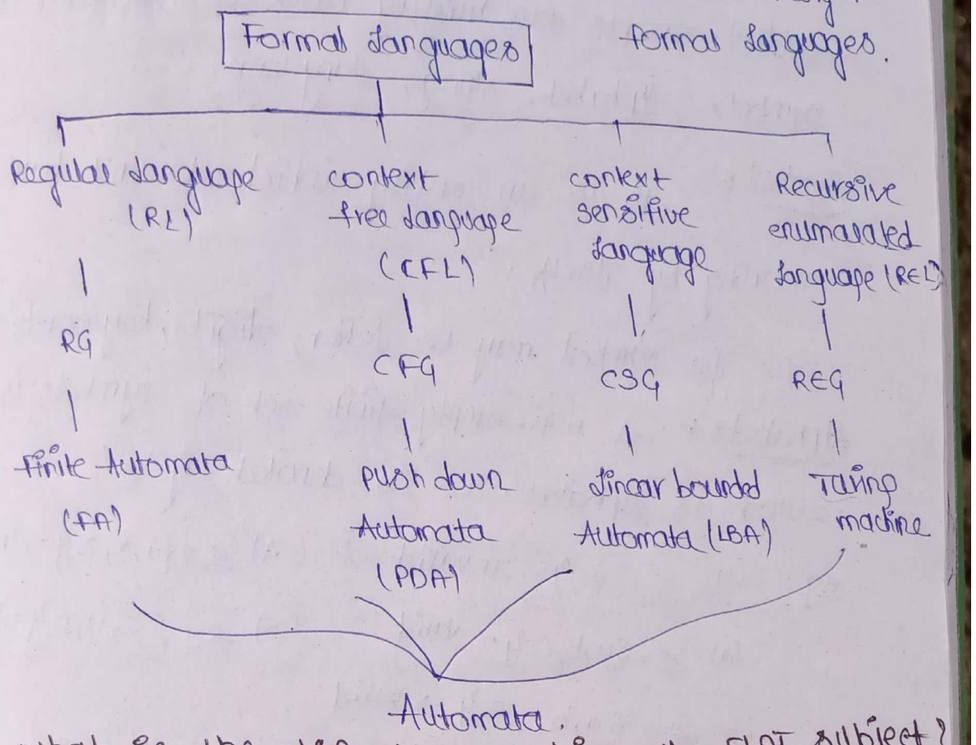
FLAT

TOC - Theory of competitions

ATFL - Automata theory & formal languages.



what is the use after studying the FLAT subject?

we can understand how the formal language is recognised by the corresponding automata, it is studied in theoritical way.

Short notations:-

FA - finite Automata

PDA - Push down Automata

LBA - linear bounded Automata

TM - Turing machine

RG - Regular Grammar

CFG - context free Grammar

CSG - context sensitive Grammar

REG - Recursive enumerated Grammar

RL - Regular language

CFL - context free language

CSL - context sensitive language

REL - Recursive Enumerated language

DFA - Deterministic finite Automata.

NFA - Non-deterministic finite Automata.

⇒ Central Concepts of Automata [Theory]

Central concepts are building block of Automata are symbols, Alphabet, string, language.

Symbol :- It is an entity which doesn't have meaning by itself.

Eg :- The symbol may be letter, digit, keywords etc.,

Alphabets :- A non-empty finite set of symbols is called as Alphabet. It is denoted by 'Σ'

Eg :- (1) $\Sigma = \{ y$ - Invalid (4) $\Sigma = \{1,2,3 y$ - Invalid

(2) $\Sigma = \{a,b,c y$ - valid (5) $\Sigma = \{-,v,n y$ - Valid.

(3) $\Sigma = \{a,b \ldots z y \rightarrow$ valid

Strings :- The sequence of symbols from the given alphabet is called as string. It is indicated by 's' (or) 'w'.

Eg :- Identify the following strings which are valid (or) invalid for the alphabet $\Sigma = \{a,b y$

The string is valid when all symbols of that string should be belongs to the given alphabet

(1) $w_1 = ab$ ✓ (3) $w_3 = abcab$ ✗

(2) $w_2 = abab$ ✓ (4) $w_4 = abcaa$ ✗

Length of a string :- It is nothing but how many no. of symbols are present in that string. It is indicated '$|w|$'.

