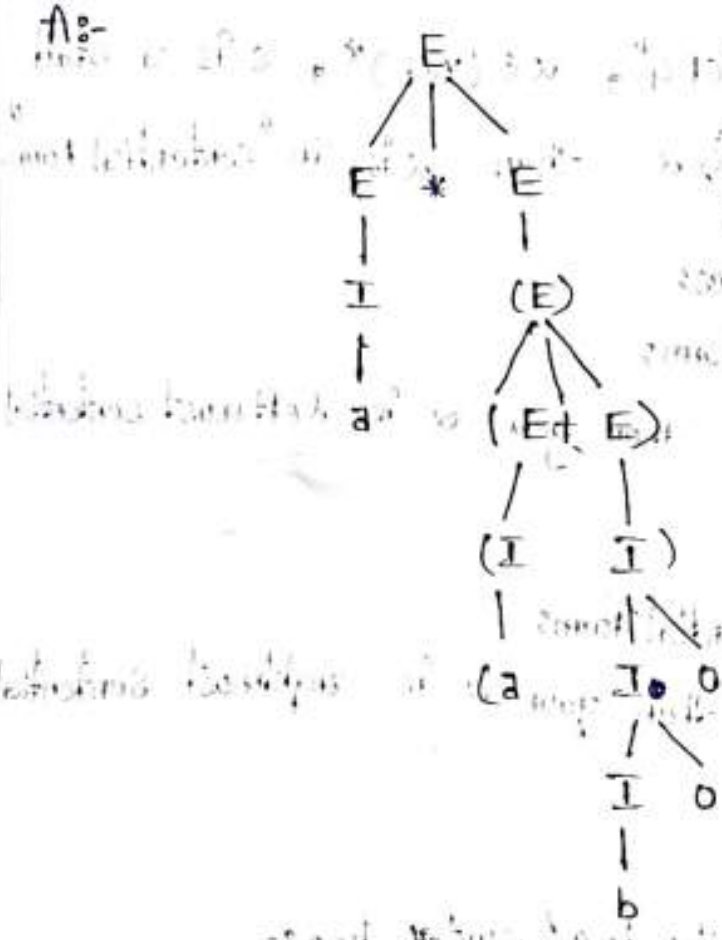
2) Intermideate / Internal node must be a variable (including starting stymbol).

3) leafnode must be a terminal (or) ε

→ construct parse tree for the string

a*(a+boo) using the CFG E→E+E/E*E/(E)/I. I→Ib/Ia/Io/I1/b/a.

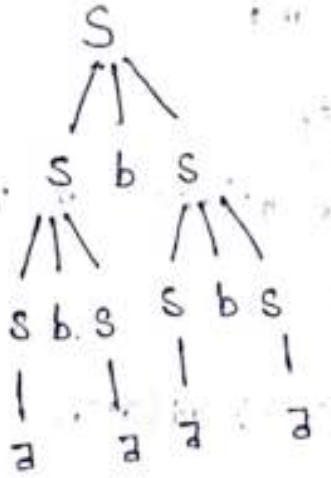
A:-



→ construct parse tree for the string

abababa using the Grammer

S → sb / a.

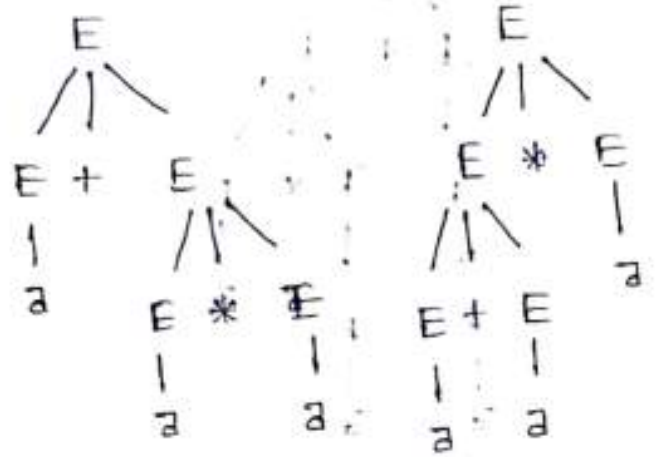


Ambiguos Grammer:

A "CFG" "G" is said to be ambiguos for atleast one string in its language if there exist two distinct parse trees.

Example: Grammer is $E \to E+E$, $E \to E*E$, $E \to a$

construct "a+a*a" parse tree.



∴ The Given Grammer is ambiguos Grammer.

Removing Ambiguty:

Removing ambiguty is done

i) by adding precedence and

ii) by adding associativity.

→ To implement this these take an unequivalent an unambiguous grammer.

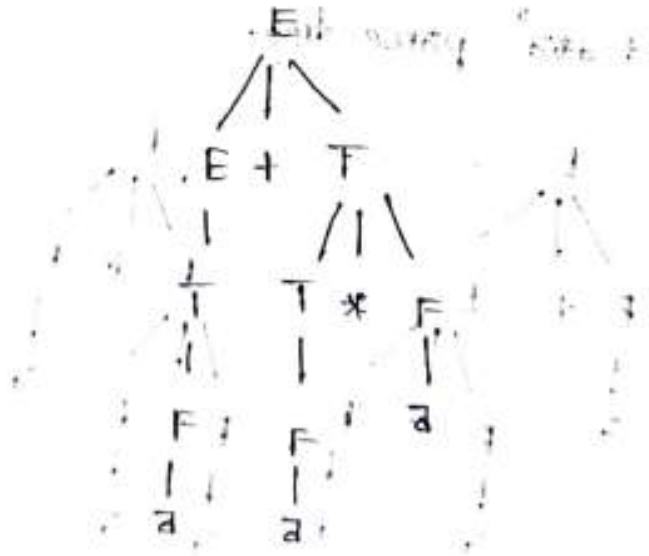
⇒ a+a+a ⇒ (a+a)+a ⇒ a+(a+a) !

⇒ E→E+E · → E→E*E → F→a

→ E→E+T → T→T*F

→ E→T → T→F

a+a*a



Leftmost derivations in ambiguity:

→ There are more derivations in Leftmost and Rightmost ambiguity.

→ There are only one derivation in the Un ambiguty Grammer.

Inherent Ambiguos:-

A language is said to be ambiguty if all its Grammer is ambiguos.

= Unit-III =

push down Automata (PDA):-

→ If we add stack to the finite Automata then it will become Pushdown Automata.

→ If we want to store the infinite information, we cannot store in finite automata then we use the push down Automata.

→ PDA = FA + stack

→ This is used to represent the context free Grammer.

