Formal Languages and Automata Theory

Formal Language :- Formal language is an abstraction of the general characteristics of programming language. It consists of set of rules and set of symbols by which the symbol combined into sentences.

Automata Theory :- It is a mathematical model which describe the behaviour of an automatic machine.

Central concepts of Automata Theory :-

Symbol :- A symbol can be any letter (or) digit.

Ex :- a, b, 0, 1, ...

Alphabet :- An alphabet is a finite and non empty set of symbols.

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

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

$\Sigma = \{0, 1, 2\}$

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

It is denoted by 'Σ'.

String :- A string is a finite sequence of symbols from some alphabet.

Ex: $\Sigma = \{0,1\}$.

1) 0 is a string of length '1'.

2) 01 is a string of length '2'.

3) 10 is a string of length '2'

4) 001 is a string of length '3'.

1011, 111, 11000 are also the strings over $\{0,1\}$.

Note: The empty string (or) null string is denoted by ϵ (Epsilon).

Length of a string: The length of a string is the number of symbols in that string. If w is a string, then its length is denoted by $|w|$.

Ex: If $w = 1011$,

then length of the string $|w| = 4$.

powers of alphabet (Σ):

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

Σ^0 = Set of all strings over $\{0,1\}$ of length '0'.

$= \{\epsilon\}$ → Epsilon.

Σ^1 = set of all strings over $\{0,1\}$ of length '1'.

$= \{0,1\}$

Σ^2 = set of all strings over $\{0,1\}$ of length '2'.

$= \{00, 01, 10, 11\}$

Σ^3 = set of all strings over $\{0,1\}$ of length '3'.

$= \{000, 001, 010, 011, 100, 101, 110, 111\}$.

⋮

Σ^n = set of all strings over $\{0,1\}$ of length 'n'.

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

$= \{\epsilon\} \cup \{0,1\} \cup \{00, 01, 11, 10\} \cup \ldots\ldots$

Σ^* (Kleen closure) is an infinite.

Σ^* is a set of all strings over $\{0,1\}$ of all lengths.

Language:- A language 'L' is a collection of strings over an input alphabet 'Σ'.

Ex: $\Sigma = \{0,1\}$.

L_1 = set of all strings of length '2'.

$= \{00, 01, 10, 11\}$ → It is a finite set.

so, L_1 is a finite language.

L_2 = set of all strings where each string starts with 0.

$= \{0, 00, 01, 001, 010, 011, \ldots\}$

L_2 is an infinite language.

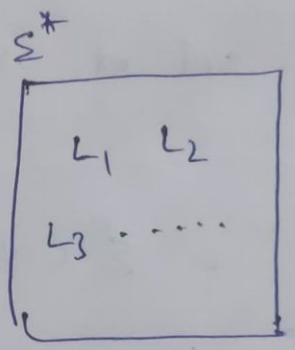
Note:- Every language is a subset of Σ^*.

$L_1 \subseteq \Sigma^*$

$L_2 \subseteq \Sigma^*$

$L_3 \subseteq \Sigma^*$

$\vdots$



Σ^* is a universal set.

Note:- $\phi \neq \{\epsilon\}$

ϕ → contains no string

$\{\epsilon\}$ → contains one string (Null string).

ϕ → Empty language over any alphabet.

$\{\epsilon\}$ → Language consisting of only empty (or) null strings over an alphabet.

positive closure :- If Σ is an alphabet then positive closure of Σ is denoted by Σ^+ and defined as:

$$\Sigma^+ = \Sigma^* - \{\epsilon\}$$

set of all strings over Σ excluding empty string ϵ.

For a C program,

$\Sigma = \{a, b, \cdots z, A, B, \cdots Z, 0, 1, \cdots 9, +, *, \cdots\}$.

```
void main()
{
    int a,b;
    :
}
```

This is a program in 'C' language.

But in FLAT, it is a string

program = string

C - programming language = set of all valid programs.

$= \{P_1, P_2, P_3, \cdots\}$ → Infinite.

In FLAT,

Language = set of all strings over 'Σ'.

(4)

Now given a language(L) and string(w), find whether string(w) is present in the language (L) (or) not?

↳ If a language is finite, we can scan all the strings and we can find string(w) is present (or) not.

↳ If a language is infinite, it is very difficult to find whether string(w) is present (or) not.

Ex:- $\Sigma = \{0,1\}$

$L_1 = \{00, 01, 10, 11\}$.

string(w) = 000.

L_1 is a finite language, so we can compare string(w) with all strings in language (L_1).

so, '000' is not present in L_1.

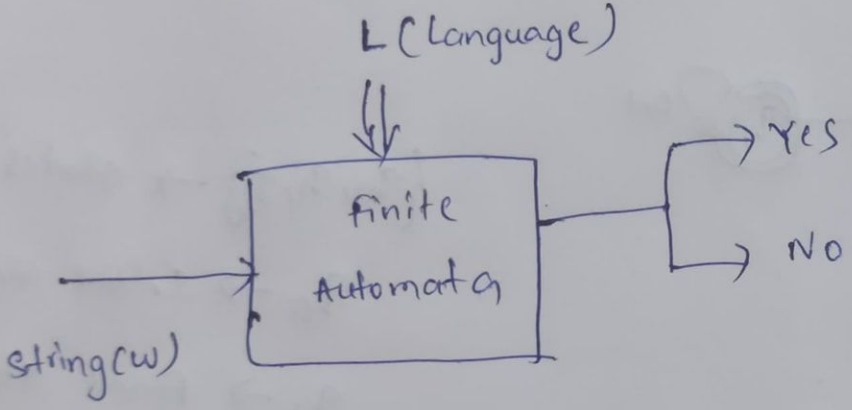
L_2 = set of all strings starts with 0.

$= \{0, 00, 001, 010, \cdots\}$

L_2 is a infinite language.

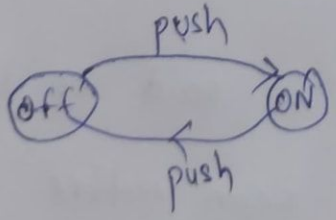
string(w) = 1010

If given language L (finite (or) infinite), come up with a finite representation (which is a finite automata) so that if we give any string 'w' to that finite automata, it should say whether string 's' is present in the language (or) not.

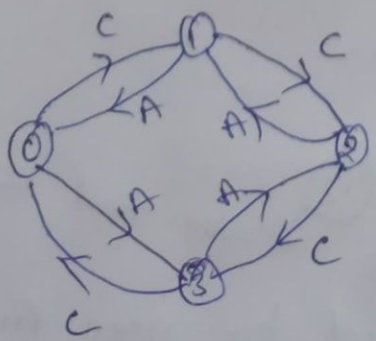


Finite Automata :- A finite automata consists of a finite memory called input tape, a finite non empty set of states, an input alphabet, a read-only head, a transition function which defines the change of configuration.

Ex :- 1) An electric switch



2) fan regulator.



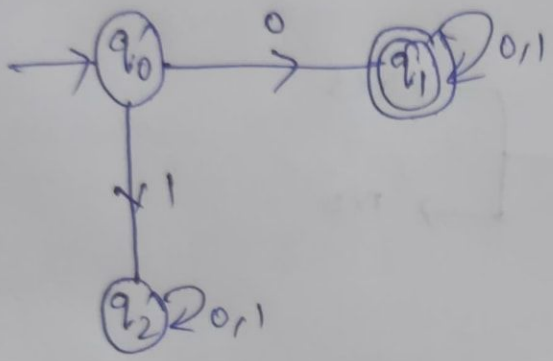
C → clockwise

A → Anti clockwise.

Ex: construct a finite automata which accepts set of all strings over $\{0,1\}$ which starts with '0'.

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

$$L = \{0, 00, 01, 001, 010, \cdots\}.$$

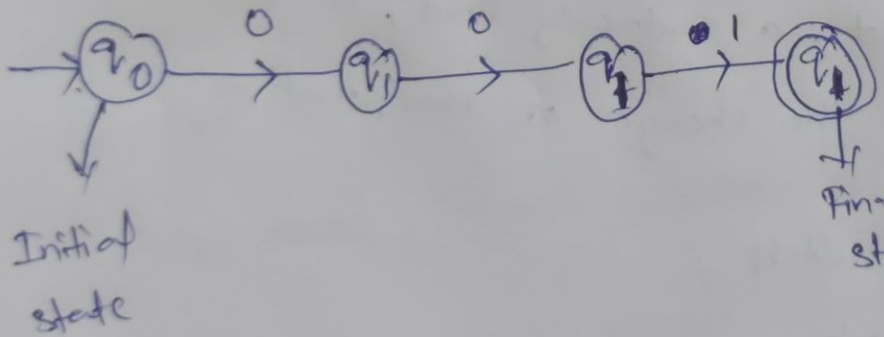


$\{q_0, q_1, q_2\}$ → states

q_0 → Initial state.

q_1 → final state.

Ex: string (w) = 001



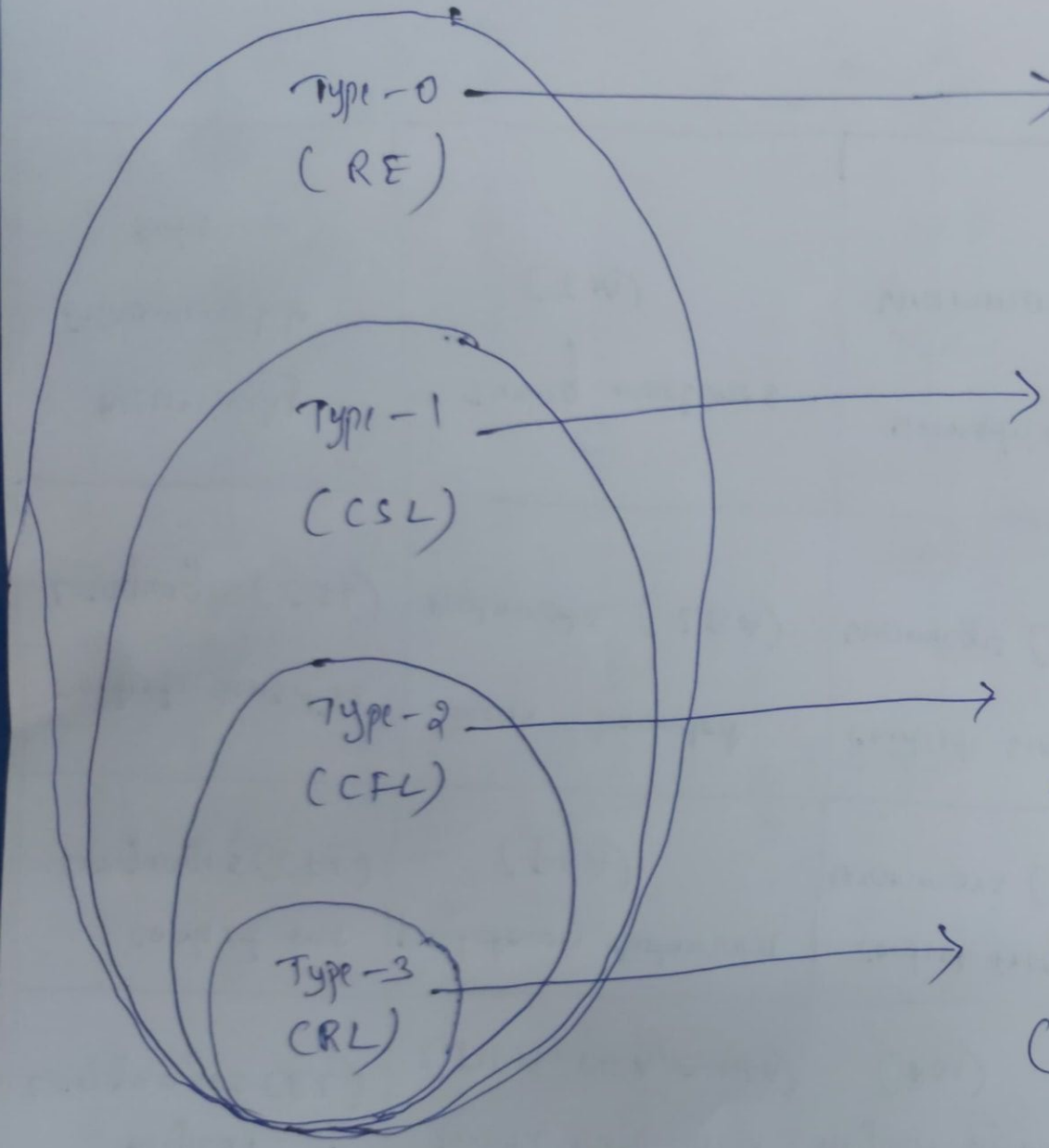
string (w) is said to be accepted by finite automata (FA) if we scan entire string from (initial) state and if we reach one of the final state.

Ex: string (w) = 110.

Hence, 110 is not accepted by given finite automata.

Chomsky hierarchy of languages :-

Languages.	Automata	Grammars	Type of grammars	Examples.
Regular Languages (RL)	Finite Automata (DFA, NFA, ε-NFA)	Regular Grammar (RG)	Type 3	$0^n / n \geq 1$
context free Languages (CFL)	pushdown Automata (PDA)	context free Grammars (CFG)	Type 2	$0^n 1^n / n \geq 1$
context sensitive Languages (CSL)	Linear bounded Automata (LBA)	context sensitive Grammars (CSG)	Type 1	$0^n 1^n 2^n / n \geq 1$
Recursively Enumerable sets	Turing machines (TM)	Unrestricted Grammars	Type 0	$0^p / p$ is a prime number.