Eg :- (1) $|w_1| = |ab| = 2$ (2) $|w_2| = |abab| = 4$

Prefixes

Suffixes

Subst

Prefixes of a string :- (left to right)

w_1 = vasavi

Prefixes → ϵ, v, va, vas, vasa, vasav, vasavi

Suffixes of a string :- (right to left)

w_1 = vasavi

vasavi, asavi, savi, avi, vi, i, ϵ.

Substrings of a string :-

w_1 = gate

Identify 0 length substring - ϵ - 1

1 length substring - g, a, te - 4

2 length substring - ga, at, te - 3

3 length substring - gat, ate - 2

4 length substring - gate - 1

11

The no. of substrings for n length substring is, $\frac{n(n+1)}{2} + 1 = \leq n+1$

Trivial substrings :-

Zero length and n length substrings for a string are called as trivial substrings

w = gate

→ ϵ, gate (Trivial)

other than trivial substrings are called as non-trivial substrings.

→ g, a, t, e, ga, at, te, gat, ate (non-trivial)

Language (L) :-

The collection of the strings is called as language, denoted by (L).

→ The language may be either finite (or) Infinite depends upon the condition

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

(1) Generate the language where string length should be zero.

$L_1 = \{\epsilon\}$

(2) Generate the language where string length is should be exactly one

$L_2 = \{a, b\}$

(3) Generate the language whose strings length are '3'.

$L_3 = \{aaa, aab, abb, aba, bbb, bab, baa, bba\}$

(4) Generate the language where strings lengths are atleast '2'.

$L_4 = a$

Finite Automata :-

The finite automata consists of 3 components they are

1) Input tape
2) tape header (Pandar)
3) finite control

(1) Input tape :-

The tape is divided into cells where each cell is capable of storing one input cell only

Eg:- consider the string $w = \{abba\}$

it can be stored into the input tape as shown

(or) Infinite

ngth should

length is

length are

a, bbab

s are

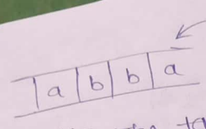
3 components

s where

only

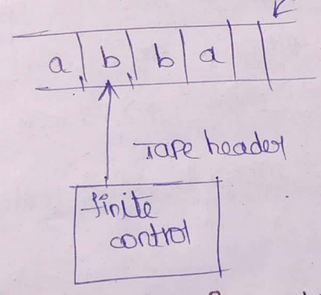
shown

below.

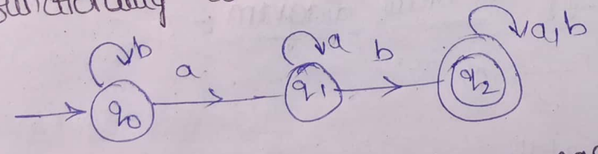


2) Tape header :- The tape header is represented with arrow and it is pointing to particular cell of the input tape that means the tape header can point to only one cell at a time

3) finite control :- The finite control controls the entire operation of the automata. The finite control needs the symbol which is pointing by the tape header & then performs the transition and it can be depicted as shown below.



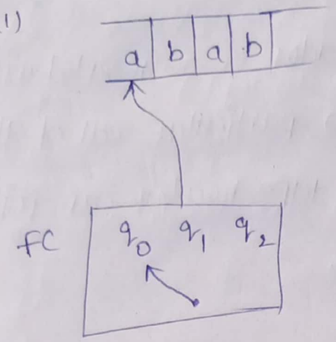
Consider the following state diagram and identify the functionality of the finite automata.



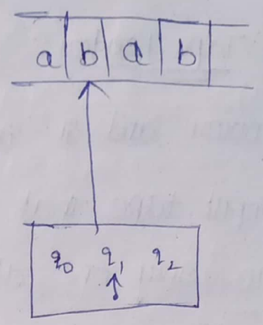
consider the string w = abab, we can store the given i/p string into the i/p tape & then list out the functionality of finite automata as shown below.