→ In PDA, we use three components

1) input tape / input string
2) finite Control (Push, Pop)
3) Stack

Formal Defination of PDA:-

→ It is represented by "P", it is a 7 tuple

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

where, P = PDA

Q = set of states

$\Sigma \rightarrow$ set of input symbols

$T \rightarrow$ set of stack elements

$\delta \rightarrow$ It is a transition Function

$$\delta(q_0, a, x) = (q_n, \alpha)$$

$q \rightarrow$ current state

$a \rightarrow$ input to be processed

$x \rightarrow$ top of the stack

$q_n \rightarrow$ new state

$\alpha \rightarrow$ symbol to be replaced in place of x

$q_0 \rightarrow$ Initial state

$z_0 \rightarrow$ top of stack

$F \rightarrow$ set of accepting states/Final states

Q:- Construct PDA for the given language

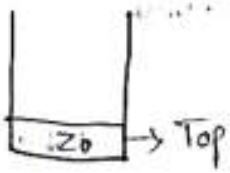
$$L = \{a^n b^n / n \geq 1\}$$

Sol:- $L = \{a^n b^n / n \geq 1\}$

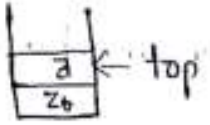
$L = \{ab, aabb, aaabbb, \ldots\}$

Let us assume "aaabbb" is our input tape/string, q_0 is the initial state.

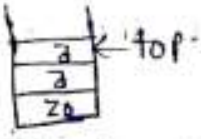
Let z_0 be at the top of the stack.



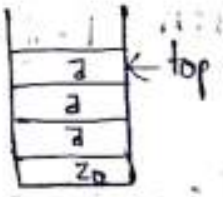
$\delta(q_0, a, z_0) = (q_0, az_0)$



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

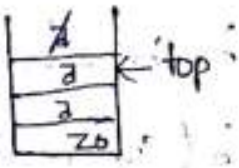


$\delta(q_0, a, a) = (q_0, aaa)$

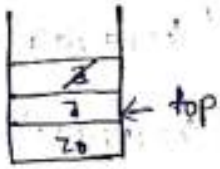


$\delta(q_0, b, a) = (q_1, \epsilon)$

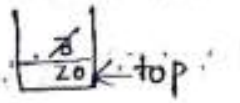
$\delta(q_1, b, a) = (q_1, \epsilon)$



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



$\delta(q_1, \epsilon, z_0) = (q_2, \epsilon)$



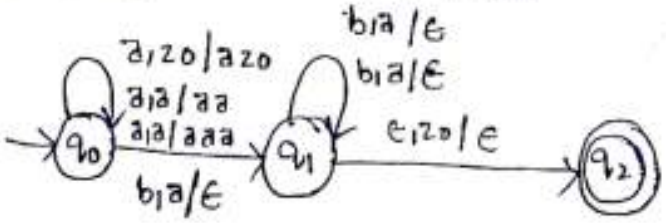
Stack empty.

$PDA = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

$Q \rightarrow \{q_0, q_1, q_2\}$ $\quad \Gamma \rightarrow \{a, z_0\}$ $\quad F \rightarrow \{q_2\}$

$\Sigma \rightarrow \{a, b\}$ $\quad \delta$

$q_0 \rightarrow q_0$ $\quad z_0 \rightarrow \{z_0\}$



Instantaneous Description of PDA's-

→ To check the acceptance of string

Find the acceptance of a string aabbb by the PDA : $P = \{q_0, q_1, q_2\}$

$P = [\{q_0, q_1, q_2\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \{q_2\}]$

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

$\delta(q_0, a, az_0) = (q_0, aaz_0)$

$\delta(q_0, a, aa) = (q_0, aaa)$

$\delta(q_0, b, aaa) = (q_1, \varepsilon)$

$\delta(q_0, b, aa) = (q_1, \varepsilon)$

$\delta(q_1, b, a) = (q_1, \varepsilon)$

$\delta(q_1, \varepsilon, z_0) = (q_2, z_0)$

$\delta(q_0, aabb, z_0) \vdash (q_0, abb, az_0)$

$\vdash (q_0, bb, aa)$

$\vdash (q_1, b, a)$

$\vdash (q_1, \varepsilon, z_0)$

$\vdash (q_2, z_0)$

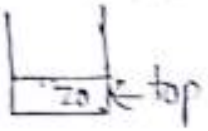
∴ the Given string aabb is accepted by $t(q_2, z_0)$ by the PDA

construct PDA to accept the language $L = \{a^n b^{2n} / n \geq 1\}$ and find acceptance of string "aaabbbbbb".

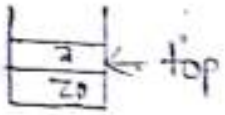
A:- push for "a" and pop for every two b's.

$L = \{abb, aabbbb, aaabbbbbb, \ldots\}$

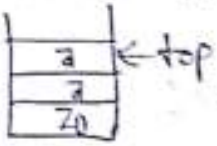
Let aabbbb $\in$ is the input tape, q_0 is the initial state and z_0 is at the top of stack



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



$\delta(q_0, a, a) = (q_0, aaz_0)$



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

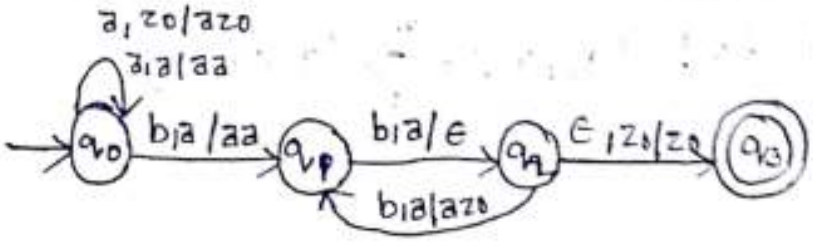
$\delta(q_1, b, a) = (q_2, \epsilon)$

$\delta(q_1, b, a) = (q_1, az_0)$

$\delta(q_1, b, a) = (q_2, \epsilon)$

$\delta(q_e, \epsilon, z_0) = (q_3, z_0)$

$P = [\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \{q_3\}]$



$\delta(q_0, aaabbbbbb, z_0) \vdash (q_0, aabbbbbb, az_0)$

$\vdash \delta(q_0, abbbbbb, aa) \vdash (q_0, bbbbbb, aaa)$

$\vdash (q_1, bbbbb, aaa) \vdash (q_2, bbbb, aa) \vdash (q_1, bbba$

$\vdash (q_2, bbb, a) \vdash (q_1, bb, a) \vdash (q_2, b, a) \vdash (q_3, z_0)$

∴ The given string aaabbbbbb is accepted $\vdash (q_3, z_0)$ by the given PDA.

→ Construct PDA for the given language $L = \{a^n b^m c^n / m, n > 0\}$

A:- push for 'a's, Pop for 'c's and 'b's for no change. $m=2, n=3$, $m=1, n=2$

$L = \{abc, abbc, aaabbccc, aabcc, \ldots\}$

languages of PDA:-

1) Acceptance by Final state

$$L = \{w \mid \delta(q_0, w, z_0) \overset{*}{\underset{P}{\Rightarrow}} (q, \epsilon, \alpha)\}$$

continuation of $L = \{a^m b^n c^n / m, n > 0\}$

let aabcc ϵ as input tape, q_0 is the initial state and z_0 at the top of the stack.