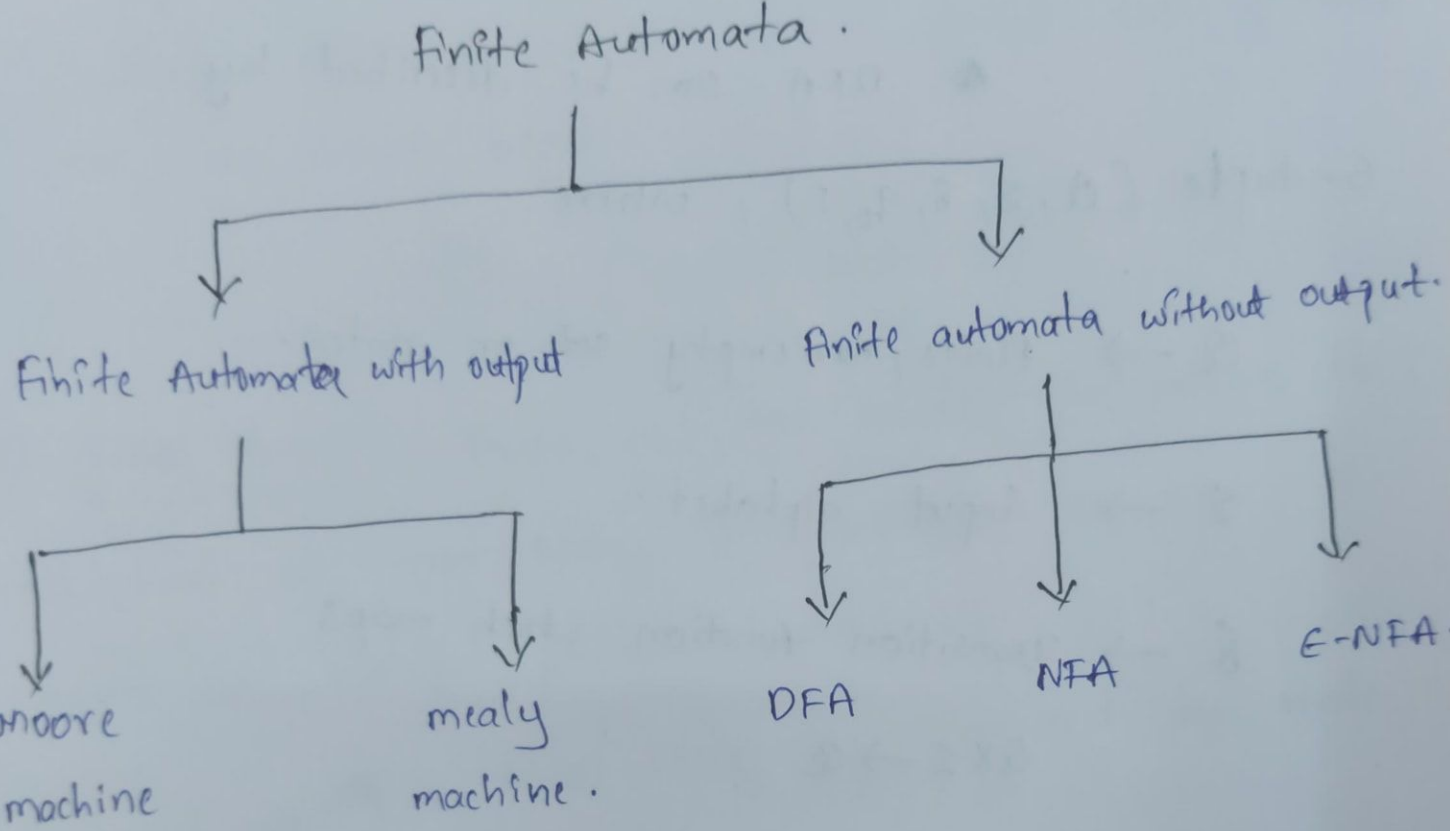


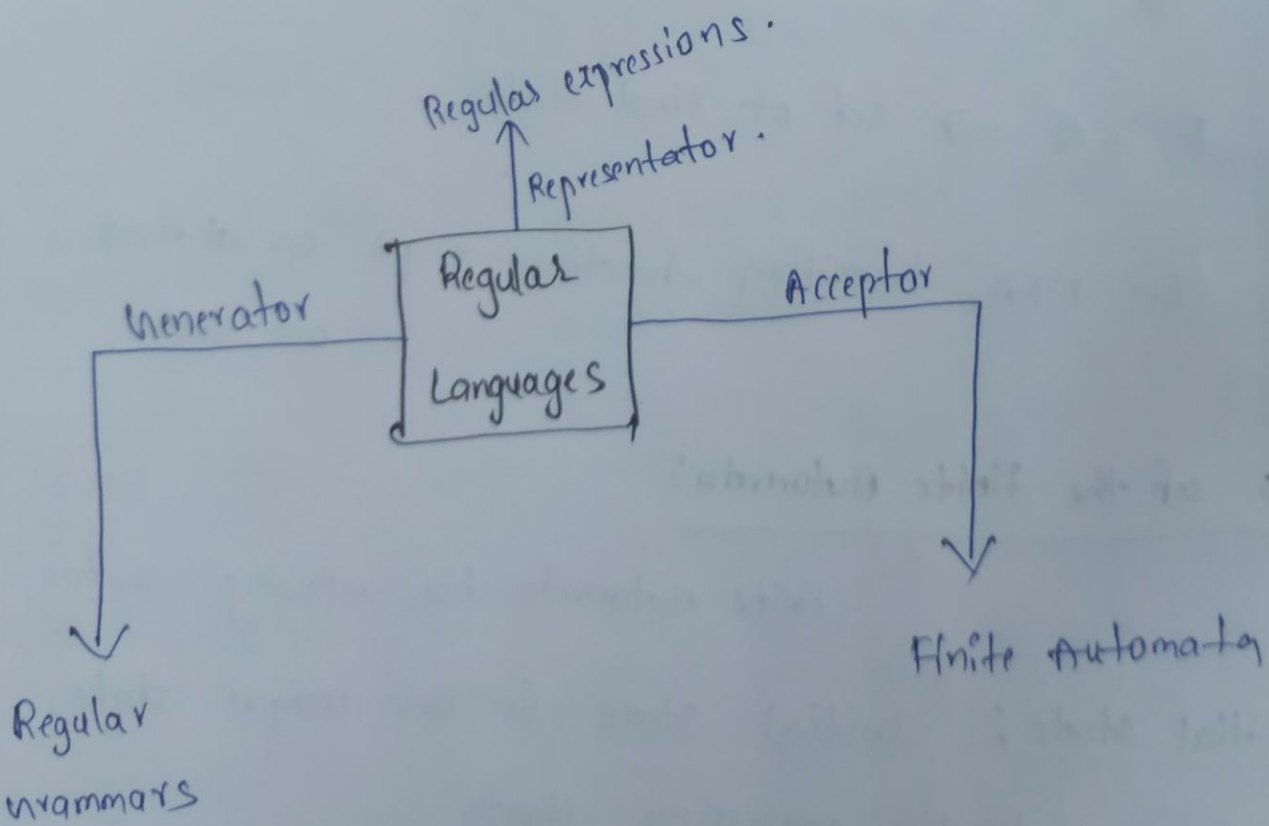
unrestricted grammar
(Accepted by Turing machine)

context-sensitive grammar
(Accepted by Linear bound automata)

context-free grammar
(Accepted by push down automata)

Regular grammar
(Accepted by Finite automata).


Finite Automata.
Finite Automata with output
Finite automata without output.
moore machine
mealy machine.
DFA
NFA
E-NFA


Regulas expressions.
Representator.
Regular Languages
Generator
Acceptor
Regular Grammars
Finite Automata

Deterministic finite Automata (DFA):-

DFA can be described by 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

$Q \rightarrow$ finite, nonempty set of states.

$\Sigma \rightarrow$ input alphabet.

$\delta \rightarrow$ Transition function which maps

$$Q \times \Sigma \rightarrow Q$$

$q_0 \in Q \rightarrow$ Initial state.

$F \subseteq Q \rightarrow$ set of final states.

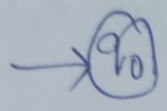
Note:- In DFA, transition function maps on at most one state.

States of the Finite Automata:-

Finite Automata has following states:

1) Initial state: Initial state is an unique state, from this state the processing starts.

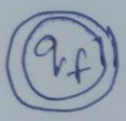
The initial state is represented by a state within a circle and an arrow entering into circle as shown below:



(Initial state q_0)

2) Final States:- These are the states in which if execution of input string is ended, then execution is known as succesful otherwise unsuccessful.

Final state is represented by state within double circles:



(Final state q_f).

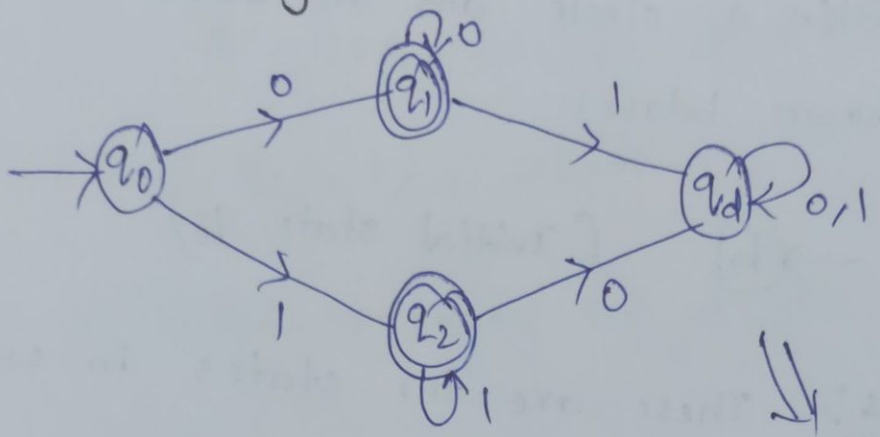
3) Non-final states:- All states except final states are known as non-final states.

4) Dead state:- After reaching these states, FA sits in idle situation.

Dead state is represented by:

(dead state).

Consider the following DFA,



This is a state transition diagram.

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

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

$q_0 = \{q_0\}$

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

q_d is a dead state.

$\delta : Q \times \Sigma \rightarrow Q$.

$\delta(q_0, 0) = q_1$

$\delta(q_0, 1) = q_2$

$\delta(q_1, 0) = q_1$

$\delta(q_1, 1) = q_d$

$\delta(q_2, 0) = q_d$

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

$\delta(q_d, 0) = q_d$

$\delta(q_d, 1) = q_d$

State transition table:

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

Acceptability of a String by DFA :-

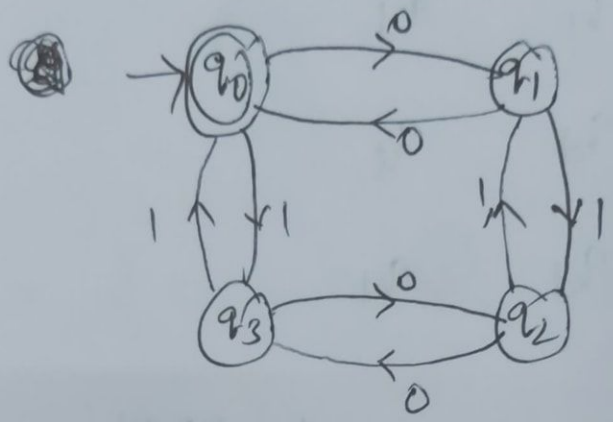
Let a DFA $= (Q, \Sigma, \delta, q_0, F)$ and an input string(w). The string w is accepted by DFA if and only if $\delta(q_0, w) = q_f$, where $q_f \in F$.

when string(w) is accepted by DFA, then the execution of string(w) ends in a final state and this execution is known as succesful otherwise unsuccessful.

Example:

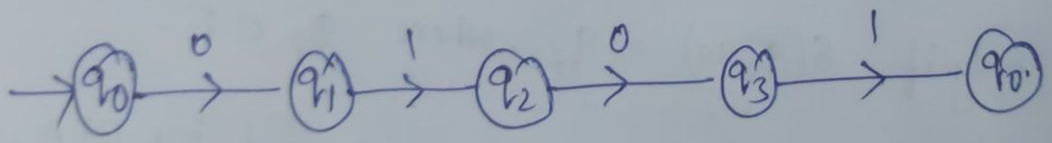
Consider the following DFA. Check the acceptability of following strings:

a) 0101 b) 0111 c) 001.



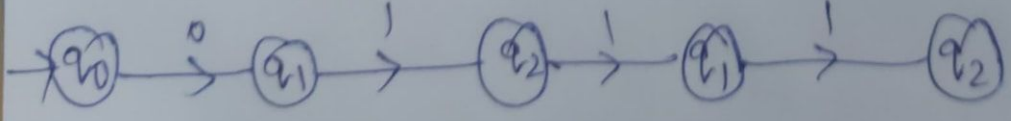
Solution:-

a) Input string(w) = 0101.



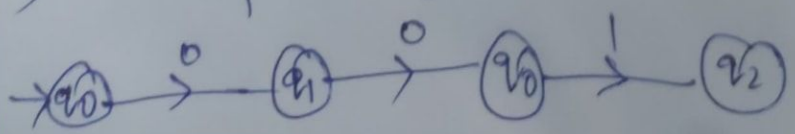
Execution ends in final state q_0, hence string 0101 is accepted.

b) Input string(w) = 0111



Execution ends in non-final state q_2, hence string 0111 is not accepted.

c) Input string(w) = 001

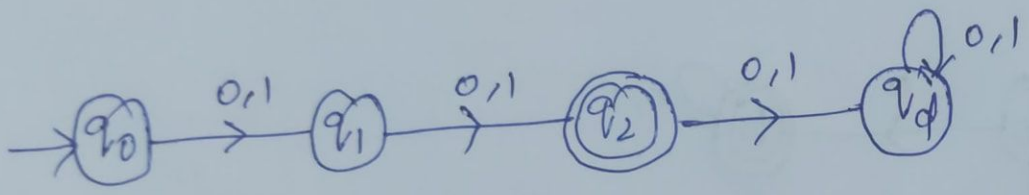


Execution ends in non-final state q_2, string 001 is not accepted.

1) Construct a DFA that accepts set of all strings over $\{0,1\}$ of length exactly '2'.

Solution:- $\Sigma = \{0,1\}$

$L = \{00, 01, 10, 11\}$.



state transition diagram.

state transition table:-

	0	1
$\rightarrow q_0$	q_1	q_1
q_1	q_2	q_2
$*q_2$	q_d	q_d
q_d	q_d	q_d.

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

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

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

q_0 is the initial state.

$F = \{q_2\}$

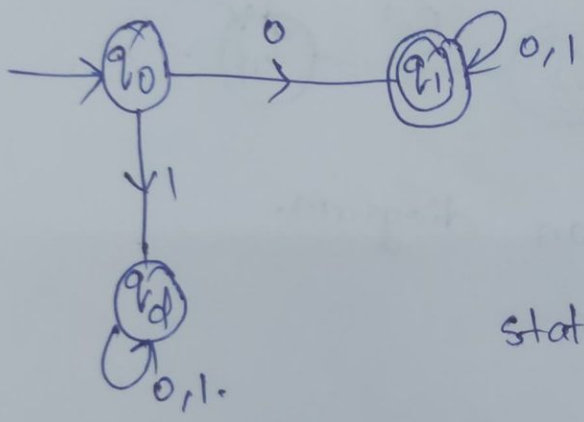
δ is shown using the transition table.

2) Construct a DFA which accepts set of all strings over $\{0,1\}$ where each string starts with '0'.

Solution:

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

$$L = \{0, 00, 01, 001, 010, 0110, \ldots\}.$$



state transition diagram.

State transition table:

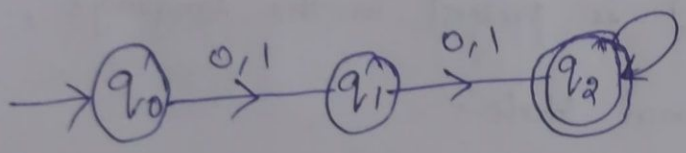
	0	1
$\rightarrow q_0$	q_1	q_d
$*q_1$	q_1	q_1
q_d	q_d	q_d

3) Construct a DFA which accepts set of all strings over {0,1} where length of the string is atleast '2'.

Solution:- $\Sigma = \{0,1\}$

$L = \{00, 01, 10, 11, 001, 010, 0000, 1011, \ldots\ldots\}$.

$|w| \geq 2$. (w is a string).



State transition table:

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

4) construct a DFA which accepts set of all strings over {0,1} where length of the string is atmost '2'.

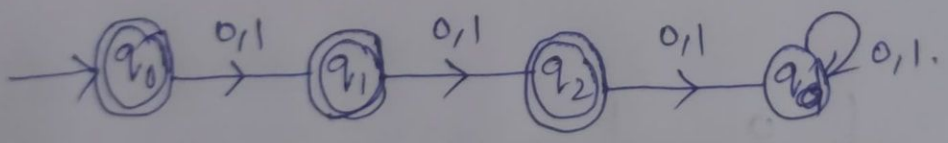
Solution :-

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

$|w| \leq 2$.

$L = \{\epsilon, 0, 1, 00, 01, 10, 11\}$.

Note: If ϵ (Epsilon) is present in the language, make initial state as final state.