W = abab

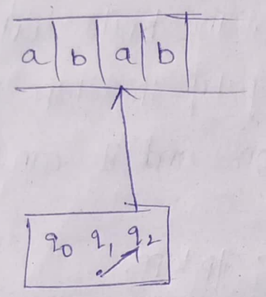
(1)



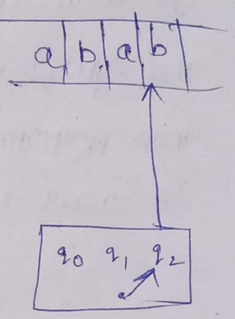
(2)



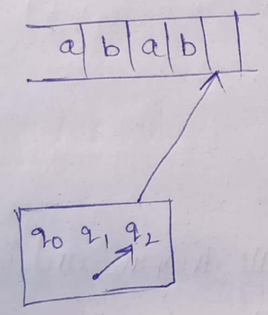
(3)



(4)



(5)



Terminology of state diagram :-

(1) state — ○

(2) Infial state — →○

(3) finite state — ◎

(4) unreachable state :-

The state which is not reachable from the initial state that is called as unreachable state.

(5) dead state :-

The state from which we can't reach the final state, it is called as dead state.

Transition systems :-

The system which performing the transition is called as transition system.

→ here, transition means moving from one state to another state (or itself).

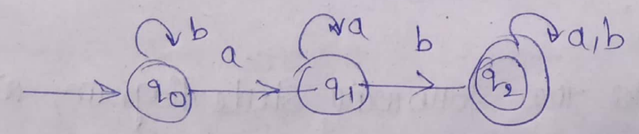
Transition systems is represented in the following ways:

1) state transition diagram
2) state transition table

(1) state transition diagram :-

In state transition diagram the transition is occured by using the i/p symbols

Eg:- Construct the state transition diagram where strings having a, b as substrings as shown below.

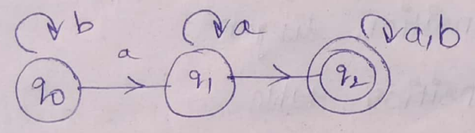


2) State transition table :-

In state transition table, rows are represented with states & columns are represented with ip symbols initially.

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

we can construct the state diagram from state transition table as shown below.



Acceptance of a string by a finite Automata :-

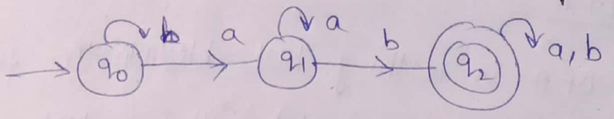
We can say whether the string is accepted by a finite automata (or) not.

→ the string is accepted when we are reaching the finial state after reading the last symbol of the string.

Consider the following state diagram, which is constructed for the recognition of strings

which are having ab as substring.

State diag



(1) W_1 = aababa.

$\rightarrow q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_2 \xrightarrow{b} q_2 \xrightarrow{a} q_2$

(final state accepted)

(2) W_2 = bbbba

The string is rejected when we don't go the final state after reading the last symbol of the i/p string.

$\rightarrow q_0 \xrightarrow{b} q_0 \xrightarrow{b} q_0 \xrightarrow{b} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_1$

(not final state)

(3) W_3 = baaa

$\rightarrow q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_1 \xrightarrow{a} q_1$

(not final state)

Hence it is rejected.

(4) W_4 = baab

$\rightarrow q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_1 \xrightarrow{b} q_2$

(final state).

Hence accepted.

(5) W_5 = baaaaaa

$\rightarrow q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_1 \xrightarrow{a} q_1 \xrightarrow{a} q_1 \xrightarrow{a} q_1 \xrightarrow{a} q_1$

(not final state)

Hence it is rejected.

Def of DFA :-

DFA is nothing but deterministic finite automata & it is represented with 5-tuple as shown below.

$$< Q, q_0, \Sigma, \delta, f >$$

where

Q = set of all states

q_0 = intial state

Σ = set of i/p symbols

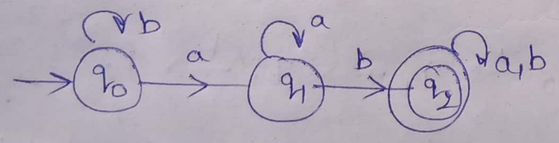
δ = transition function and represented as $\delta : Q \times \Sigma \rightarrow Q$

f = set of final states, $f \subseteq Q$

Note :-

In any state diagram it is having only one intial state and it is having any number of final states.