→ $\delta(q_0, a, z_0) = (q_0, az_0)$

→ $\delta(q_0, a, a) = (q_0, aa)$

→ $\delta(q_0, b, a) = (q_1, aa)$

→ $\delta(q_1, c, a) = (q_e, a)$

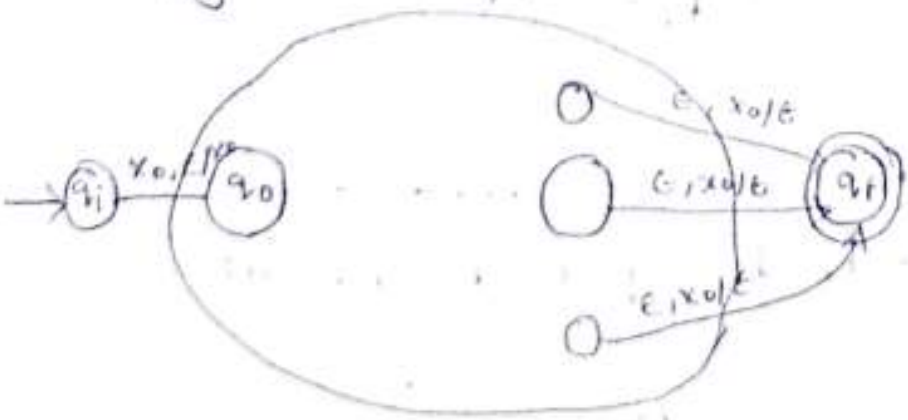
→ $\delta(q_2, c, a) = (q_2, z_0)$

→ $\delta(q_2, \epsilon, z_0) = (q_3, z_0)$

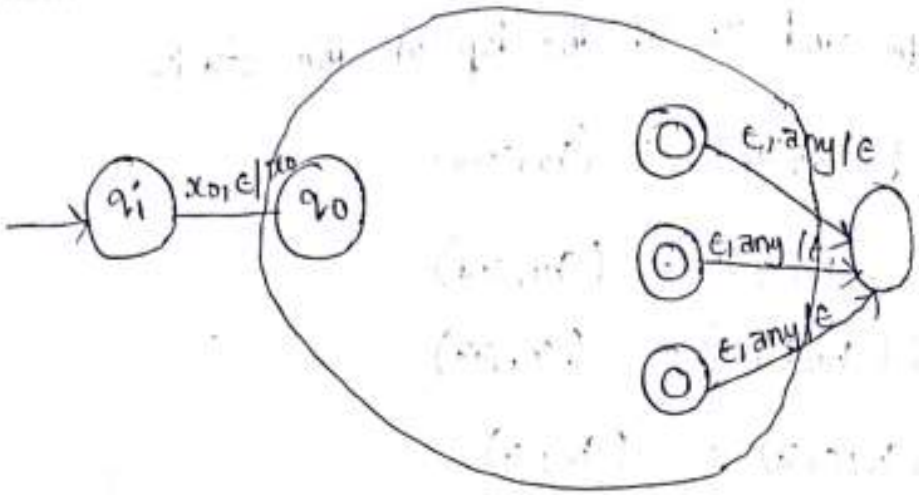
3) Acceptance by Empty stack:

$N(P) = \{ w / \delta(q_0, w, z_0) \overset{*}{\underset{P}{\Rightarrow}} (q, \epsilon, \epsilon) \}$

2) From empty stack to final state:



4) From final state to empty stack



→ Equivalence of PDA and CFG's

Conversion of CFG to PDA:-

A For non terminal: $A \to \alpha$

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

For terminal:

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

d

Example:-

Convert the given CFG into PDA

$S \to AB$

$A \to 0S/0$

$B \to 1S/1$

$S \to AB$ $B \to 1S$ $A \to 0$

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

Consider $S \to AB$

$$\delta(q_1, \epsilon, S) = (q_1, AB)$$

$A \to 0S$

$$\delta(q_1, \epsilon, A) = (q_1, 0S)$$

$A \to 0$

$$\delta(q_1, \epsilon, A) = (q_1, 0)$$

$B \to 1S$

$$\delta(q_1, \epsilon, B) = (q_1, 1S)$$

$B \to 1$

$$\delta(q_1, \epsilon, B) = (q_1, 1)$$

$$\Rightarrow \delta(q_1, 0, 0) = (q_1, \epsilon)$$

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

$$\delta(q_1, \epsilon, S) = \{(q_1, AB)\}$$

$$\delta(q_1, \epsilon, 0) = \{(q_1, 0S), (q_1, 0)\}$$

$$\delta(q_1, \epsilon, 1) = \{(q_1, 1S), (q_1, 1)\}$$

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

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

Q construct PDA

convert the given CFG to PDA:-

$E \to E+E \mid E*E \mid I \mid (E)$

$I \to I_0 \mid I_1 \mid I_a \mid I_b \mid a \mid b$

→ $E \to E+E$

for Non-Terminal $A \to \alpha$

$\delta(q_1, \epsilon, A) = (q_1, \alpha)$

→ $E \to E+E$

$\delta(q_1, \epsilon, E) = (q_1, E+E)$

→ $E \to E*E$

$\delta(q_1, \epsilon, E) = (q_1, E*E)$

→ $E \to I$

$\delta(q_1, \epsilon, E) = (q_1, I)$

→ $E \to (E)$

$\delta(q_1, \epsilon, E) = (q_1, (E))$

→ $I \to I_0$

$\delta(q_1, \epsilon, I) = (q_1, I_0)$

→ $I \to I_1$

$\delta(q_1, \epsilon, I) = (q, I_1)$

→ $I \to I_a$

$\delta(q_1, \epsilon, I) = (q_1, I_a)$

→ $I \to I_b$

$\delta(q, \epsilon, I) = (q, Ib)$

→ $I \to a$

$\delta(q, \epsilon, I) = (q, a)$

→ $I \to b$

$\delta(q, \epsilon, I) = (q, b)$

For Terminal

$T = \{a, b, +, *, 0, 1\}$

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

→ $\delta(q, a, a) = (q, \epsilon)$

→ $\delta(q, b, b) = (q, \epsilon)$

→ $\delta(q, 0, 0) = (q, \epsilon)$

→ $\delta(q, 1, 1) = (q, \epsilon)$

→ $\delta(q, +, +) = (q, \epsilon)$

→ $\delta(q, *, *) = (q, \epsilon)$

$\Rightarrow \delta(q, E, E) = \{(q, E+E), (q, E*E), (q, I), (q, (E)\}$

$\Rightarrow \delta(q, \epsilon, I) = \{(q, I0), (q, I1), (q, Ia), (q, Ib), (q, a), (q, b)\}$

$\therefore PDA = \left[\{q\}, \{a, b, 0, 1, +, *\}, \Gamma, \delta, q, z_0, \{q\}\right]$

Conversion of PDA to CFG:- based on Transition functions

convert the given PDA to CFG

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

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

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

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

$\delta(q_0, \epsilon, a) = (q_1, \epsilon)$

if there are no stack symbols - 1

$1 \to 2$

$2 \to 4$.