State transition table :-

	0	1
$\rightarrow^{*} q_0$	q_1	q_1
$^{*} q_1$	q_2	q_2
$^{*} q_2$	q_d	q_d
q_d	q_d	q_d

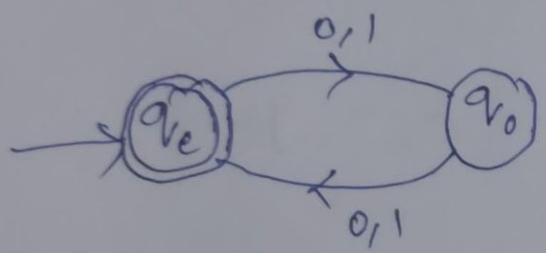
5) Construct a DFA that accepts set of all even length strings over $\{0,1\}$.

Solution:-

$\Sigma = \{0,1\}$ $\quad w \in \{0,1\}^*$

$|w| \bmod 2 = 0.$

$L = \{\epsilon, 00, 01, 10, 11, 0000, 0100, 1100, \ldots\}.$



q_e - even

q_o - odd.

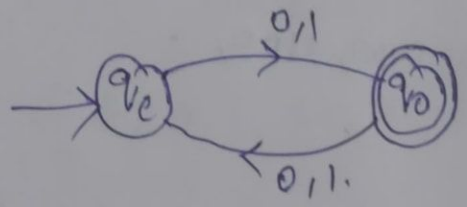
State transition table:-

	0	1
$\rightarrow *q_e$	q_o	q_o
q_o	q_e	q_e

6) Construct a DFA that accepts set of all odd length strings over $\{0,1\}$.

Solution:- $\Sigma = \{0,1\}$. $w \in \{0,1\}^*$

$|w| \bmod 2 = 1$.



$L = \{0, 1, 000, 010, 110, \ldots\}$.

State transition table:-

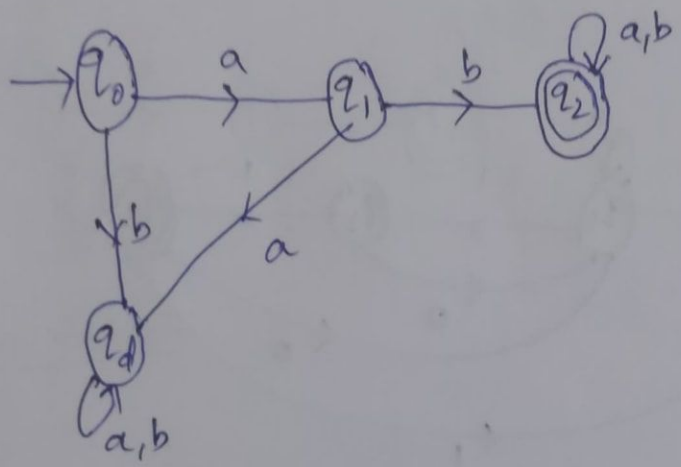
	0	1
$\rightarrow q_e$	q_o	q_o
$*q_o$	q_e	q_e

7) Construct a DFA that accepts set of all strings over {a,b} where all strings should start with ab.

Solution:-

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

$L = \{ab, aba, abaa, abb, abba, \cdots\}$



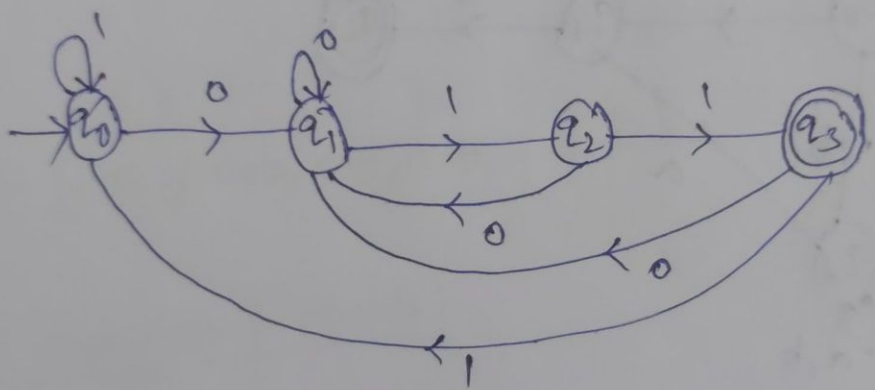
State transition table:-

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

8) Design a DFA to accept set of all strings over $\{0,1\}$ ending with the string 011.

Solution:- $\Sigma = \{0,1\}$

$L = \{011, 1011, 0011, 11011, 01011, \ldots\}$



State transition table:-

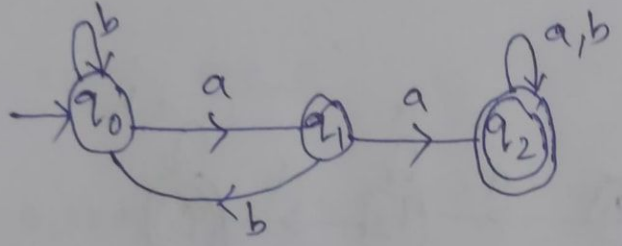
	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_3
$*q_3$	q_1	q_0

9) obtain a DFA to accept of all strings over $\{a,b\}$ having a substring aa.

Solution:-

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

$$L = \{aa, aab, baa, aaa, abaa, \ldots\}$$



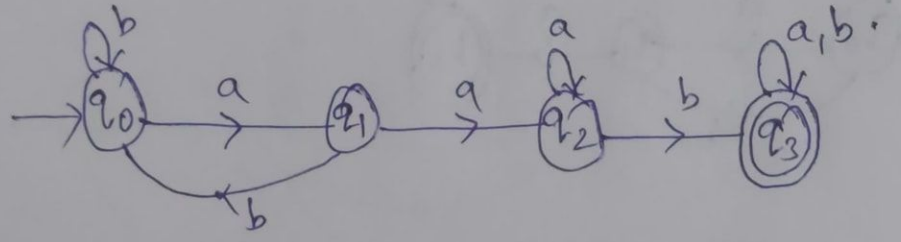
State transition table:

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

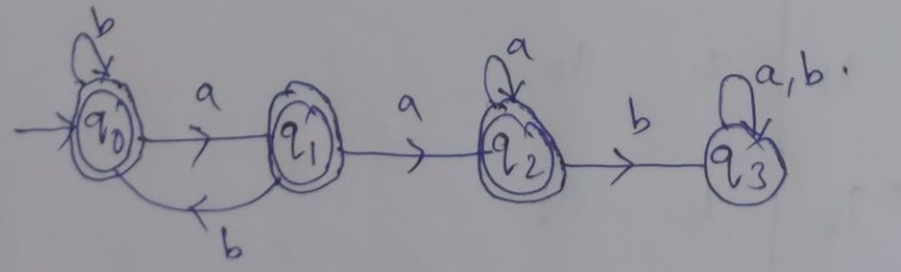
10) obtain a DFA to accept set of all strings over $\{a,b\}$ except those containing the substring aab.

Solution:- $\Sigma = \{a,b\}$.

First construct the DFA to accept all strings containing aab as a substring.



Now change the final states to non-final states and non-final states to final states.



⇓

This DFA will accept all strings except those containing the sub-string aab.

State transition table:-

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

11) Obtain a DFA to accept all strings over $\{0,1\}$

a) having exactly one 0

b) having atleast one 0

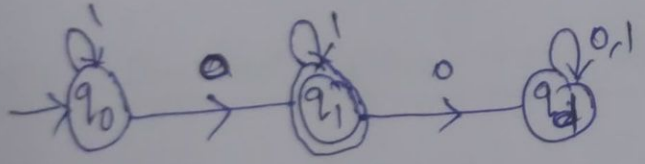
c) having not more than three 0's.

Solution:-

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

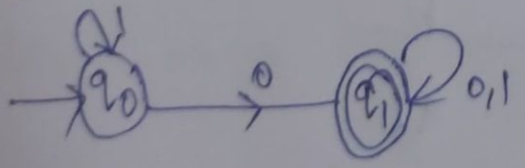
a) having exactly one 0.

$L = \{0, 100, 110, 01, 011, 1011, 11011, \cdots\}$.



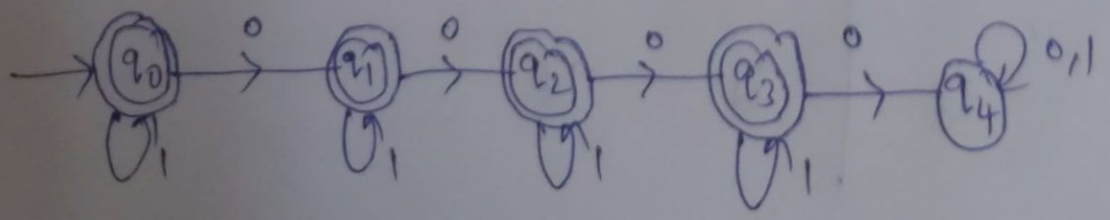
b) having atleast one 0

$L = \{0, 001, 1000, 1100, 011000, \cdots\}$



c) having not more than three 0s

$L = \{\epsilon, 0, 00, 000, 010, 10010, 01011, \cdots\}$



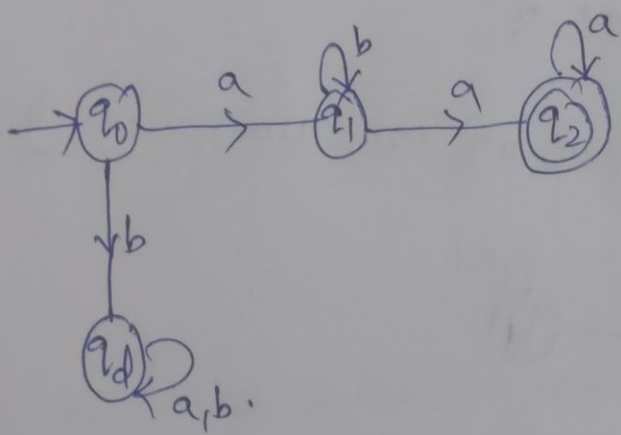
12) Obtain a DFA to accept the language

$L = \{ awa \mid w \in \{a,b\}^* \}$.

Solution:

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

$L = \{aa, aaa, aba, abaa, aabba, \cdots\}$.



State transition table:

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

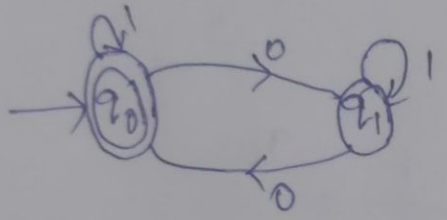
13) Construct a DFA that accepts set of all strings over {0,1} having even number of 0s and 1's.

Solution:

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

$L = \{\epsilon, 00, 11, 0110, 1111, 0000, 011101, \cdots\}$

First construct DFA for even no. of 0's



Now Construct DFA for even no. of 1's

Now combine above two DFA's using cross product method.

$= \{q_0, q_1\} \times \{q_2, q_3\}$

$= \{q_0q_2, q_0q_3, q_1q_2, q_1q_3\}$

Now new DFA contains four states.

For the above two DFA's, q_0 & q_2 are final states.

In the required DFA, q_0q_2 is a final state.

$q_0 \xrightarrow{0} q_1$, $q_2 \to q_2$

$q_0 \xrightarrow{1} q_0$, $q_2 \to q_3$

$q_1 \xrightarrow{0} q_0$, $q_2 \to q_2$

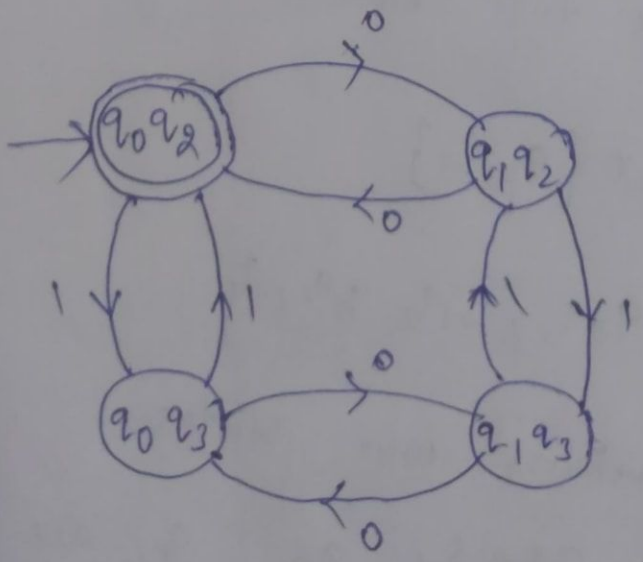
$q_1 \xrightarrow{1} q_1$, $q_2 \to q_3$

$q_0 \xrightarrow{0} q_1$, $q_3 \to q_3$

$q_0 \xrightarrow{1} q_0$, $q_3 \to q_2$.

$q_1 \xrightarrow{0} q_0$, $q_3 \to q_3$

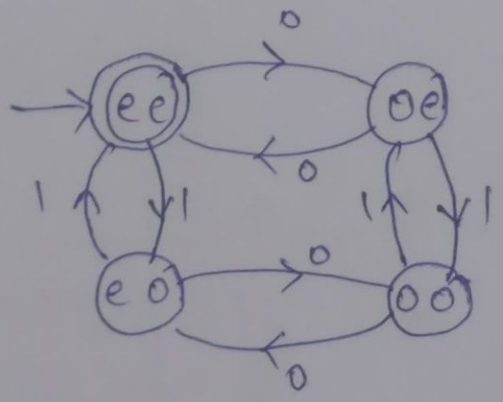
$q_1 \xrightarrow{1} q_1$, $q_3 \to q_2$.



Another method:-

$L = \{ \epsilon, 00, 11, 0011, 1010, \cdots \}$

$n_0(w)$	$n_1(w)$	
e	e	$\{ \epsilon, 00, 11, \cdots \}$
e	o	$\{ 001, \cdots \}$
o	e	$\{ 00011, \cdots \}$
o	o	$\{ 01, \cdots \}$



e – even
o – odd.

Note:- 1) DFA to accept even number of 0's and odd number of 1's, make eo as the final stede instead of ee.

2) DFA to accept odd number of 0's and even number of 1's, make oe as the final state instead of ee.

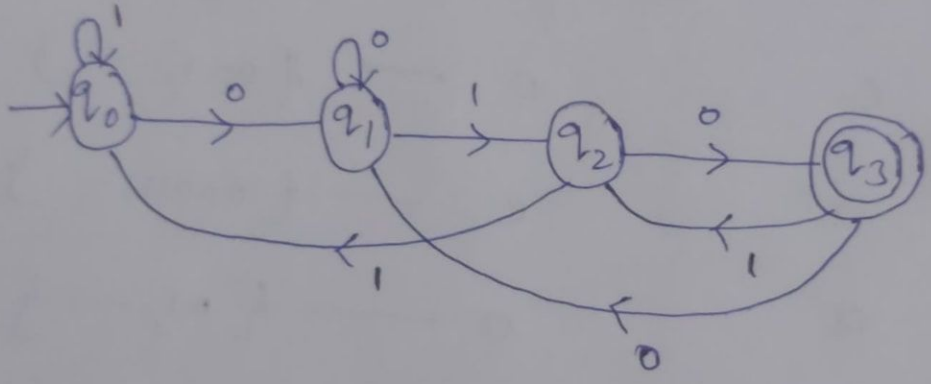
3) DFA to accept odd number of 0's and odd number of 1's, make oo as the final state instead of ee.