Consider the following DFA state diagram :-



from the above state diagram, the 5-tuple representation is,

1) $Q = \{ q_0, q_1, q_2 \}$

Initial state $= q_0$

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

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

$q_1 \times a \rightarrow q_1$

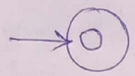
$q_1 \times b \rightarrow q_2$

$F = \{q_2\}$

Design of DFA :-

1) for the language design DFA

$L = \{\in\}$



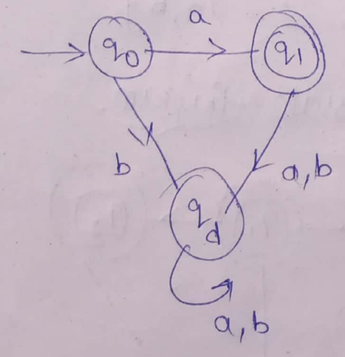
2) Design DFA to accept the language

$L = \{a\}$; over the alphabet $\Sigma = \{a, b\}$

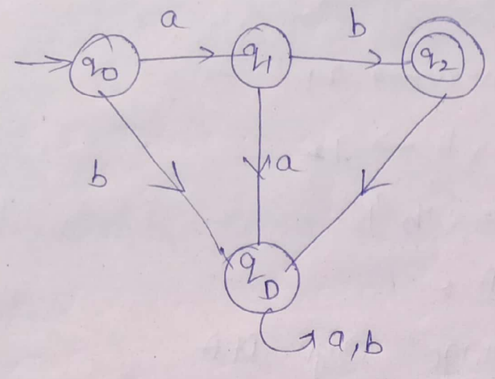
Note :-

In DFA, every state should performs exactly one transition for every input symbol of the given alphabet.