$PDA = [\{q_0, q_1\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \{q_1\}]$

$Q = \{q_0, q_1\}$

$\therefore |Q| = 2$

Ans:- Let us assume, S be the start symbol

Find variables

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

$T = \{a, b\}$

Find productions:

$\to \delta(q_0, a, z_0) = (q_0, az_0)$

$[q_0 z_0 q_0] \to a [q_0 a q_0] [q_0 z_0 q_0]$

$[q_0 z_0 q_0] \to a [q_0 a q_1] [q_1 z_0 q_1]$

$[q_0 z_0 q_1] \to a [q_0 a q_0] [q_0 z_0 q_1]$

$[q_0 z_0 q_1] \to a [q_0 a q_1] [q_1 z_0 q_1]$

→ $\delta(q_0, a, a) = (q_0, aa)$

$[q_0\ a\ q_0] \rightarrow a\,[q_0\ a\ q_0]\ [q_0\ a\ q_0]$

$[q_0\ a\ q_0] \rightarrow a\,[q_0\ a\ q_1]\ [q_1\ a\ q_1]$

$[q_0\ a\ q_1] \rightarrow a\,[q_0\ a\ q_0]\ [q_0\ a\ q_1]$

$[q_0\ a\ q_1] \rightarrow a\,[q_0\ a\ q_1]\ [q_0\ a\ q_0]$

→ $\delta(q_0, b, a) = (q_1, \epsilon)$

$[q_0\ a\ q_0] \rightarrow b\,[q_1\ a\ q_0]$

$[q_0\ a\ q_1] \rightarrow b\,[q_1\ a\ q_1]$

→ $\delta(q_1, b, a) = (q_1, \epsilon)$

$[q_1 a q_1] \rightarrow b$

→ $\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$

$[q_1\ z_0\ q_1] \rightarrow \epsilon$

$S \rightarrow [q_0 z_0 q_0]$

$S \rightarrow [q_0 z_0 q_1]$

Construct CFG for the PDA Given PDA

$\delta(q_1, 1, z_0) = (q_1, 1z_0)$

$\delta(q_1, 1, x) = (q_1, xx)$

$\delta(q_1, 0, x) = (p_1, x)$

$\delta(q_1, \epsilon, x) = (q_1, \epsilon)$

$\delta(p_1, 1, x) = (p_1, \epsilon)$

$\delta(p_1, 0, z_0) = (q_1, z_0)$

$P = \{\{p, q\}, \{0, 1\}, \{x, z_0\}, \delta, q_1, z_0, \{p\}\}$

Sol:- Let S be the start Symbol

Let CFG $\Rightarrow G = (V, T, P, S)$

Find Variables

$V = \{S, [PxP], [PxQ], [QxP], [QxQ],$

$[PzoP], [PzoQ], [QzoP], [QzoQ]\}$

$T = \{0, 1\}$

P → Find productions: $S \to Q z_0 Q$ $S \to Q z_0 P$

$\to \delta(q, 1, z_0) = (q, xz_0)$

$[q\ z_0\ q] \to 1\,[q\ x\ q]\ [q\ z_0\ q]$

$[q\ z_0\ q] \to 1\,[q\ x\ p]\ [p\ z_0\ q]$

$[q\ z_0\ p] \to 1\,[q\ x\ q]\ [q\ z_0\ p]$

$[q\ z_0\ p] \to 1\,[q\ x\ p]\ [p\ z_0\ p]$

$\to \delta(q, 1, x) = (q, xx)$

$[q\ x\ q] \to 1\,[q\ x\ q]\ [q\ x\ q]$

$[q\ x\ q] \to 1\,[q\ x\ p]\ [p\ x\ q]$

$[q\ x\ p] \to 1\,[q\ x\ q]\ [q\ x\ p]$

$[q\ x\ p] \to 1\,[q\ x\ p]\ [p\ x\ p]$

→ $\delta(q, 0, x) = (p, x)$

$[q \times q] \rightarrow 0 [p \times q]$

$[q \times p] \rightarrow 0 [p \times q]$

→ $\delta(q, \epsilon, x) = (q, \epsilon)$

$[q \times q] \rightarrow \epsilon$

→ $\delta(p, 1, x) = (p, \epsilon)$

$[p \times p] = 1$

→ $\delta(p, 0, z_0) = (q, z_0)$

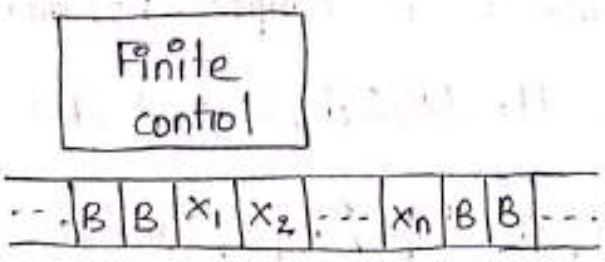
$[p\ z_0\ q] = 0 [q\ z_0\ q_0]$

$[p z_0 p] \rightarrow 0 [q z_0 p]$

Turing Machine:

Turing machine consists of a finite control, which can be any finite set of states.

→ There is a tape divided into cells, each cell can hold finite number of symbols.



$X_1, X_2, \ldots X_n \rightarrow$ Input Symbols

B → Blank symbol

→ choose input string from the input alphabet and place in tape.

→ Extend the length of the tape to the left and right and place blank symbol over the extended cells.

→ The Blank is a tape but not input symbol.

→ There is a read write head (or) tape head; Initially the tape head is at leftmost cell that holds the input.

→ A move of the turing machine is a function of the state of the finite control and tape symbol scanned.

→ In one move a turing machine will

1) change state: The next state optionally may be same as the current state.

2) Write a tape symbol to the cell scanned, this tape symbol replaces the symbol in that cell.

3) move the tape head to left or right but don't allow the head to remain stationary.

Defination:-

Turing machine is a "7 tuple", notation, And the symbol is $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$.

$\Gamma \rightarrow$ input tape / Tape symbols

$B \rightarrow$ Blank symbol

$Q \rightarrow$ set of states

$\Sigma \rightarrow$ set of input symbols

$q_0 \rightarrow$ Initial / starting state

$F \rightarrow$ Final state set

$\delta \rightarrow$ Transistion Function

where, the transistion Function maps to

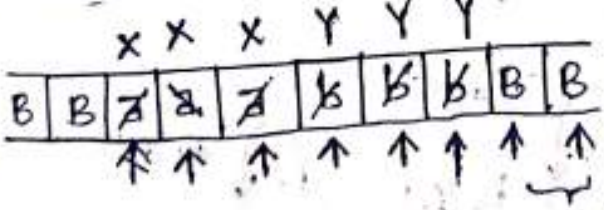
$Q \times \Sigma \Rightarrow "Q \times \Gamma \times \{L, R\}"$.