14) Design a DFA to accept set of all strings over $\{0,1\}$ ending with 010.

Solution:-

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

$$L = \{010, 11010, 01010, \ldots\}.$$



State transition table:-

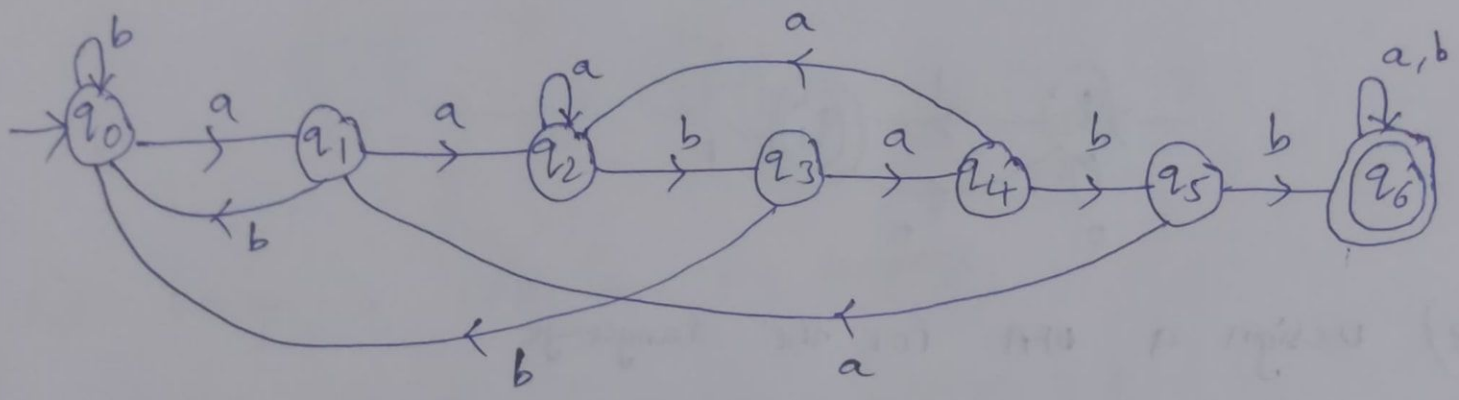
	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_1	q_2
q_2	q_3	q_0
$*q_3$	q_1	q_2

15) Give DFA that accepts any string with aababb as a substring over {a, b}.

Solution:-

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

$L = \{aababb, baababba, abaababbbab, \ldots\}$



Transition table:

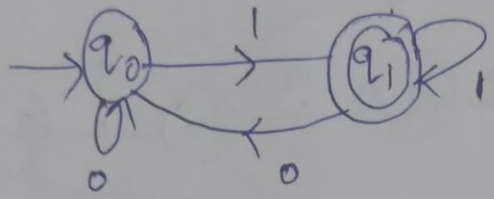
	a	b
$\rightarrow q_0$	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_3
q_3	q_4	q_0
q_4	q_2	q_5
q_5	q_1	q_6
$*q_6$	q_6	q_6

16) Design a DFA for the language

$L = \{ w / w \in \{0,1\}^* \text{ and } w \text{ ends with } 1 \}$.

Solution:- $\Sigma = \{0,1\}$

$L = \{ 1, 01, 11, 001, 101, \cdots \}$

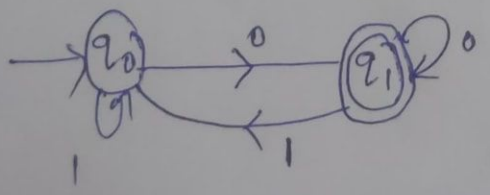


17) Design a DFA for the language

$L = \{ w / w \in \{0,1\}^* \text{ and } w \text{ ends with } 0 \}$.

Solution:- $\Sigma = \{0,1\}$

$L = \{ 0, 00, 10, 110, 010, \cdots \}$

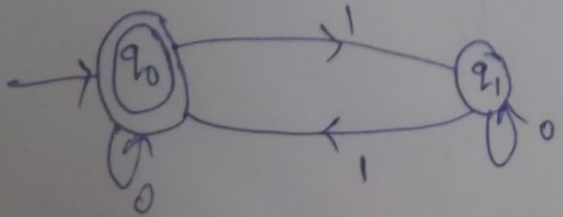


18) Design a DFA for the language

$L = \{ w / w \in \{0,1\}^* \text{ and } w \text{ has even no. of 1's} \}$.

Solution:- $\Sigma = \{0,1\}$

$L = \{ \epsilon, 0, 00, 011, 11, 101, \cdots \}$

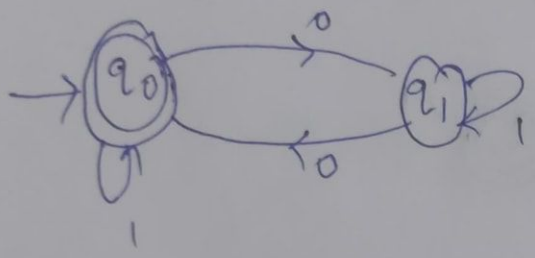


19) Design a DFA for the language

$L = \{ w / w \in \{0,1\}^* \text{ and } w \text{ has even no. of 0's} \}$

Solution:- $\Sigma = \{0,1\}$

$L = \{ \epsilon, 1, 00, 100, 0000, 1010, \cdots \}$



20) Design a DFA for the language

$L = \{ w / w \in \{0,1\}^* \text{ and } w \text{ has a substring } 01 \}$.

Solution:- $\Sigma = \{0,1\}$

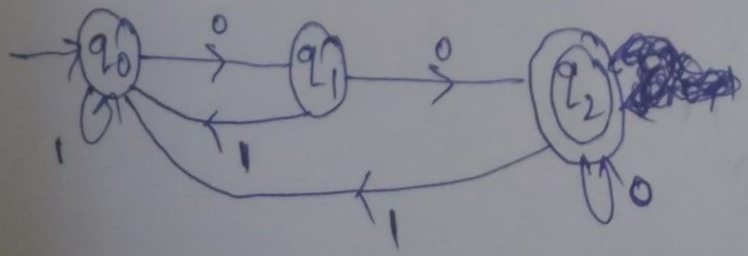
$L = \{ 01, 101, 1011, 00100, \cdots \}$

21) Design a DFA for the language

$L = \{ w / w \in \{0,1\}^* \text{ and } w \text{ is ending with } 00 \}$.

Solution:- $\Sigma = \{0,1\}$

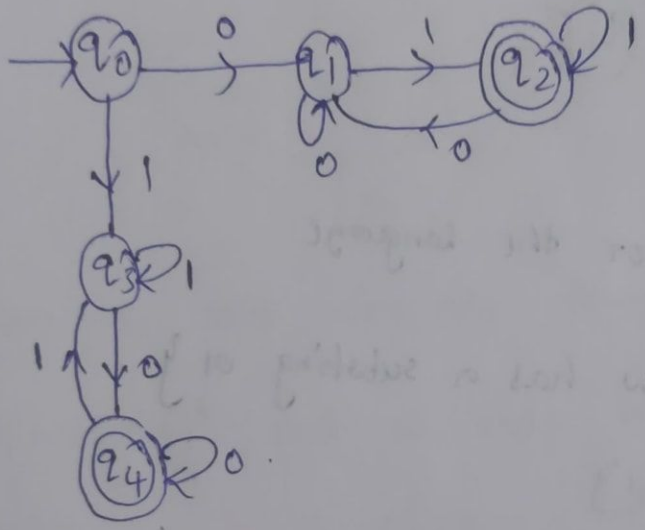
$L = \{ 00, 100, 000, 1100, 0100, \cdots \}$



22) Construct a DFA which accepts set of all strings over {0,1} where strings starts and ends with different symbol.

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

$L = \{01, 10, 001, 100, 110, 011, \ldots\}$

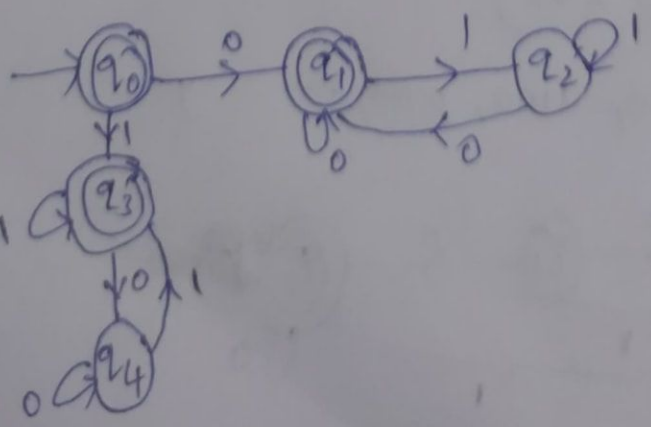


23) Construct a DFA which accepts set of all strings over {0,1} where strings starts and ends with same symbol.

Solution:- $\Sigma = \{0,1\}$

$L = \{\in, 00, 11, 0, 1, 0110, 0010, 1011, \ldots\}$

Take the previous question DFA and make non-final states to final states, final states to non-final states.



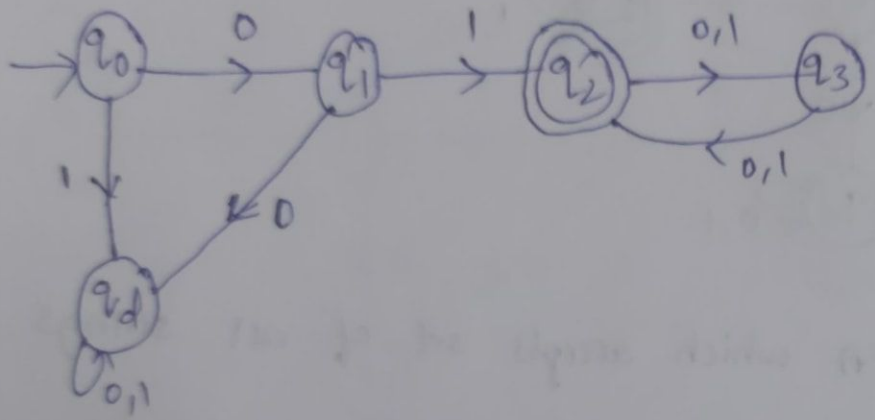
24) Design a DFA to accept the language

$L = \{ w / w \in \{0,1\}^* \text{ and } w \text{ is of even length and begins with } 01 \}$

Solution:-

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

$L = \{ 01, 0110, 0100, 011100, \cdots \}$

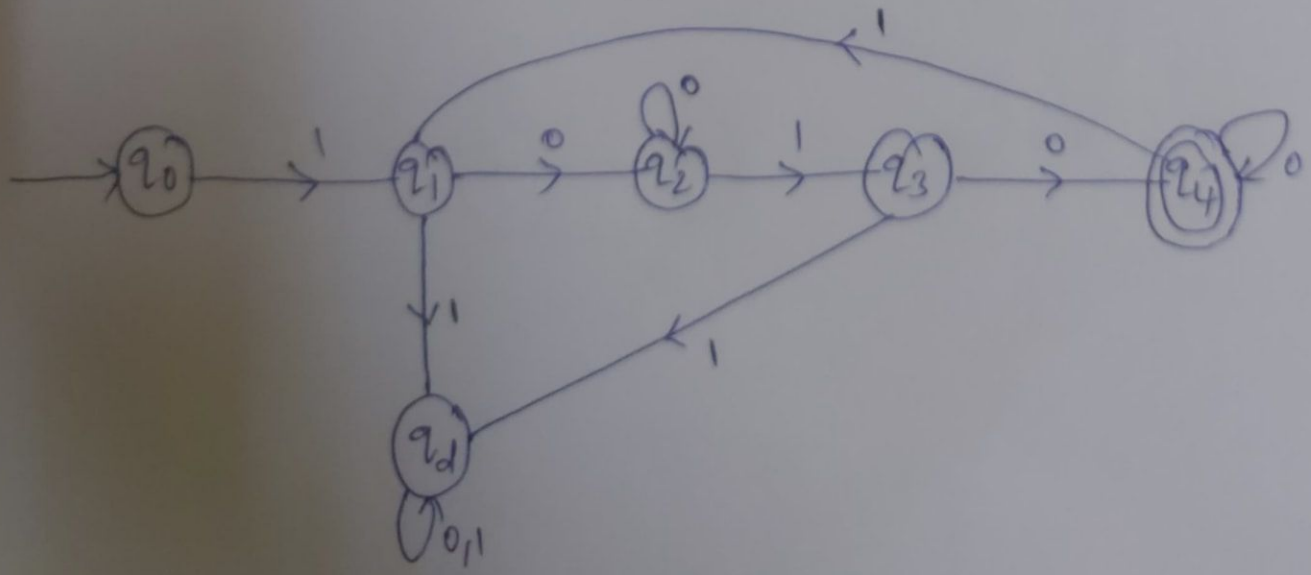


25) Design a DFA to accept set of all strings over $\{0,1\}$ having even number of 1's and each 1 is followed by atleast one 0.

Solution:

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

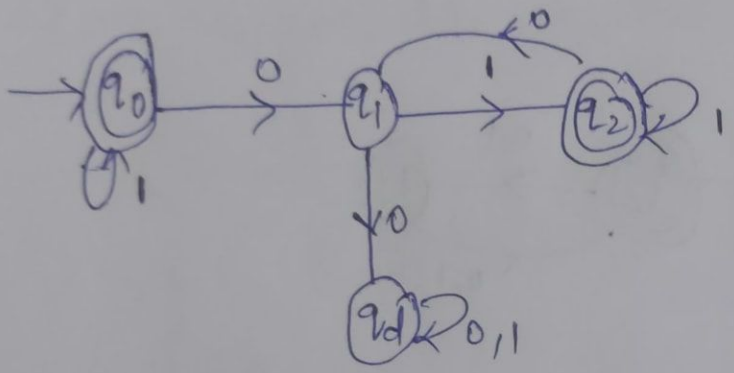
$L = \{ 1010, 01010, 0100010, \cdots \}$



26) construct a DFA which accepts set of all strings over {0,1} in which every 0 is followed by 1.

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

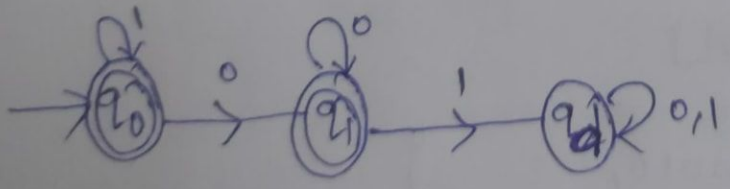
$L = \{ \epsilon, 01, 0101, \cdots, 1, 11, 111, \cdots \}$.



27) Construct a DFA which accepts set of all strings over {0,1} in which every 0 should never followed by 1.

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

$L = \{ \epsilon, 0, 00, 000, \cdots, 1, 11, 111, 10, 110, \cdots \}$.