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

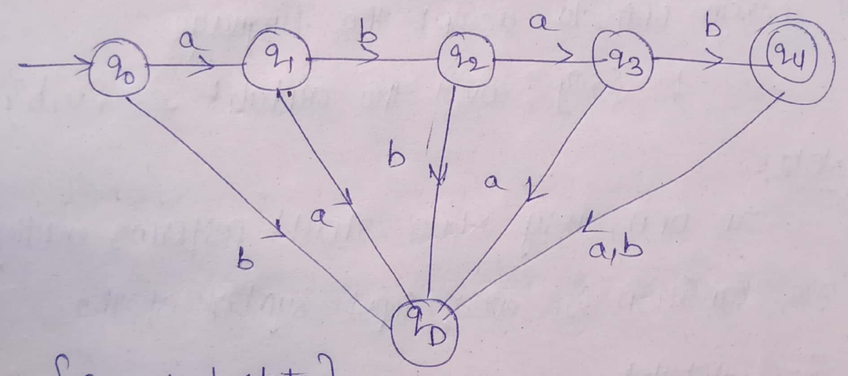


$L = \{a, b\}$ $\Sigma = \{a, b\}$



(3) $L = \{abab\}$; $\Sigma = \{a, b\}$

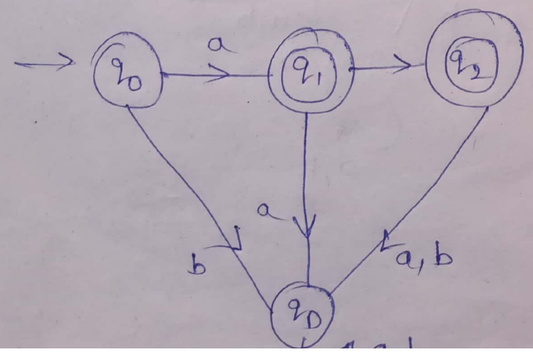
State diagram :-



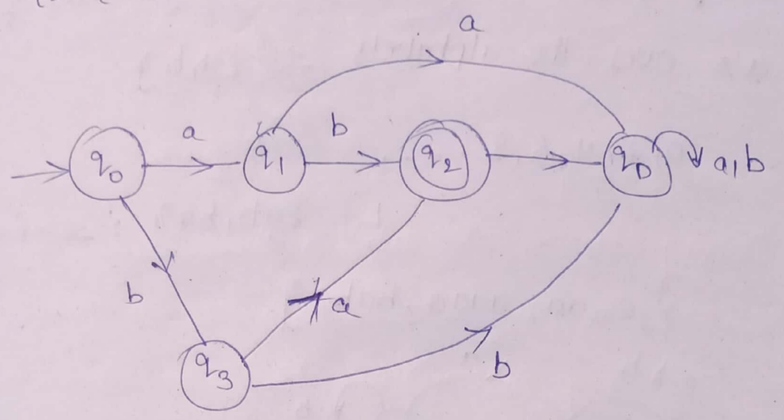
$\{q_D$ - dead state$\}$

(4) $L = \{a, ab\}$; $\Sigma = \{a, b\}$

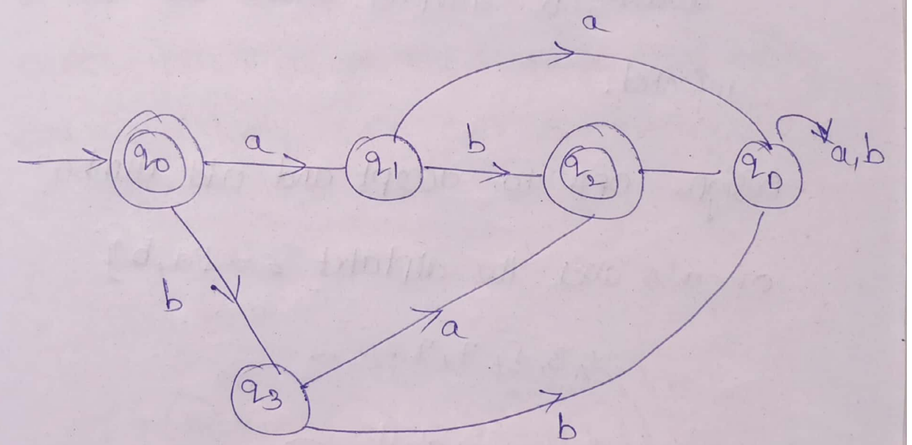
construct the state diagram.



(5) $L = \{ab, ba\}$, $\Sigma = \{a, b\}$



(6) $L = \{\epsilon, ab, ba\}$

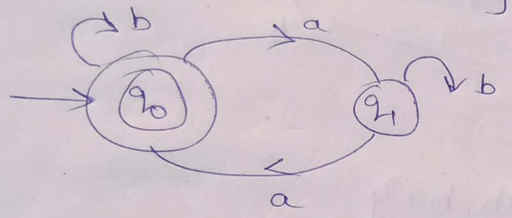


Design DFA to accept the even number of a's over the alphabets $\Sigma = \{a, b\}$

0, 2, 4, 6, 8 . . . .

$L = \{ab, ba\}$; $\Sigma = \{a, b\}$

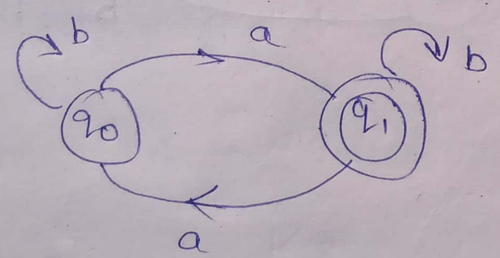
$\{\epsilon, aa, aaaa, baba\}$



aaaa is accepted while as aaa is rejected.

Design DFA to accept and odd number of a's over the alphabet $\Sigma = \{a, b\}$

1, 3, 5, 7, 9 . . .



Design

contai

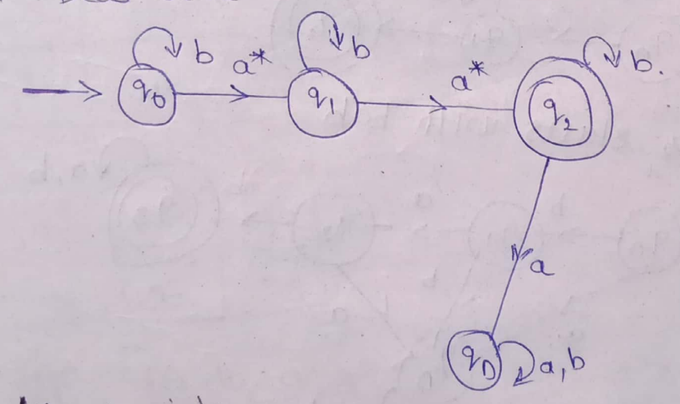
1.

2.

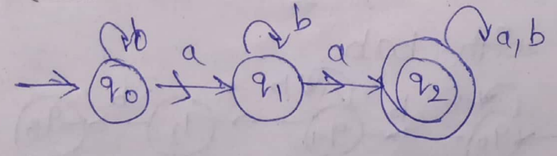
3.