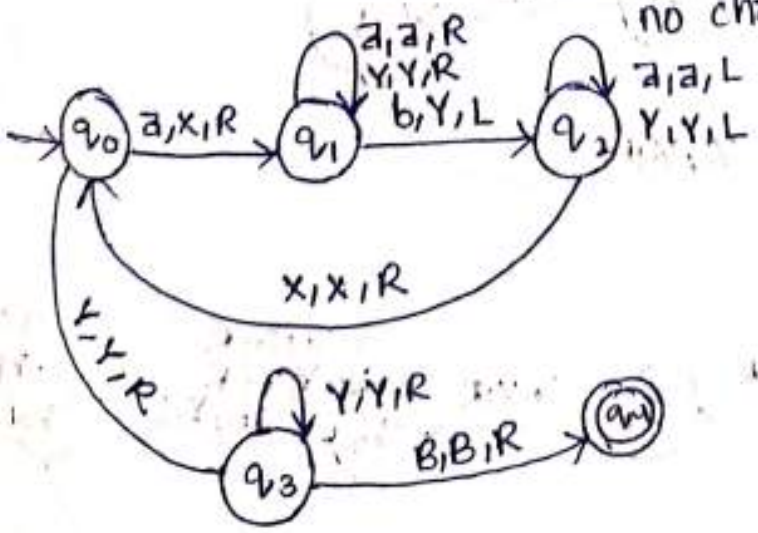
Transistion diagram in Turing machine:

Q Design Turing machine for the language $L = \{a^n b^n / n > 0\}$.

Ans: $L = \{ab, aabb, aaabbb, \ldots\ldots\}$

let "aaabbb" be the input tape, q_0 be the initial state.





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

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

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

$\Gamma = \{a, b, X, Y, B\}$

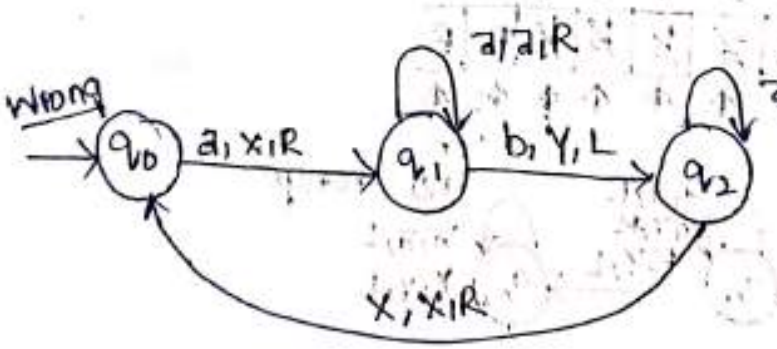
$F = \{q_4\}$

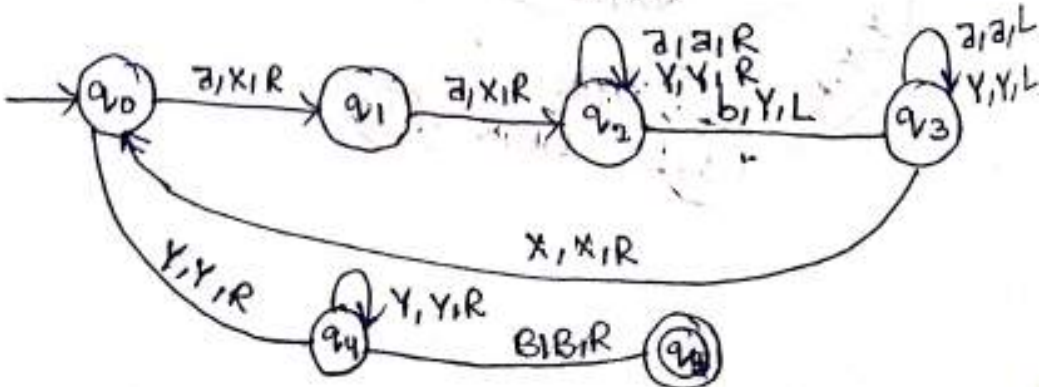
② Design Turing machine for the language

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

A:- $L = \{ab, aabb, aaabbb, \ldots\}$

Let "aaaabb" be the input tape and q_0 be the initial state. [∵ For 2 a's we need to change to X].





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

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

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

$\Gamma = \{a, b, X, Y, B\}$

$F = \{q_5\}$

Transition Table:-

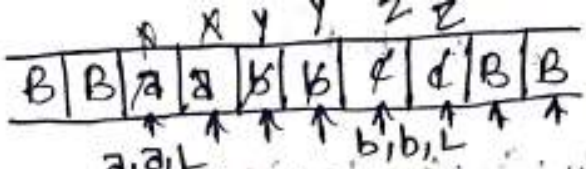
	a	b	X	Y	B
q_0	q_1, X, R	-	-	q_4, Y, R	-
q_1	q_2, X, R	-	-	-	-
q_2	q_2, a, R	q_3, b, L	-	q_2, Y, R	-
q_3	q_3, a, L	-	q_0, X, R	q_3, Y, L	-
q_4	-	-	-	q_4, Y, R	q_5, B, R
q_5	-	-	-	-	-

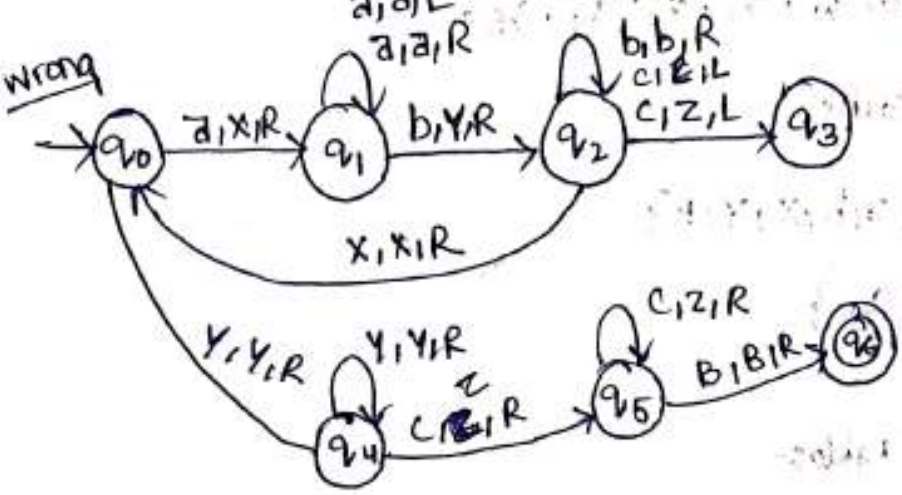
(3) Design Turing machine for the given language