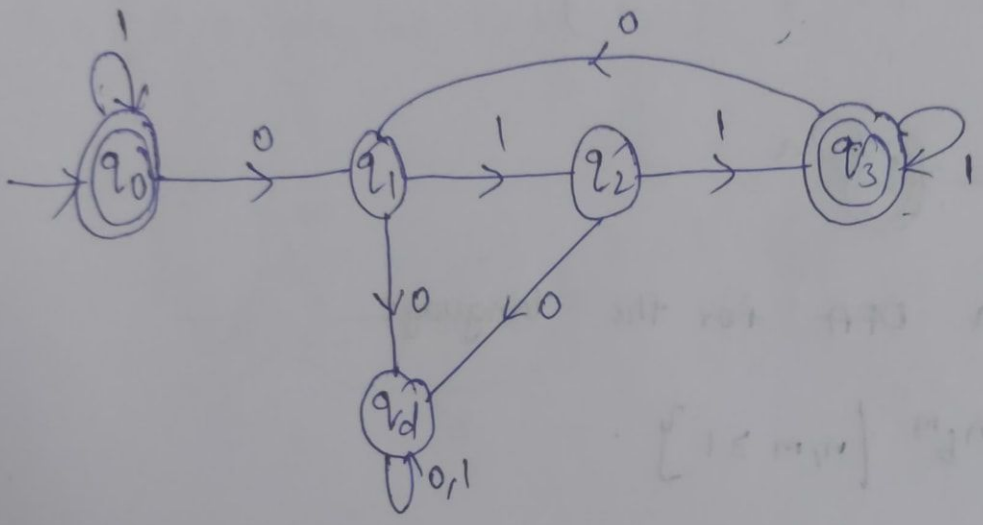


28) Construct a DFA which accepts set of all strings over {0,1} in which every 0 is followed by 11.

Solution:

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

$L = \{ \epsilon, 011, 011011, 01111, 1, 11, 111, \cdots \}$

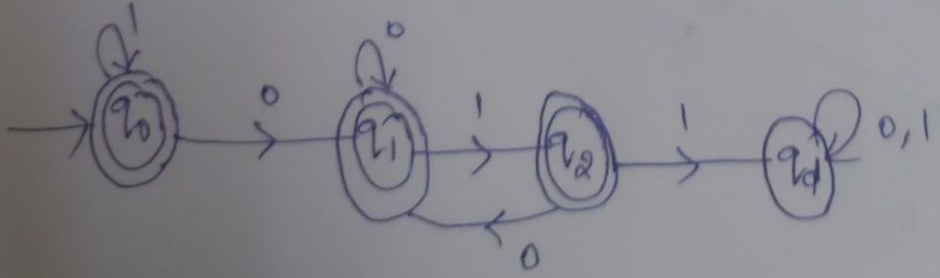


29) Construct a DFA which accepts set of all strings over {0,1} in which every 0 should never be followed by 11.

Solution:-

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

$L = \{ \epsilon, 0, 00, 01, 10, 101, 1, 11, \cdots \}$



30) Construct a DFA for the language

$$L = \{ a^n \mid n \geq 0 \}.$$

Solution:

$$\Sigma = \{a\}$$

$$L = \{\epsilon, a, aa, aaa, \ldots\}$$

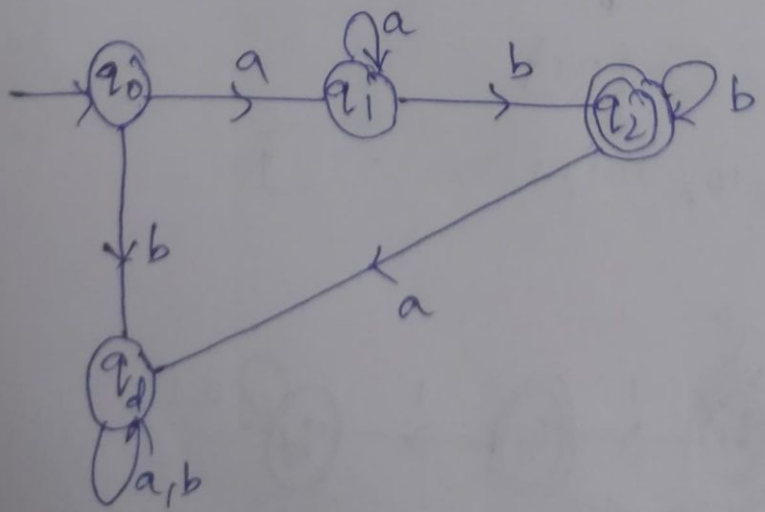
31) Construct a DFA for the language

$$L = \{ a^n b^m \mid n, m \geq 1 \}.$$

Solution:-

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

$$L = \{ab, aab, abb, aabb, aaabb, \ldots\}$$



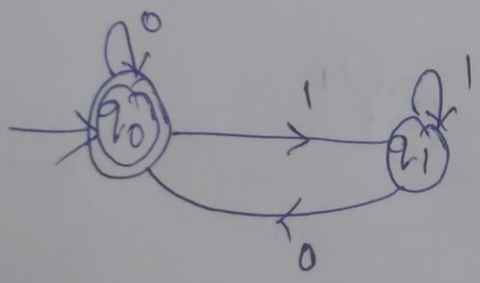
32) Construct a DFA which accepts set of all strings over $\{0,1\}$ which when interpreted as a binary number is divisible by 2.

Decimal	Binary
0	0
2	010
4	100
10	1010

Solution:-

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

$$L = \{\epsilon, 0, 010, 100, 1010, \cdots\}$$



State transition table:-

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

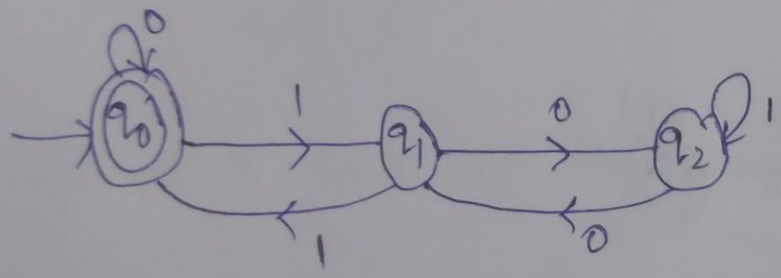
33) Construct a DFA which accepts set of all strings over $\{0,1\}$ which when interpreted as a binary number is divisible by 3.

Solution:

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

$L = \{0, 011, 110, 1001, \ldots\}$

Decimal	binary
0	0
3	011
6	110
9	1001
.	.



State transition table:

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

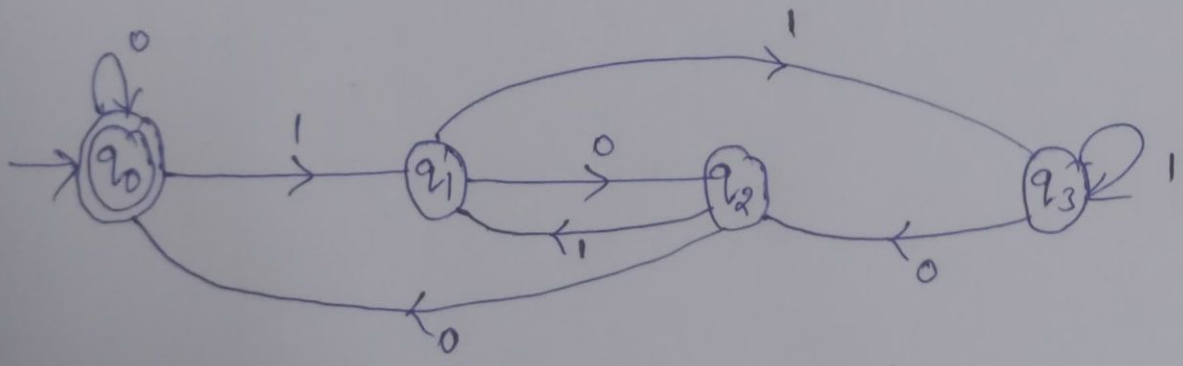
34) Construct a DFA which accepts set of all strings over $\{0,1\}$ which when interpreted as binary numbers divisible by 4.

decimal	binary
0	0
4	0100
8	1000

Solution:

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

$L = \{0, 0100, 1000, \ldots\}$



State transition table:-

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

Non-deterministic finite automata (NFA):-

→ In NFA, a state may contain more than one transition from the same input.

$$q_0 \xrightarrow{0} q_1$$
$$\xrightarrow{0} q_2$$
$$\xrightarrow{0} q_3$$

→ It is not compulsory that a state has to consume all the input symbol present in Σ.

$$q_0 \xrightarrow{0} \phi$$

→ NFA is more easier to design than DFA.

→ Every DFA is a NFA.

NFA can be defined by 5 tuples,

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

Q → Finite set of states.

Σ → input alphabet.

q_0 → Initial state & $q_0 \in Q$.

F → Set of final states & $F \subseteq Q$.

δ : Transition function which maps.

$$Q \times \Sigma \rightarrow 2^Q$$

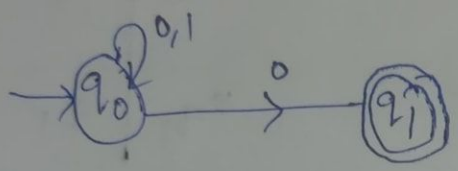
where 2^Q is the power set of Q.

1) Construct an NFA that accepts set of all strings over $\{0,1\}$ where all strings ends with 0.

Solution:-

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

$$L = \{0, 00, 10, 110, 000, \ldots\}$$



(Transition diagram).

Transition function (δ):

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

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

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

$$\delta(q_1, 1) = \phi.$$

State transition table:-

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

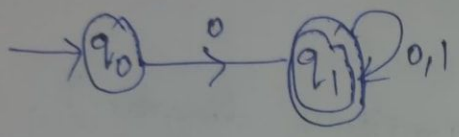
2) Construct a NFA for the language

$L = \{ w / w \in \{0,1\}^* \text{ and } w \text{ starts with } 0 \}$.

Solution:-

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

$L = \{0, 00, 01, 0011, 010, \ldots\}$



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

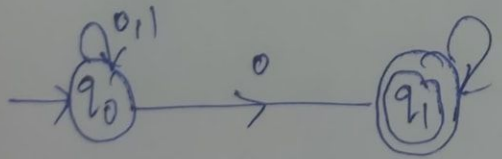
3) Construct a NFA for the language

$L = \{ w / w \in \{0,1\}^* \text{ and } w \text{ contains } 0 \text{ as a substring} \}$.

Solution:-

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

$L = \{0, 00, 10, 010, 101, 110, 011010, \ldots\}$



(Transition diagram)

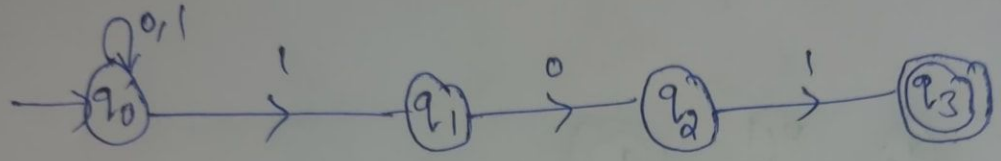
Transition table :-

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

3) Design an NFA that accepts all strings over {0,1} that end with 101.

Solution:- $\Sigma = \{0,1\}$

$L = \{101, 1101, 0101, 10101, 00101, \ldots\}$.



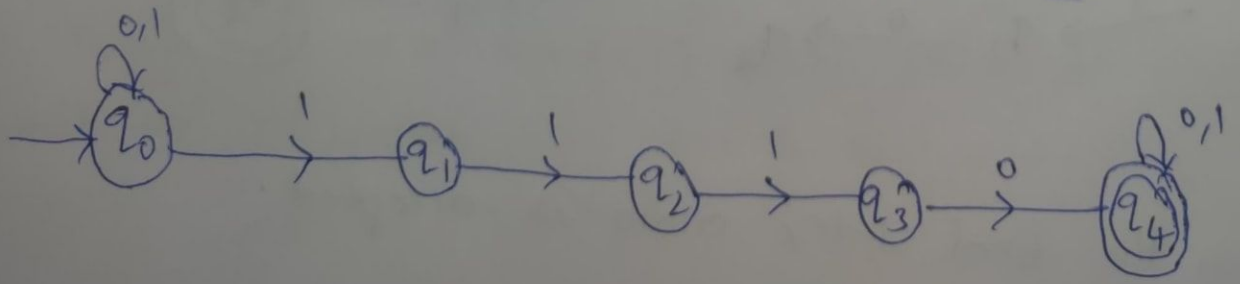
Transition table:-

	0	1
$\rightarrow q_0$	q_0	$\{q_0, q_1\}$
q_1	q_2	ϕ
q_2	ϕ	q_3
$*q_3$	ϕ	ϕ

4) Construct a NFA to accept set of all strings over {0,1} having a substring 1110.

Solution:- $\Sigma = \{0,1\}$

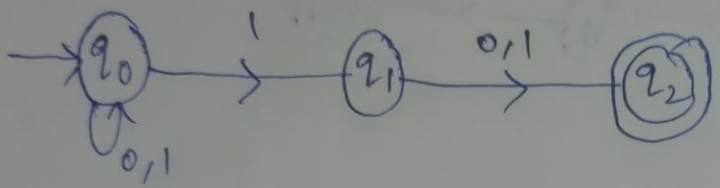
$L = \{1110, 01110, 10110, 01110101, \ldots\}$



5) Design an NFA which accepts set of all strings over {0,1} in which second symbol from RHS is always 1.

Solution:- $\Sigma = \{0,1\}$

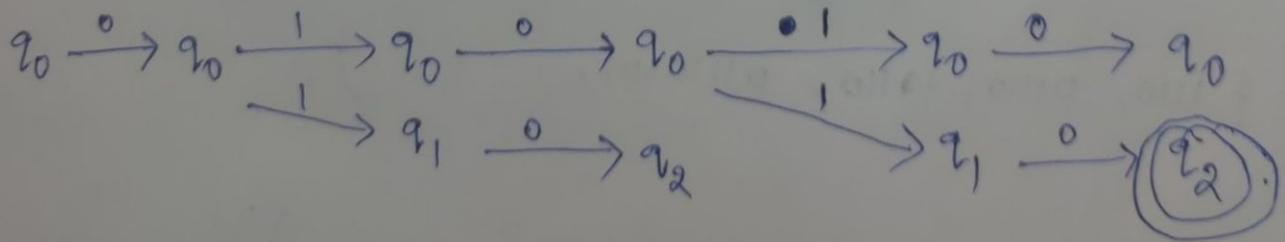
$L = \{10, 11, 010, 011, 1111, 1010, \ldots\}$



Transition table:-

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

Ex: Input string(w) = 01010.

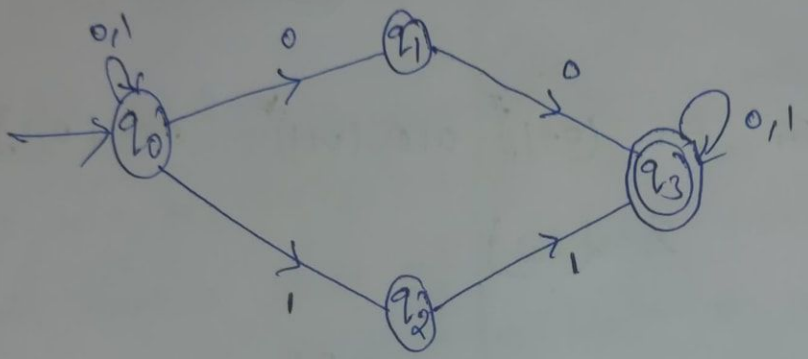


As q_2 is the final state, 01010 is accepted.

6) Design an NFA which accepts all strings over $\{0,1\}$ that contains 00 (or) 11 as substring.