Design DFA to accept the language whose string contains exactly 2 a's over $\Sigma = \{a, b\}$

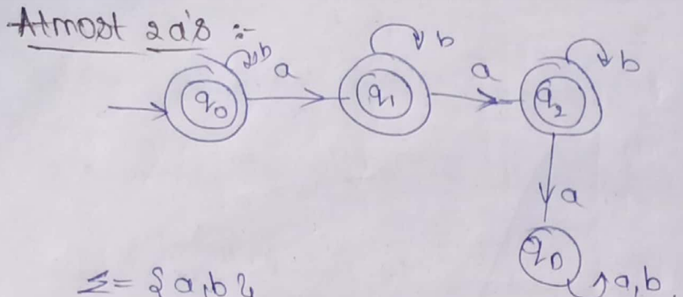
1. Loop
2. Previous state
3. Dead state



Atleast 2 a's



Atmost 2 a's :-



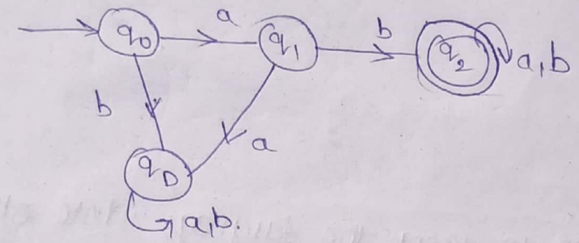
$\Sigma = \{a, b\}$

(1) string starts with 'ab' → ab____

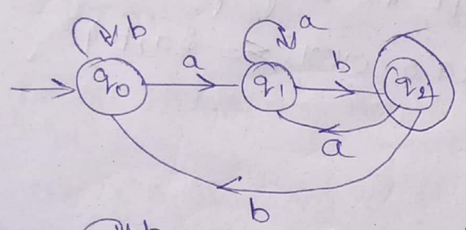
(2) ends with ab → ____ab

(3) sub string ab → ___ab___

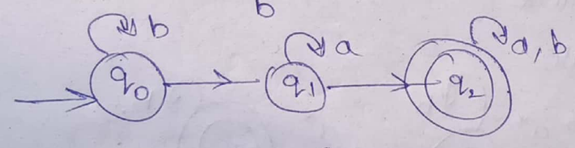
(1)



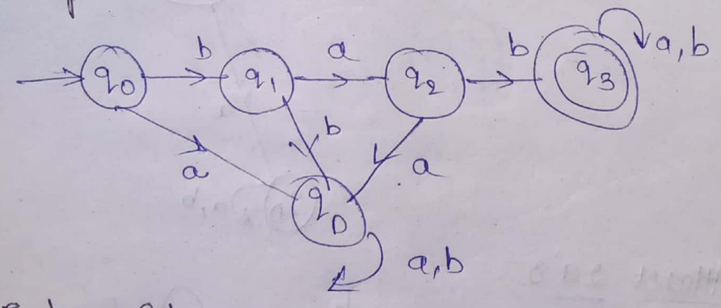
(2)



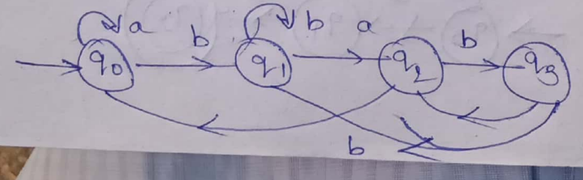
(3)



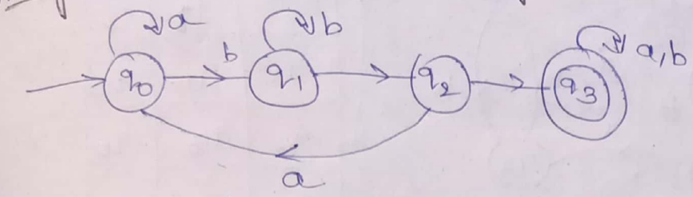
Strings starts with bab



Ends with bab

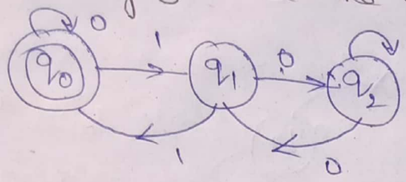


String having bab as substring



Design DFA to accept the binary numbers which are divisible by 3 over the alphabet $\Sigma = \{0,1\}$