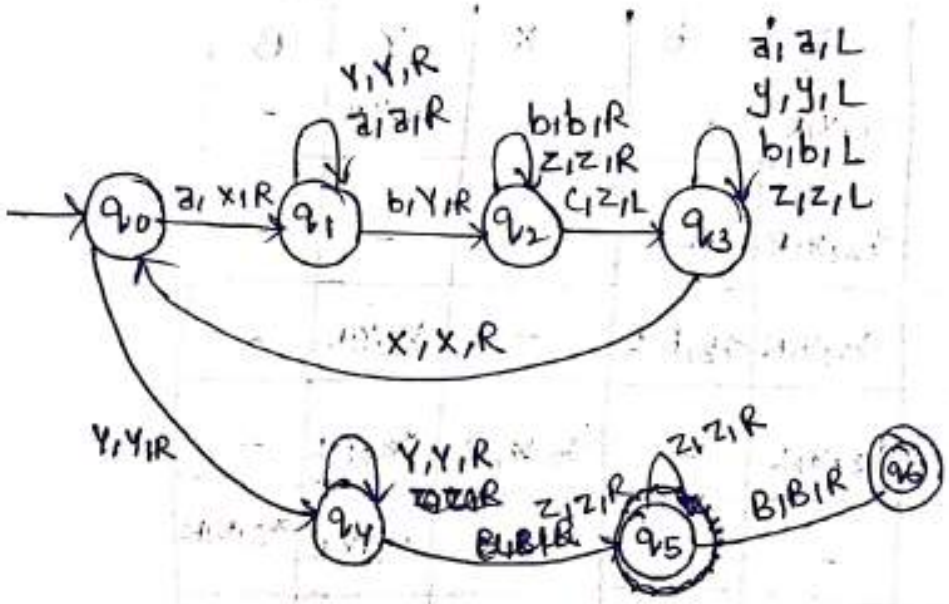
$L = \{a^n b^n c^n / n > 0\}$.

A:- $L = \{abc, aabbcc, aaabbbccc, \cdots\}$

let "aabbcc" be the Input tape and q_0 be the initial state.







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

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

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

$\Gamma = \{a, b, c, X, Y, B\}$

$F = \{q_6\}$

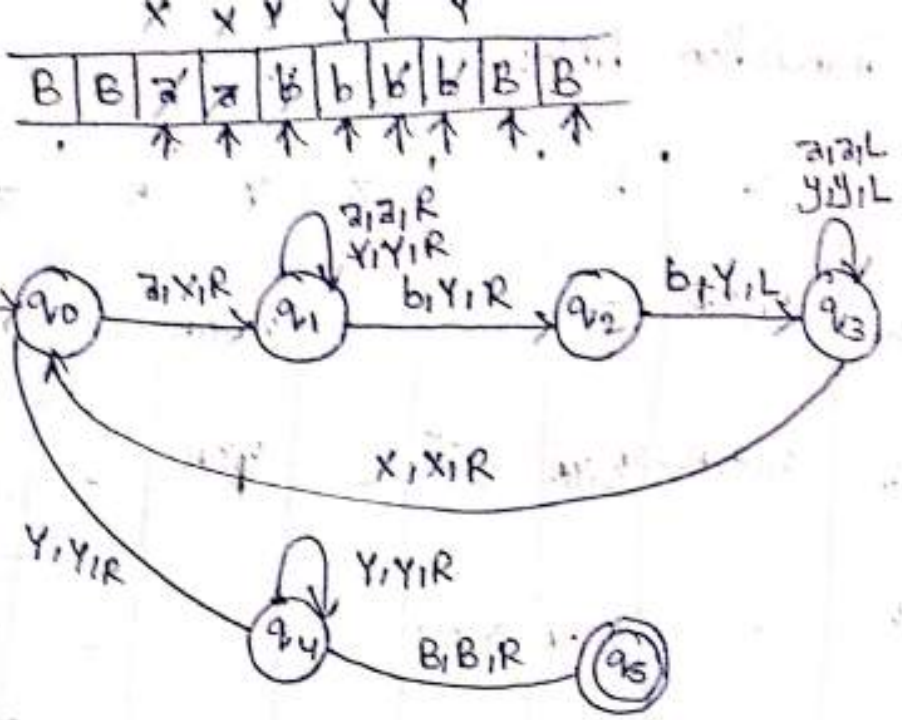
Transition table:

	a	b	c	X	Y	B	Z
q_0	q_1, X, R	-	-	-	q_4, Y, R	-	
q_1	q_1, a, R	q_2, Y, R	-	-	q_1, Y, R	-	-
q_2	-	q_2, b, R	q_3, Z, L	-	-	-	q_2, Z, R
q_3	q_3, a, L	q_3, b, L	-	q_0, X, R	q_2, Y, L	-	q_3, Z, L
q_4	-	-	-	-	q_4, Y, R	-	q_5, Z, R
q_5	-	-	-	-	-	q_6, B, R	q_5, Z, R
q_6	-	-	-	-	-	-	-

(4) Design turing machine for the language $L = \{a^n b^{2n} / n \geq 1\}$

A:- $L = \{abb, aabbbb, aaabbbbbb, \ldots\}$

let "aabbbb" be the input tape and q_0 be the initial state [∵ for 2 b's we need to change to Y]



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

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

$\Gamma = \{a, b, X, Y, B\}$

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

$F = \{q_5\}$

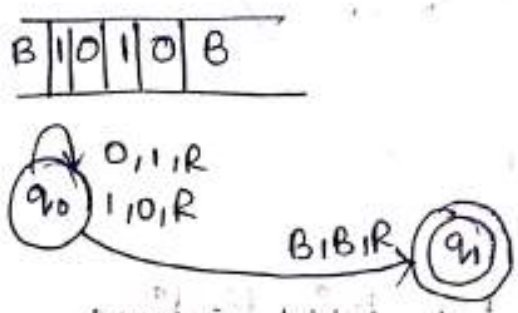
Transistion Table:

	a	b	X	Y	B
q_0	q_1, X, R	–	–	q_4, Y, R	–
q_1	q_1, a, R	q_2, Y, R	–	q_1, Y, R	–
q_2	–	q_2, Y, L	–	–	–
q_3	q_3, a, L	–	q_0, X, R	q_3, Y, L	–
q_4	–	–	–	q_4, Y, R	q_5, B, R
q_5	–	–	–	–	–

(5) construct Turing machine for

i) Ones complement ii) Two's complement.