Solution:- $\Sigma = \{0,1\}$

$L = \{00, 11, 100, 011, 000, 111, \ldots\}$

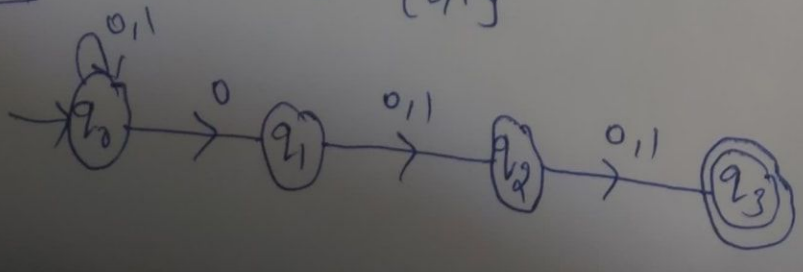


Transition table:-

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

7) Design an NFA over $\{0,1\}$ which accepts all strings in which the 3rd symbol from RHS is always 0.

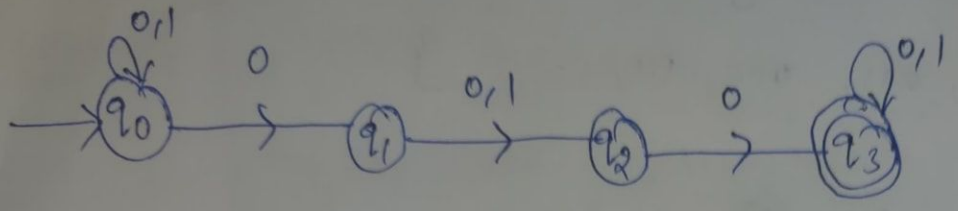
Solution:- $\Sigma = \{0,1\}$



8) Construct NFA for $(0+1)^* 0(0+1)0(0+1)^*$ over $\{0,1\}$. (5M)

Solution:

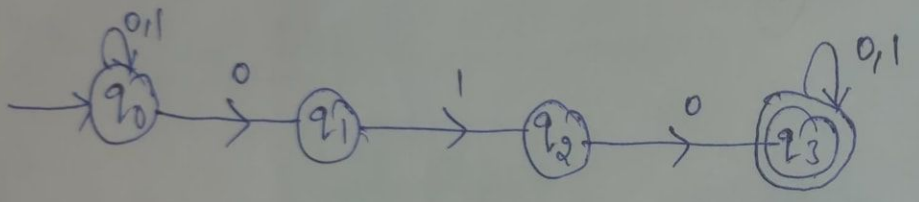
$\Sigma = \{0,1\}$



9) Construct NFA for $(0+1)^* 010 (0+1)^*$ over $\{0,1\}$.

Solution:-

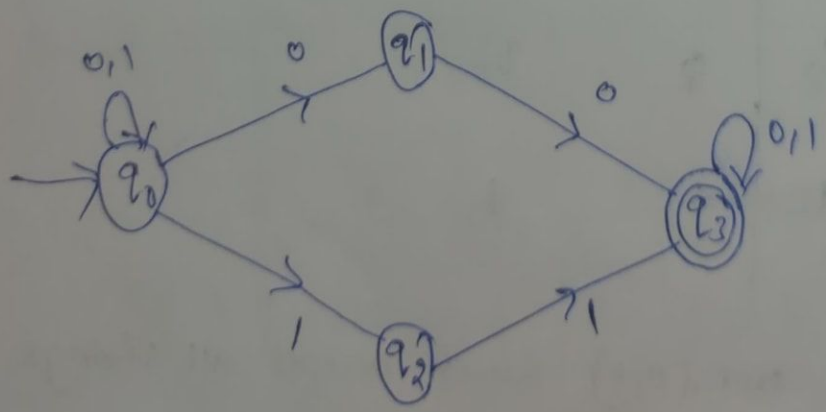
$\Sigma = \{0,1\}$



10) Construct NFA for $(0+1)^* (00+11) (0+1)^*$ over $\{0,1\}$.

Solution:-

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



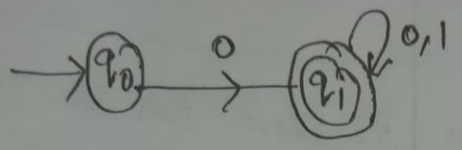
Differences between DFA and NFA:-

DFA	NFA
1) DFA and NFA are same except in the definition of δ. $\delta : Q \times \Sigma \to Q$	1) $\delta : Q \times \Sigma \to 2^Q$.
2) There will be exactly one transition from a state on an input symbol.	2) There can be zero, one (or) more transitions from a state on an input symbol.
3) Difficult to construct	3) Easy to construct.

Conversion of NFA to DFA :-

→ Subset construction method is used for converting NFA to DFA.

1) Convert the following NFA to DFA.



Solution:

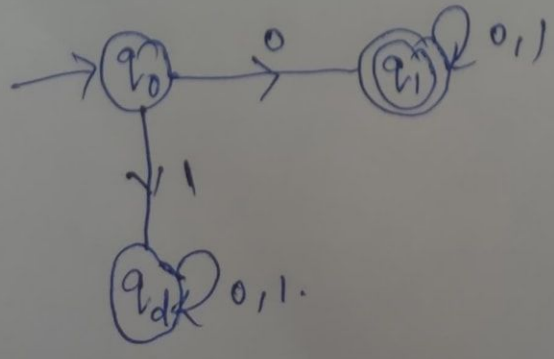
State transition table of NFA:

	0	1
→ q_0	q_1	ϕ
* q_1	q_1	q_1

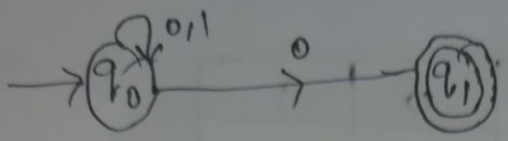
State transition table for DFA:

	0	1
→ $[q_0]$	$[q_1]$	$[q_d]$
* $[q_1]$	$[q_1]$	$[q_1]$
$[q_d]$	$[q_d]$	$[q_d]$

DFA :-



2) Convert the following NFA to DFA:



Solution:

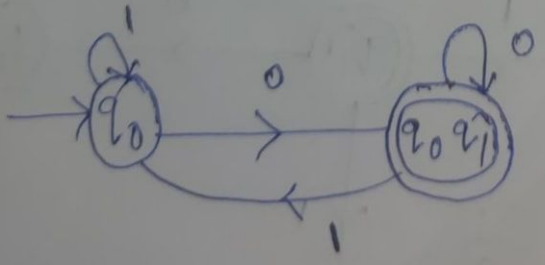
State transition table for NFA:

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

State transition table for DFA:

	0	1
$\rightarrow [q_0]$	$[q_0, q_1]$	$[q_0]$
$*[q_0, q_1]$	$[q_0 q_1]$	$[q_0]$

DFA:



3) Convert the following NFA to equivalent DFA

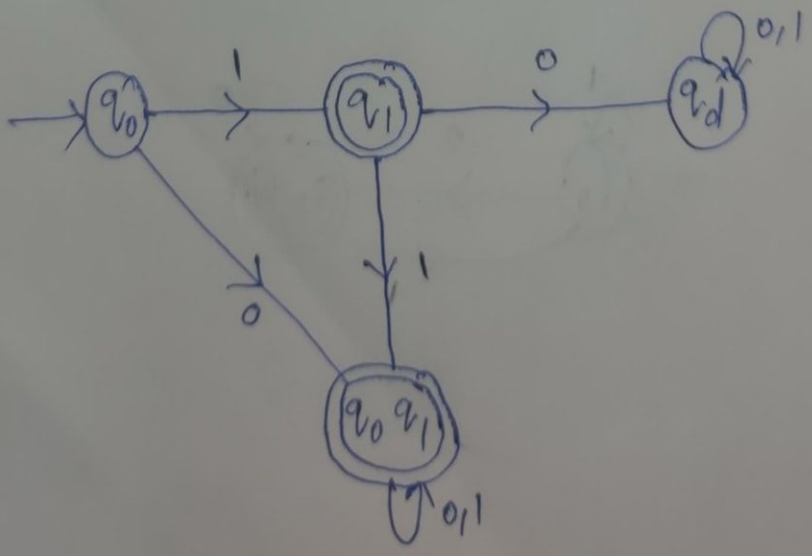
	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	q_1
$*q_1$	ϕ	$\{q_0, q_1\}$

Solution:-

State transition table for DFA:-

	0	1
$\rightarrow [q_0]$	$[q_0 q_1]$	$[q_1]$
$*[q_1]$	$[q_d]$	$[q_0 q_1]$
$*[q_0 q_1]$	$[q_0 q_1]$	$[q_0 q_1]$
$[q_d]$	$[q_d]$	$[q_d]$

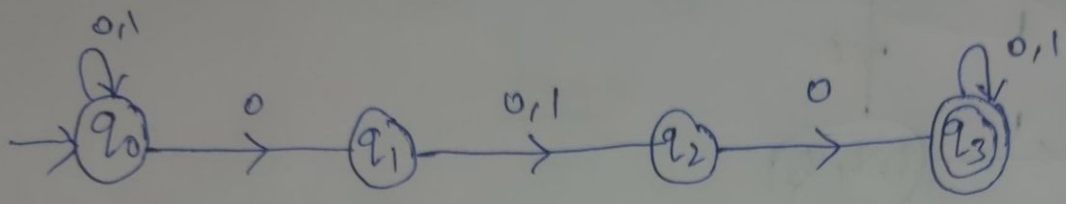
DFA:-



4) Construct NFA for $(0+1)^* 0(0+1)0(0+1)^*$ and convert to DFA.

Solution:

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



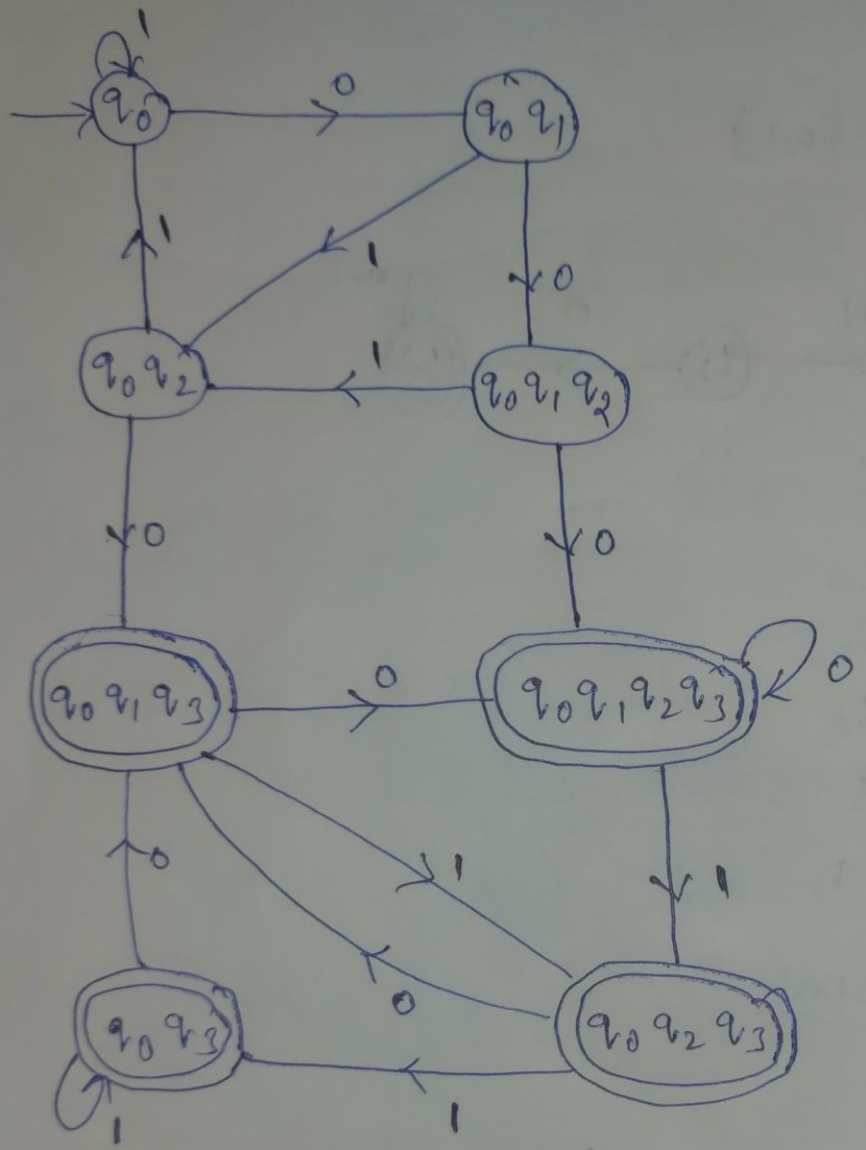
Transition table for NFA:

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	q_0
q_1	q_2	q_2
q_2	q_3	ϕ
$*q_3$	q_3	q_3

Transition table for DFA:

	0	1
$\rightarrow [q_0]$	$[q_0 q_1]$	$[q_0]$
$[q_0 q_1]$	$[q_0 q_1 q_2]$	$[q_0 q_2]$
$[q_0 q_2]$	$[q_0 q_1 q_3]$	$[q_0]$
$[q_0 q_1 q_2]$	$[q_0 q_1 q_2 q_3]$	$[q_0 q_2]$
$*[q_0 q_1 q_3]$	$[q_0 q_1 q_2 q_3]$	$[q_0 q_2 q_3]$
$*[q_0 q_2 q_3]$	$[q_0 q_1 q_3]$	$[q_0 q_3]$
$*[q_0 q_3]$	$[q_0 q_1 q_3]$	$[q_0 q_3]$
$*[q_0 q_1 q_2 q_3]$	$[q_0 q_1 q_2 q_3]$	$[q_0 q_2 q_3]$

DFA:-

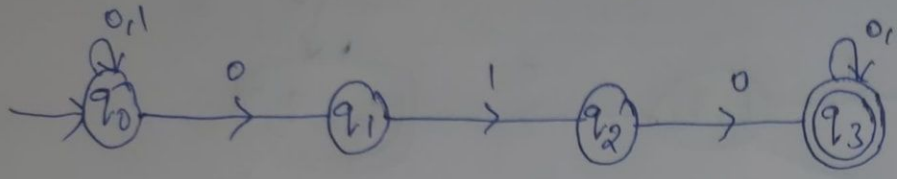


(SA)

5) Construct NFA for $(0+1)^*\,010\,(0+1)^*$ and convert to DFA.