(nth remainder = n-1
3 = 0,1,2
4 = 0,1,2,3)



(or) →

Σ / Q	0	1
*q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

B D

0 → 0 mod 3 = 0

1 → 1 mod 3 = 1

10 → 2 mod 3 = 2

11 → 3 mod 3 = 0

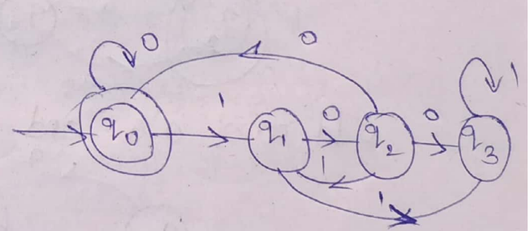
100 = 4 mod 3 = 1

101 = 5 mod 3 = 2,

Design DFA to accept the binary strings which are divisible by 4 over the alphabet $\Sigma = \{0,1\}$

divisible by 4 → 0, 1, 2, 3

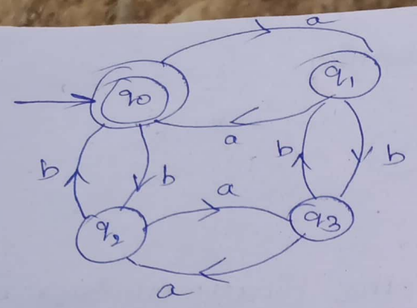
Σ / Q	0	1
→ *q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_0	q_1
q_3	q_2	q_3



★ Design DFA to accept even no. of a's and even no. of b's in the strings over the alphabet $\Sigma = \{a,b\}$

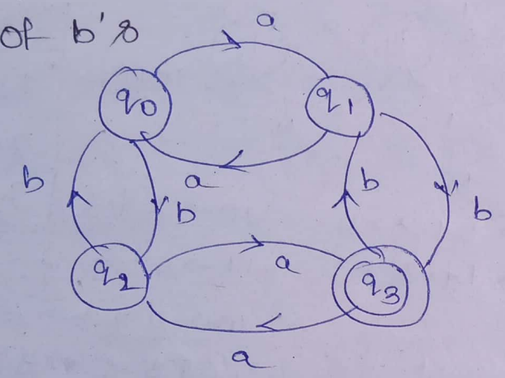
$\in$ - a's - 0, 2, 4, 6 ---

$\in$ b's - 0, 2, 4, 6 ---



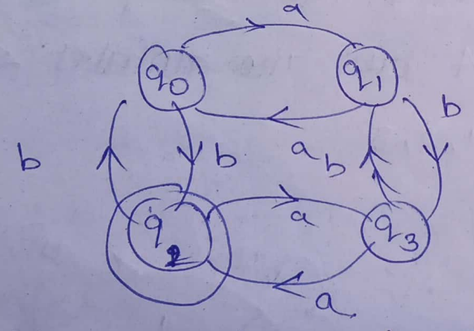
	a	b
* q_0	q_2	q_1
q_1	q_0	q_3
q_2	q_3	q_0
q_3	q_2	q_1