A:- i) One's complement



⇒ Two's complement

1 0 1 0 1

0 1 0 1 0

1

0 1 0 1 1

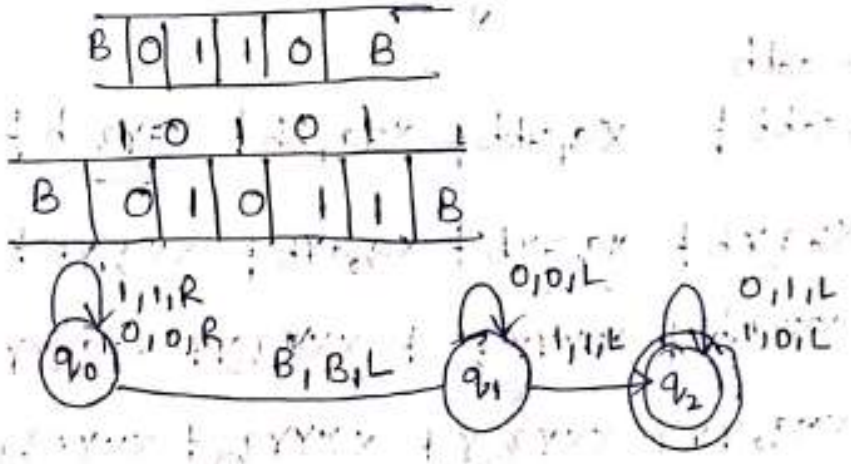
ii) Two's complement

1 0 1 0

0 1 0 1

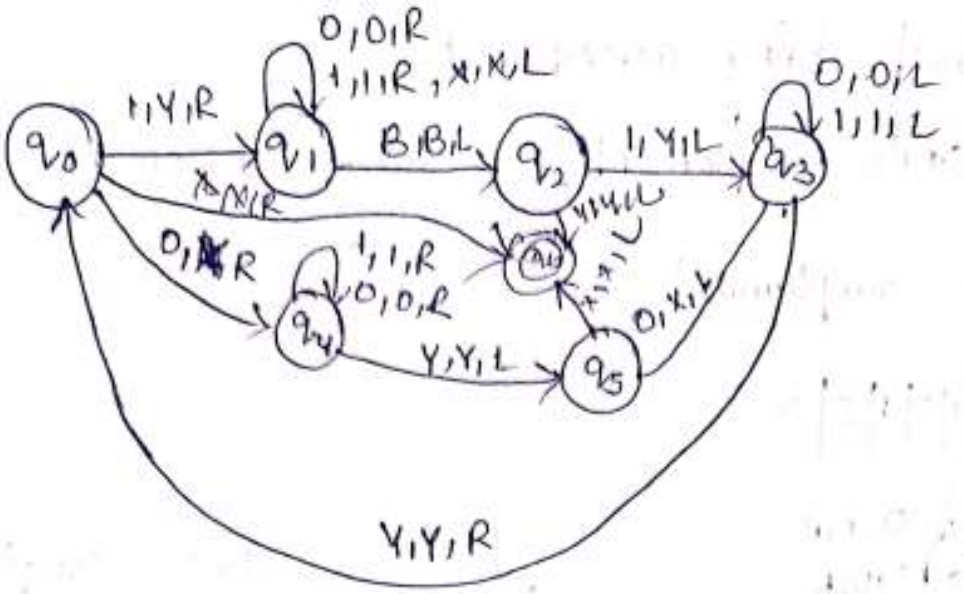
1

0 1 1 0



(6) construct turing machine for the palindrome of the 0's and 1's.

0 10, 000, 0101, 01, 111



Instantaneous Description For Turing Machine:-

Let "aaabbb" be the input string/tape.

Acceptance For the Turing machine:

$\omega = aabb$

$q_0 aabb \vdash Xq_1 abb \vdash Xaq_1 bb \vdash Xaq_2 Yb \vdash$

$Xaq_2 Yb \vdash Xq_2 aYb \vdash Xq_0 aYb \vdash XXq_1 Yb \vdash$

$XXYq_1 b \vdash XXYYq_2 \vdash XXYq_2 Y \vdash XXq_2 YY \vdash$

$XXq_0 YY \vdash XXYq_3 Y \vdash XXYYq_3 \vdash XXYYq_4$

∴ The Acceptance of Turing machine is verified.

Language of Turing machine:

Language of Turing machine M is denoted by L(M) and it is defined as

$$L(M) = \{ w \text{ in } \Sigma^* / q_0 w \vdash^* \alpha p \beta \}$$

$\Rightarrow \alpha, \beta$ are any strings of tape symbols

$\Rightarrow$ P can be any state.

Undecidability :-

Recursive language: A language, if their exists a turing machine, That accepts all strings in L and rejects all strings not in L.

$$L = \{ a^n b^n \}$$

$$= \{ ab, aabb, aaabbb, \ldots \}$$

Let "abab" be the input tape symbol, if we find the Acceptance for this, this is not present in the language then it is a "recursive language".

Recursively Enaumerable language: If the A language, if their easits a turing machine that accepts all strings in L that may or may not rejects all strings in the L.

→ These are said to be partially decidable.

⊖ The)

Decidability:-

The recursive languages are said to be decidable.

Undecidability:

A language which is not decidable is said to be undecidable.

→ There is no turing machine for the undecidable languages.

→ The languages which are not recursively enumerable.

Language that is not recursively Ennumerable:-

1) In order to prove that not a recursively enumerable we need to

i) Ennumerating binary strings:- In order to prove this we need to write "binary strings".

{ε, 0, 1, 00, 01, 10, 11, 000, 001, ...}

Here "ε" is the "0th" string

"0" is the "1st" string and so on.

Constructing codes for turing machine: Transistion

→ Assign positive integers to the symbols of in turing machine.

→ For input symbols, Assign the / Take the $x_1(0)$, $x_2(1)$ and $x_3(B)$.

→ For directions, Take the $D_1(L)$, $D_2(R)$.

Diagnolized Language: It is represented by "L_d".

It is a set of strings "w_i" and "w_i" is not in "Turing machine (or) $L(M_i)$" for some "M_i".

Table of Acceptance: We have write "0 and 1" in the acceptance table. Rows we take Turing machine and for columns we take strings. The strings must be accepted by the turing machine.

→ If turing machine accepts take "1", if not accepts we takes "0".

→ $L(M_i)$ will have the diagonal of w_i and M_i.

Proof of the language that is not recursively enumerable (Diagnolized Language):-

let L_d be $L(M_i)$ for some M_i.

1) If w_i is in L_d, then w_i is in $L(M_i)$.
But, by the defination L_d, w_i is not in the $L(M_i)$ L_d.

2) If w_i is not there in L_d, then it is not there in $L(M_i)$.

But by the defination of L_d, w_i is in the L_d.

∴ There is no turing machine for the L_d.

So that we can have turing machine for L_d, which means L_d is not recursively enumerable language. This proof is called as "proof by contradiction". This means: Firstly we assume that there will be the L_d later that we assume there will not have L_d.

Undecidable problem that is not recursively Enumerable:

Complements of recursive and recursively enumerable language:

Theorem 1: If L is recursive then, $\bar{L}$ is also recursive.

proof:- Let "L" be a recursive language.

So, "L" halts always for Turing machine "M".

→ Let "M'" be the Turing machine of "$\bar{L}$".

To some changes. To get "M'" do some changes:

→ Make all accepting states as non-accepting states.

→ Take a new state as a accepting state and write translations from each non-accepting state of "M" on each tape symbol to the accepting state.

The Turing machine "M'" accepts strings which are not accepted by "M" and rejects all strings which are accepted by "M".

That means, the Turing machine "M'" always halts.

∴ "$\bar{L}$" is recursive.

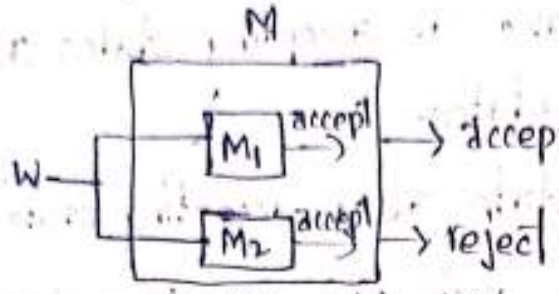
Theorem:- If "L" and "$\bar{L}$" are recursively enumerable then "L" is recursive.

Proof:- Take a Turing Machine "M".