Solution:-

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



NFA

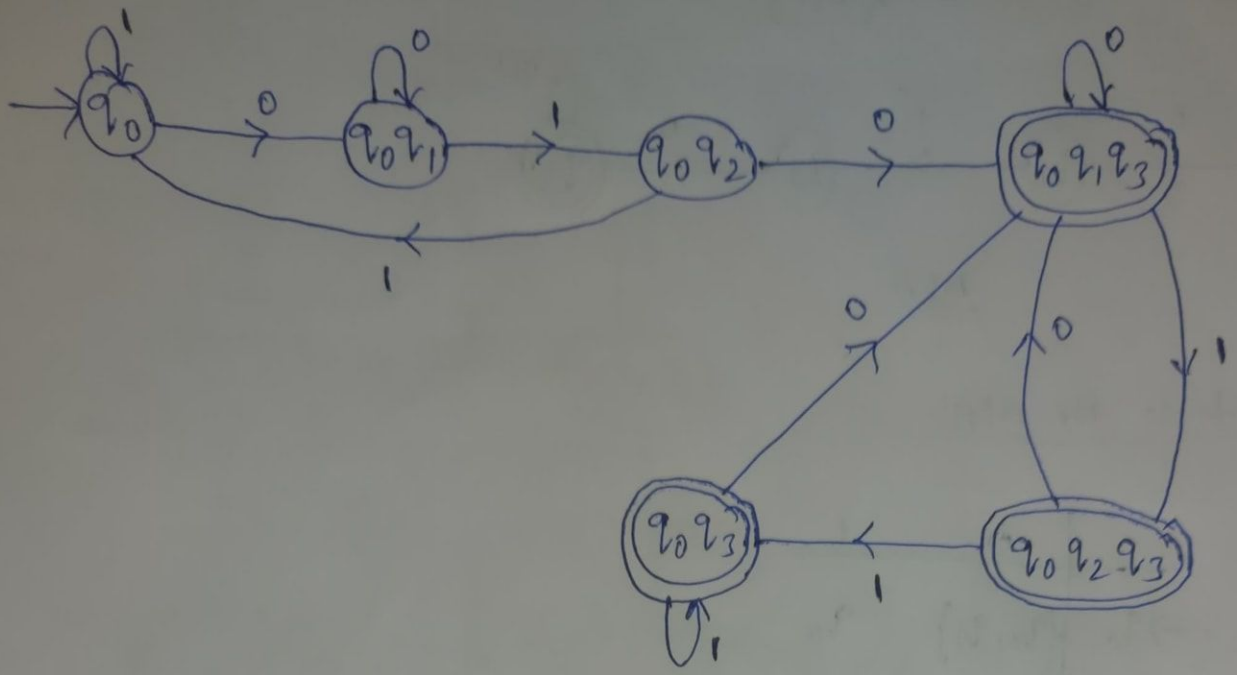
Transition table for NFA:-

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	q_0
q_1	ϕ	q_2
q_2	q_3	ϕ
$*\,q_3$	q_3	q_3

Transition table for DFA:-

	0	1
$\rightarrow [q_0]$	$[q_0\, q_1]$	$[q_0]$
$[q_0\, q_1]$	$[q_0\, q_1]$	$[q_0\, q_2]$
$[q_0\, q_2]$	$[q_0\, q_1\, q_3]$	$[q_0]$
$*[q_0\, q_1\, q_3]$	$[q_0\, q_1\, q_3]$	$[q_0\, q_2\, q_3]$
$*[q_0\, q_2\, q_3]$	$[q_0\, q_1\, q_3]$	$[q_0\, q_3]$
$*[q_0\, q_3]$	$[q_0\, q_1\, q_3]$	$[q_0\, q_3]$

DFA:-

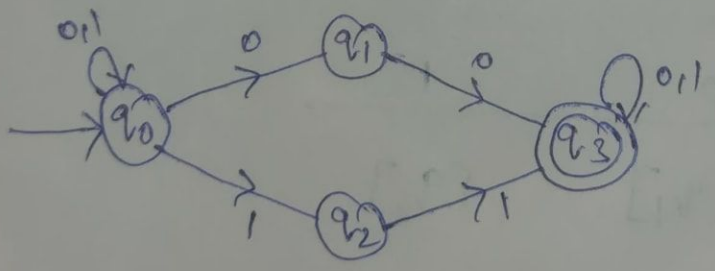


Transition diagram

6) Construct NFA for $(0+1)^*(00+11)(0+1)^*$ and convert to DFA.

Solution:- $\Sigma = \{0,1\}$

NFA:



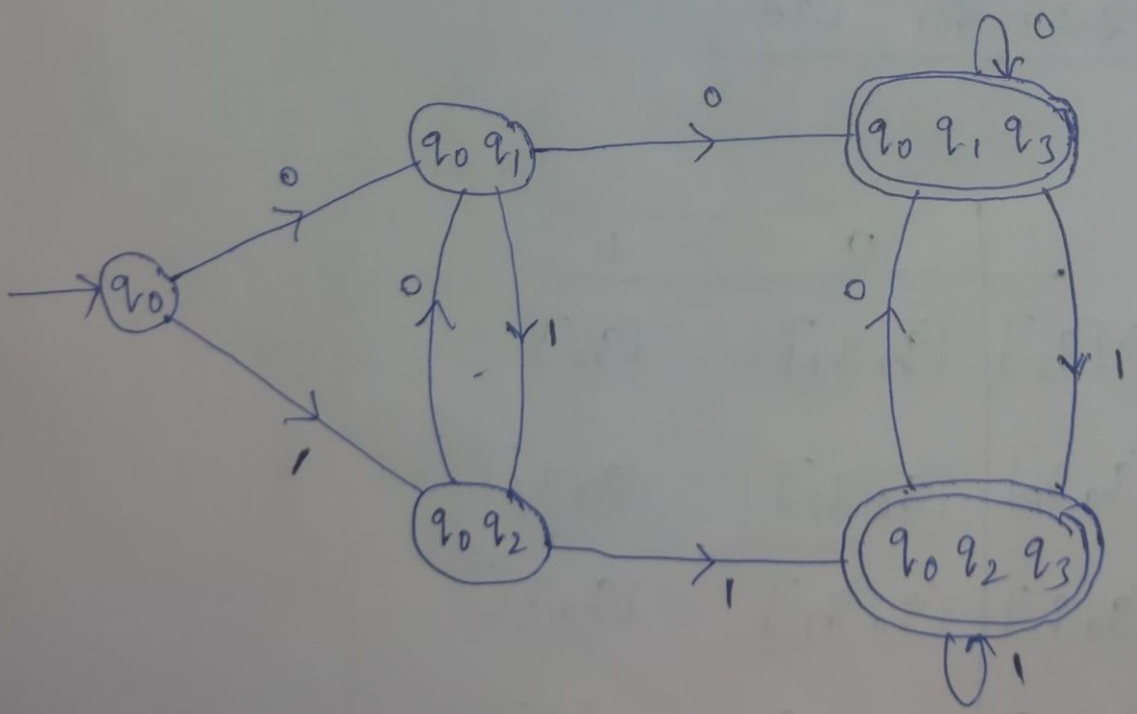
State transition table for NFA:

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

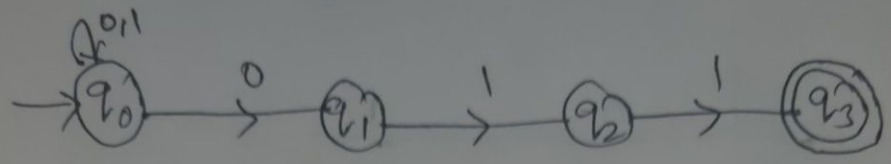
Transition table for DFA:-

	0	1
→ $[q_0]$	$[q_0 q_1]$	$[q_0 q_2]$
$[q_0 q_1]$	$[q_0 q_1 q_3]$	$[q_0 q_2]$
$[q_0 q_2]$	$[q_0 q_1]$	$[q_0 q_2 q_3]$
* $[q_0 q_1 q_3]$	$[q_0 q_1 q_3]$	$[q_0 q_2 q_3]$
* $[q_0 q_2 q_3]$	$[q_0 q_1 q_3]$	$[q_0 q_2 q_3]$

Transition diagram for DFA:-



7) Convert the following NFA to equivalent DFA:



Solution:

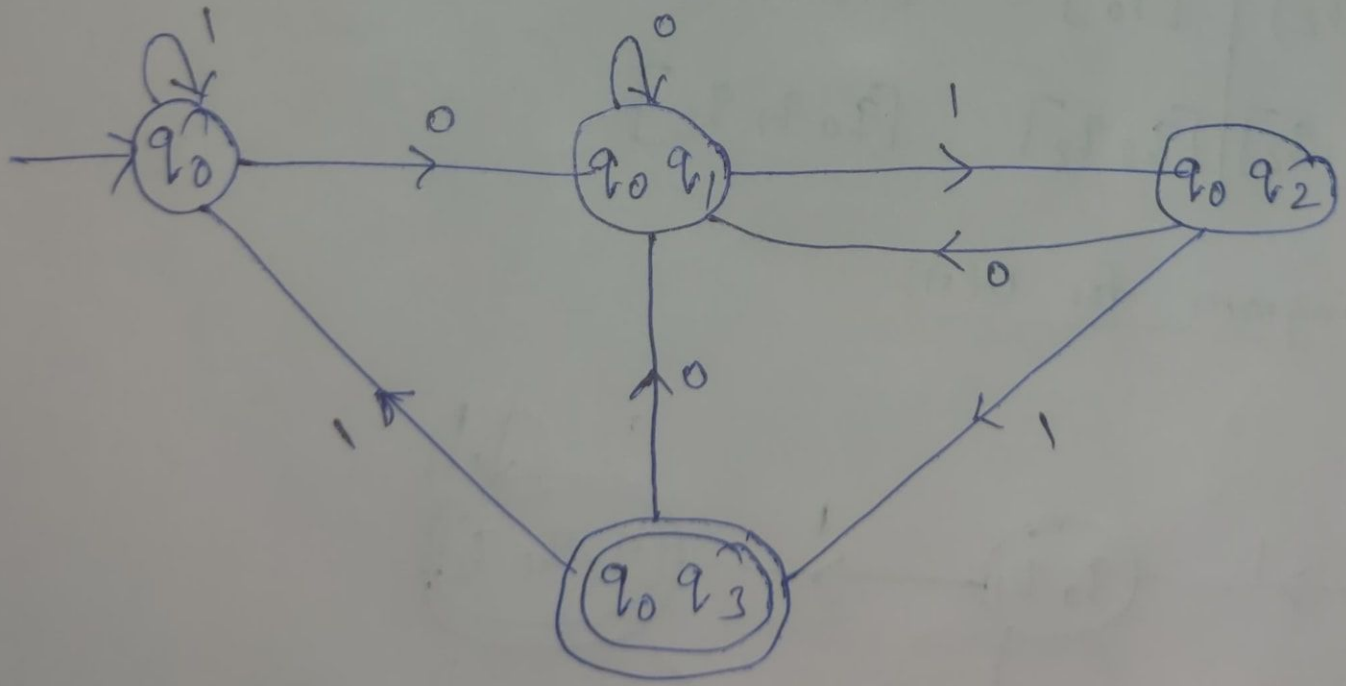
Transition table for NFA:

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

Transition table for DFA:

	0	1
→ $[q_0]$	$[q_0\ q_1]$	$[q_0]$
$[q_0\ q_1]$	$[q_0\ q_1]$	$[q_0\ q_2]$
$[q_0\ q_2]$	$[q_0\ q_1]$	$[q_0\ q_3]$
* $[q_0\ q_3]$	$[q_0\ q_1]$	$[q_0]$

transition diagram for DFA:



8) Convert the following NFA to equivalent DFA:

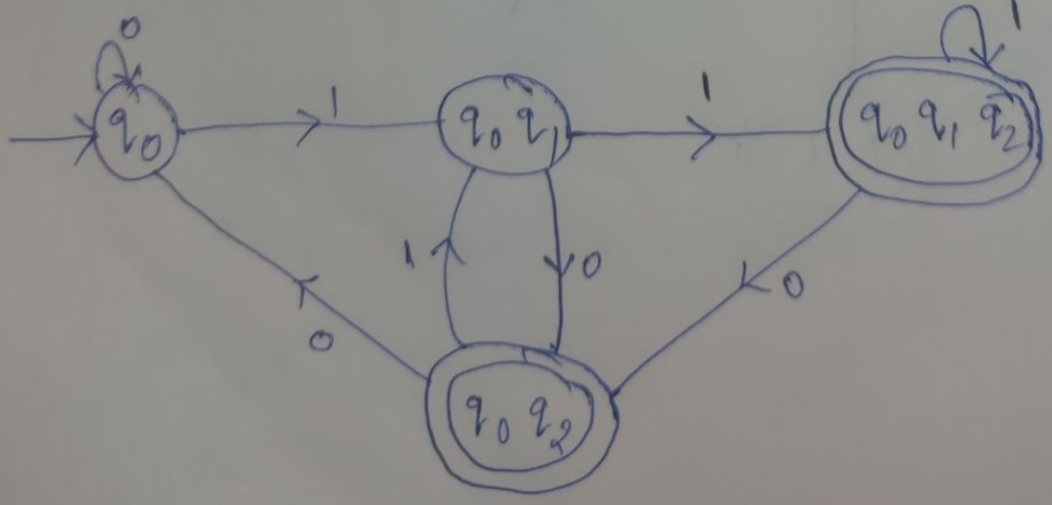
	0	1
$\rightarrow q_0$	q_0	$\{q_0 q_1\}$
q_1	q_2	q_2
$*q_2$	ϕ	ϕ

Solution:

Transition table for DFA:

	0	1
$\rightarrow[q_0]$	$[q_0]$	$[q_0 q_1]$
$[q_0 q_1]$	$[q_0 q_2]$	$[q_0 q_1 q_2]$
$*[q_0 q_2]$	$[q_0]$	$[q_0 q_1]$
$*[q_0 q_1 q_2]$	$[q_0 q_2]$	$[q_0 q_1 q_2]$

Transition diagram for DFA:



Unit - II

Regular Languages and Finite Automata.

1) The languages accepted by finite automata are called regular languages.

2) Regular languages are easily described (or) represented by simple expressions called regular expressions.

3) we have some algebraic notations to represent the regular expressions.

Regular sets:- A special class of sets of words over S, called regular sets, is defined recursively as follows

1) Every finite set of words over S (including ϵ, empty set $\{\}$) is a regular set

2) If A and B are regular sets over S, then
A∪B and AB are also regulars.

3) If S is a regular set over S, then so is its closure S^*

4) No set is regular unless it is obtained by repeating (1) to (3) any number of times.

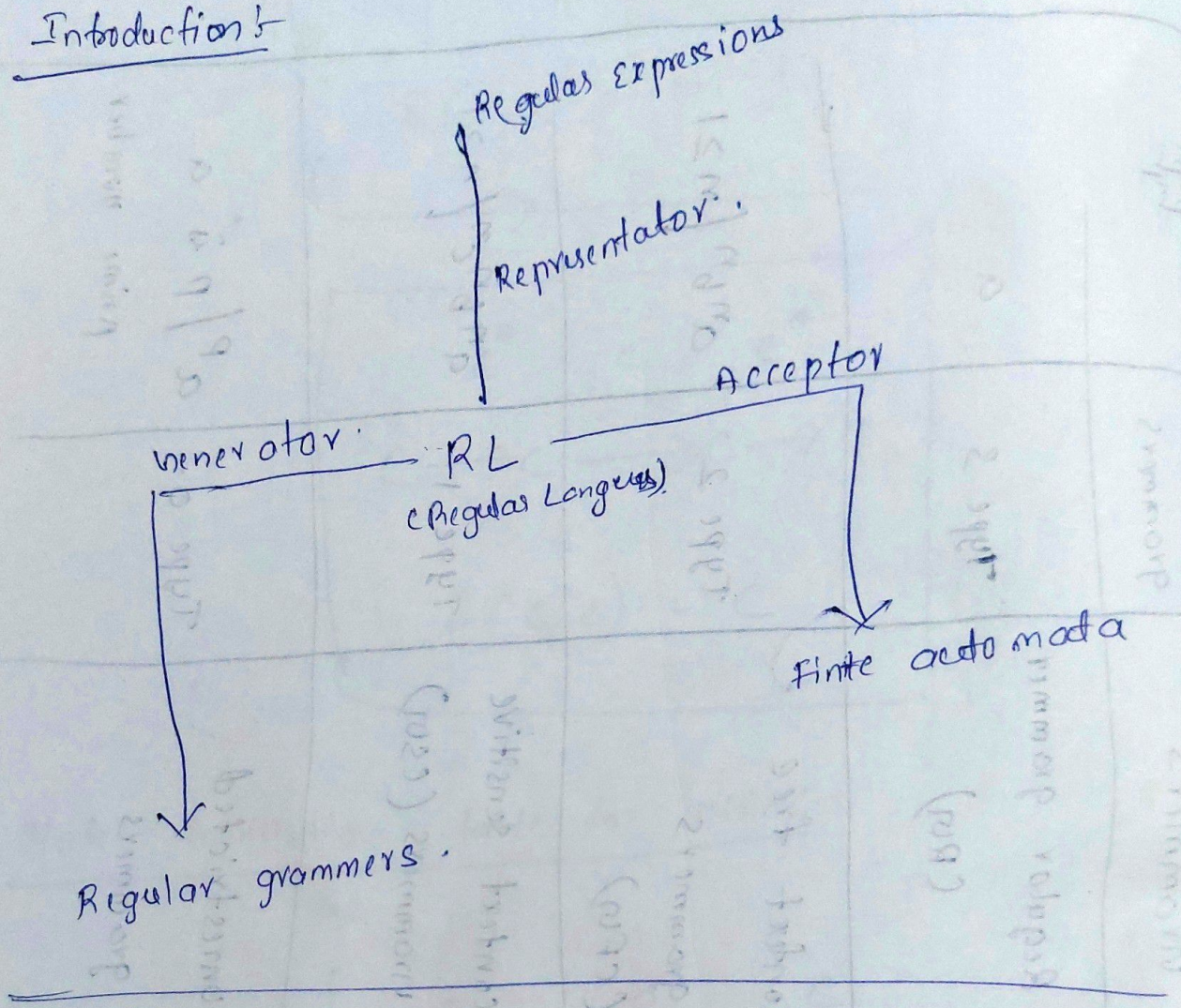
The class of regular sets over S is the smallest class containing all finite sets of words over S and closed under union, concatenation and Kleene closure.

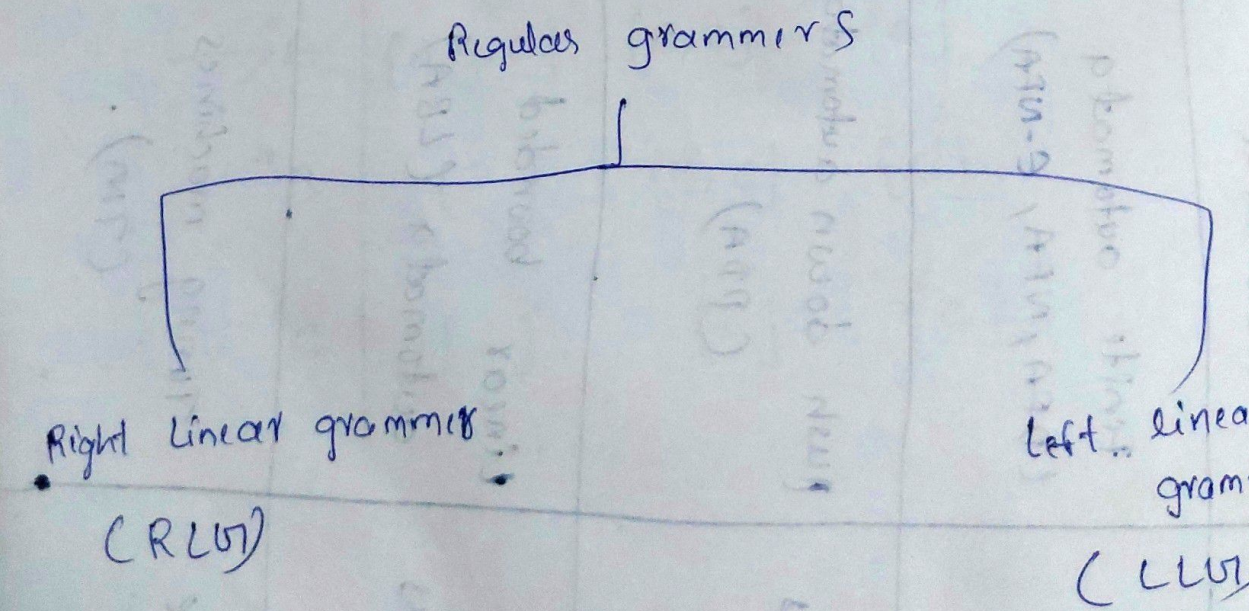
Ex: Let $\Sigma = \{0,1\}$, then set of strings $\{01,10\}$ is a regular set.

$\Sigma = \{a,b\}$, then set of strings that contain both odd number of a's & b's is a regular set.

Regular expressions and conversions

Introduction:





Regular expressions :-

Regular expressions contains three operators :

(i) + (union)

(ii) . (concatenation)

(iii) * (Kleene closure).

a) ϕ , ϵ , $a \in \Sigma$ are regular expressions.
↓ ↓ ↓
primitive
$\{\}$, $\{\epsilon\}$

b) r_1 and r_2 are primitive regulas expressions, then

(i) $r_1 + r_2$ is a regulas expression.

(ii) $r_1 . r_2$ " " "

(iii) r_1^* , r_2^* " " "

Regular expressions	Language generated
ϕ	$\{\ \}$
ϵ	$\{\epsilon\}$
a	$\{a\}$
a^*	$\{\epsilon, a, aa, \cdots\}$
a^+ $(a^+ = a \cdot a^*)$ (or) $(a^+ = a^* \cdot a)$.	$= a \cdot a^*$ $= \{a, aa, aaa, \cdots\}$
$(a+b)^*$ (set of all strings possible over $\{a, b\}$	$\{\epsilon, a, b, ab, ba, bb, ab, \cdots\}$.

Examples of Regular expressions :-

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

→ L_1 = { set of all strings whose length is exactly 2 }

= { ab, aa, ba, bb }.

Regular expression (RE)

= aa + ab + ba + bb

= a(a+b) + b(a+b)

RE = (a+b)(a+b)

for exactly '3' length,

RE = (a+b)(a+b)(a+b)

for exactly '4' length,

RE = (a+b)(a+b)(a+b).

→ L = { set of all strings whose length is atleast '2' }.

= { aa, ab, ba, bb, aaa, bbb, ... }.

RE = $(a+b)(a+b)(a+b)^*$

Note:- If Language is infinite, there should be a '*' in the RE.

→ L = { set of all strings whose length is atmost '2' }.

= { ∈, a, b, aa, ab, ba, bb }.

RE = $\epsilon + a + b + aa + ab + ba + bb$.

= ~~$\epsilon + a + b + (aa + ab + ba + bb)$~~

= $(a+b+\epsilon)(a+b+\epsilon)$

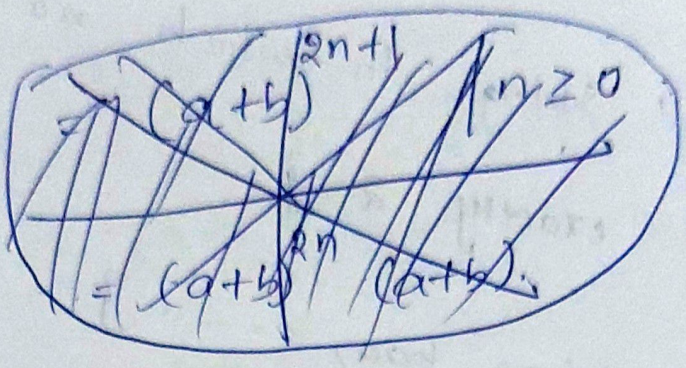
→ L = { set of all strings of even length }.

= { ∈, aa, ab, ba, bb, aabb, abab, ... }.

RE = $((a+b)(a+b))^*$

→ L = { set of all strings of odd length }.

= { a, b, aaa, aba, baa, ... }.



RE = $((a+b)^2)^*(a+b)$

~~$(a+b)^{2n}((a+b)^2)^*$~~

RE = $((a+b)(a+b))^*(a+b)$

→ L = { set of all strings whose length is divisible by 3 }.

= { ϵ, aaa, bbb, aba, baa, bba, ... }.

RE = $((a+b)(a+b)(a+b))^*$

↳ set of all strings of length 3.

~~$(a+b)^{3n}$~~

→ L = { set of all strings whose length when divided by '3' leaves a remainder of '2' }.

$RE = ((a+b)(a+b)(a+b))^* (a+b)(a+b)$

→ L = { set of all strings in which no. of a's are exactly 2 }.

= { aa, aab, aba, baa, ... }.

$RE = b^* a b^* a b^*$

→ L = { no. of a's are atleast 2 }

$RE = b^* a b^* a (a+b)^*$

→ L = { no. of a's are atmost 2 }.

$RE = b^* (\epsilon + a) b^* (\epsilon + a) b^*$

→ L = { no. of a's are even }.

$RE = (b^* a b^* a b^*)^* + b^*$

→ L = { starts with a }

RE = $a \cdot (a+b)^*$

→ L = { ends with a }

RE = $(a+b)^* a$.

→ L = { contains a }.

RE = $(a+b)^* a (a+b)^*$

→ L = { starting and ending with different symbol }.

RE = $a(a+b)^* b + b(a+b)^* a$.

→ L = { starting and ending with same symbol }.

= { ∈, a, b, aa, bb, abb, ... }.

RE = $a(a+b)^* a + b(a+b)^* b + a + b + \epsilon$.

→ L = { no. two a's should come together }.

= { ∈, b, bb, bbb, ..., a, ab, aba, abab, ...
ba, bab, baba, babab ... }

RE = $(b+ab)^*$ + $(b+ab)^*$ a.

= $(b+ab)^*$ (∈ + a).

(or)

RE = a . $(b+ba)^*$ + $(b+ba)^*$

= $(a+\epsilon)(b+ba)^*$

→ L = { no. two a's and no two b's should come together }.

= { ∈, a, b, ab, ba, aba, bab, ... }.

Starts with	Ends with
{a, aba, ababa, ...} → a	a → $(ab)^*a$ (or) $a(ba)^*$
{ab, abab, ababab, ...} → a	b → $(ab)^*$ (or) $a(ba)^*b$
{ba, baba, bababa, ...} → b	a → $(ba)^*$ (or) $b(ab)^*a$
{b, bab, babab, ...} → b	b → $(ba)^*b$ (or) $b(ab)^*$

$$RE(1) = (ab)^* a + (ab)^* + b(ab)^* a + b(ab)^*$$

(or)

$$RE(2) = a(ba)^* + a(ba)^* b + (ba)^* + (ba)^* b.$$

After simplifying,

$$RE(1) = (ab)^* (a+\epsilon) + b(ab)^* (a+\epsilon).$$

$$= (\epsilon + b)(ab)^* (a+\epsilon).$$

$$RE(2) = a(ba)^* (\epsilon + b) + (ba)^* (\epsilon + b).$$

$$= (\epsilon + b)(ba)^* (\epsilon + b)(\epsilon + a).$$

$$= (\epsilon + a)(ba)^* (\epsilon + b).$$

pumping Lemma for regular Languages :-

If L be an infinite regular Language and $w \in L$ such that $|w| \geq n$ for some positive integer 'n', then 'w' can be decomposed into three strings $w = xyz$ such that

(i) $y \neq \epsilon$

(ii) $|xy| \leq n$.

(iii) for every $k \geq 0$, the string $xy^k z$ is also in L.

Applications of pumping Lemma :-

1) every regular Language satisfies the pumping Lemma property.

2) If a Language 'L' does not satisfy the pumping lemma property, then 'L' is non-regular.

3) If a language 'L' satisfies the pumping lemma property, then 'L' need not be regular.

4) pumping Lemma is used to prove that some of the languages are not regular.

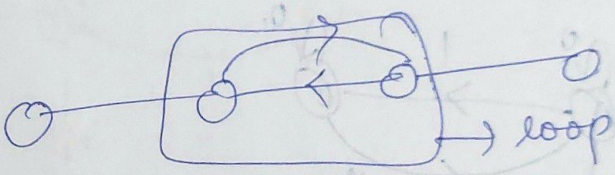
Testing whether a language is regular (or) not :-

If a language is finite, it is always regular.

If a language is infinite, it may be regular (or) not.

An infinite language is regular only if there is a loop in the finite automata.

Ex: $L = \{ab, abab, ababab, \ldots\}$ — pattern: ab-pattern.



pumping Lemma:

An infinite language is not regular only if there is a no pattern in the language.

This is called pumping Lemma..

pumping Lemma is a negativie test.

pumming Lemma doesn't say that if there is a pattern in the language, it is regular.

Steps to prove that the language is not regulas using pumping lemma :-

1) Assume that A is regulas.
2) It has to have a pumping length (say p).
3) All strings s longer than p can be pumped $|s| \geq p$.
4) Now find a string 's' in A such that $|s| \geq p$.
5) Divide s into xyz.
6) Show that $xy^iz \notin A$ for some i.
7) Then consider all ways that s can be divided into xyz.
8) Show that none of these can satisfy all the 3 pumping conditions at the same time.
9) s cannot be pumped == CONTRADICTION.

prove that the language $A = \{a^n b^n / n \geq 0\}$ is not regular.

proof:

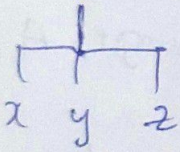
Assume that A is regular.

pumping length = 4 (p)

$S = a^p b^p$

$= a^4 b^4$

S ⇒ aaaabbbb



Case(i): The 'y' is in the 'a' part.

$S = \underbrace{aaaa}_{x\ y}\underbrace{bbbb}_{z}$

$xy^i z \Rightarrow xy^2 z \quad (i = 2)$

⇒ aaaaaabbbb

no. of a's ≠ no. of b's

6 ≠ 4.

string 'aaaaaabbbb' is not in the language 'A'.

So, by contradiction 'A' is not regular.

case '2' :- The y is in the 'b' part.

$$S = \underbrace{aaaa}_{x}\underbrace{bbb}_{y}\underbrace{b}_{z}$$

$xy^iz \Rightarrow xy^2z \quad (i=2)$

$$\Rightarrow \underbrace{aaaa}_{x}\underbrace{bbbb}_{y}\underbrace{bb}_{z}$$

No. of a's $\neq$ no. of b's

$4 \neq 6$

string 'aaaabbbbbb' is not in the language A.

case '3' :- The y is in the 'a' and 'b' part.

$$S = \underbrace{aaa}_{x}\underbrace{ab}_{y}\underbrace{bbb}_{z}$$

$xy^iz \Rightarrow xy^2z \quad (i=2)$

$\Rightarrow$ aaaababbbb

string 'aaaababbbb' is not in the language 'A'.

So, by contradiction 'A' is not regular.

prove that the following language $\{a^n | n$ is a prime number$\}$ is not regular?

Solution:

Assume given language a^n is regular.

pumping length = 7 (prime number).

string(w) = aaaaaaa

Now divide the string into 3 parts x, y, z.

$$w = \underbrace{a}_{x}\underbrace{aaaaa}_{y}\underbrace{aa}_{z}$$

xy^Kz = $\underbrace{aa}_{x}\underbrace{aaaaaa}_{y^2}\underbrace{aa}_{z}$ (K = 2)

= aaaaaaaaaaa (10).

10 is not a prime number.

So, the string a^{10} aaaaaaaaaa is not in the given language.

So, by contradiction, a^n is not regular.