(Let "L" be recursive)

Let "M_1" be the Turing machine for "L" and "M_2" be the Turing machine for "$\bar{L}$".

we can have a Turing machie 'M' by taking M_1 & M_2 as tapes.



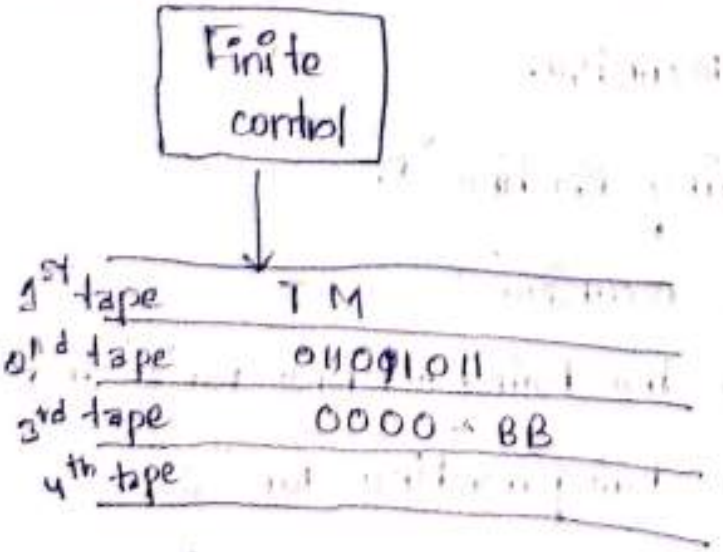
"M_1" acts same as "M"

Since "M" always halts

∴ L is recursive

Universal language:-

It is a language of strings representing turing machine and an input that is accepted by the turing machine.



→ 1st tape holds TM, w

→ 2nd tape holds simulated code of TM

→ 3rd tape holds the state q_i in oform.

→ 4th tape is used to do scratch.

→ If it is represented by the "d".

① Examine input to ensure that the code of Turing machine is legitimate.

② place input in 2nd tape in encoded form i.e, place 10 for each '0', 100 for each 1 and 1000 for each B.

③ place start state (0) in tape 3, and also place tape head at 1st symbol of simulated in 2nd tape.

④ for each move of TM of the form

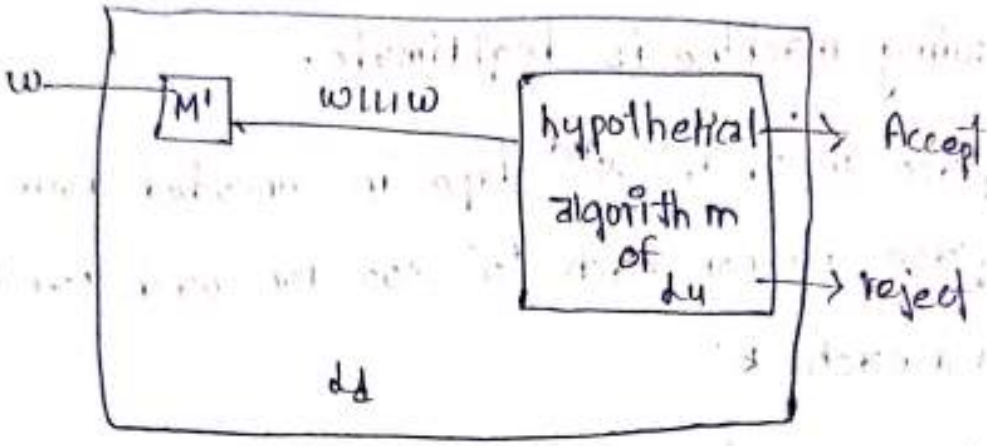
$$\delta(q_i, x_j) = (q_k, x_l, Dm)$$

a) replace the contents 0^j with 0^l

b) change the content of tape 3 by places 0^k and make all previous 0's, 10's.

Theorem: If L_u is RE, then L_u is not recursive

Proof: Let $\bar{L}_u$ is Recursively Ennumerable, then there exists a Turing Machine M.



If there exist TM for $\bar{L}_u$ then TM can also possible for L_d.

As per the defination of L_d, there exist no TM for L_d.

So that there exist no TM fo $\bar{L}_u$ i.e. L_u is non requ recursively ennumerable.

So that L_u is not recursive.

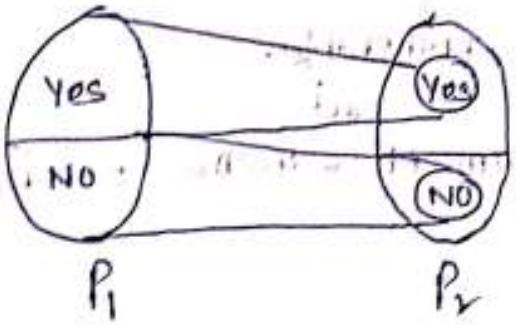
Undecidability about Turing machine:

Turing Reducability: It is a technique to convert one language into another language, when L_1 is recursive in L_2

→ In general an instance of, If there is an algorithm to convert P_1 to P_2 then we can say that an arbitrary instance of P_1 is reduced to an instance of P_2 with the same answer.

→ If P_1 is (recursively enumerable) is not recursive then P_2 is not recursive.

→ If P_2 is not recursive then P_1 can't be recursive.



Reduction turns the positive.

Reduction from P_1 to P_2 is TM which take an instance of P_1 and accepts instance of P_2.

In General, The reduction from P_1 to P_2 is program which states instance of P_2 as Output.

Theorem:- If there is reduction from P_1 to P_2

a) If P_1 is undecidable, then P_2 is also undecidable

Proof:-

Let P_1 is undecidable and then exists an algorithm to solve P_2

So, P_2 is decidable, but P_1 is reduced to P_2

Since, there is reduction from P_1 to P_2 and P_1 is undecidable.

So that P_2 is also undecidable.

b) If P_1 is non-recursively enumerable then P_2 is also non-recursively enumerable.

Proof: Let P_1 be form RE and there exists an algorithm for P_2

Since there is a reduction from P_1 to P_2 and P_1 is noth RE P_2 is non RE.

Turing Machine for empty language:

L_e → empty language

L_{ne} → non-empty language

$L_e = \{M / L(M) \neq 0\}$

$L_{ne} = \{M / L(M) \neq 0\}$

→ L_e and L_{ne} are complement to each other.

→ L_{ne} is recursively enumerable, and L_{ne} is not recursive.

→ L_e is not recursively enumerable.

Rice Theorem and Property of RE Languages :-

Property of RE Languages:- It is a set of RE languages. There are two properties

① Trivial Property of RE Language: A property said to be Trivial, when it is has no strings or when it is empty or a set of all RE languages.

#② Non-Trivial Property of RE languages: A property said to be nonTrivial, when it is a non-empty not a set of all RE languages.

Rice Theorem:-

For Every non-Trivial property of a recursively enumerable language is undecidable.

Implementation of Turing machine Specifications:-

Applications of Rice Theorem:-

① The Rice theorem is used to say the given problems are undecidable or not.