Design dfA to accept odd no. of a's and odd no. of b's

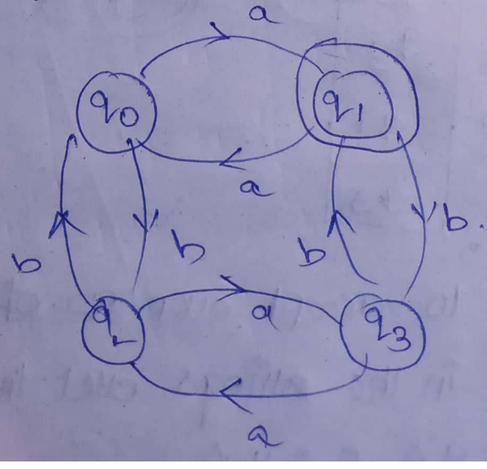


aaabbb

Even no. of a's and Odd no. of b's

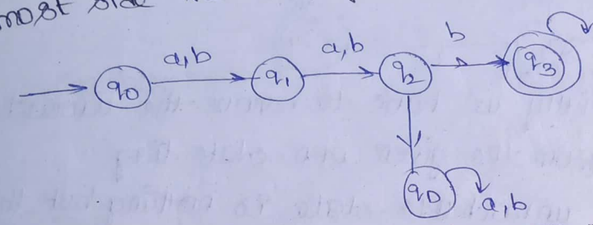


Odd no. of a's and even no. of b's



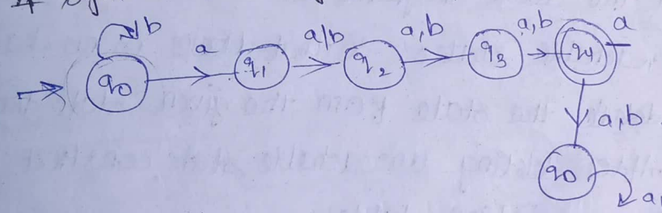
aaabb

Design DFA to accept the language where the string should have, b as the input at from left most side the third symbol should be b.



L → R
a/b a/b b

Design DFA to accept the language for the strings 4 symbol from right end is always a.

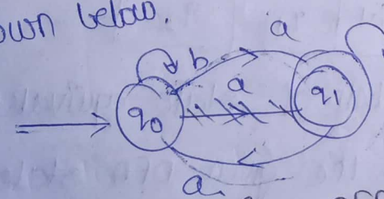


a/b a/b a/b a/b

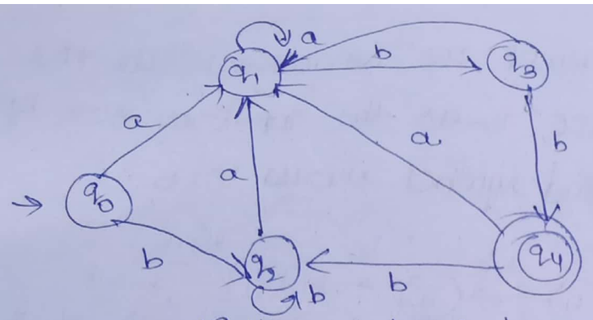
Minimization of DFA :-

for a problem there may be no. of DFA state diagram but for every problem there should be unique minimized DFA state diagram.

Q:- for accepting the language, strings having odd no. of a's, we are having 2 DFA state diagrams, as shown below.



Minimize the following DFA state diagram :-



Initially we have to remove the unreachable states from the given DFA state diag w.k.T unreachable state is nothing but the state which is not reachable from the initial state.

Step 1:- from the above diagram all states are identified as reachable states. So that there is no change to delete the state from the given state diag.

Step 2:- After deleting unreachable state, construct the state transition table.

q	a	b
→ q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	q_4
*q_4	q_1	q_2

Step 3:- With the help of state equivalence method we can minimize the given DFA state diagram as shown below.

0-equivalence :- In this we can separate the entire sets into two sets, one set is

having non-finite states and another set having finite states.

$[q_0, q_1, q_2, q_3]$ $[q_4]$

Non-finite set finite set.

1-equivalence method :-

It is derived from 0 equivalence

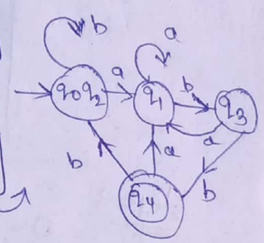
$[q_0\ q_1\ q_2]$ $[q_3]$ $[q_4]$

2-equi derived from 1-equi

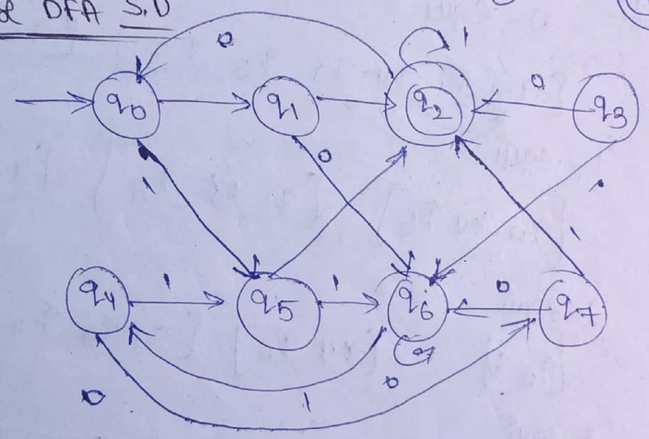
$[q_0\ q_2]$ $[q_1]$ $[q_3]$ $[q_4]$

3-equi derived from 2-eq

$[q_0\ q_2]$ $[q_1]$ $[q_3]$ $[q_4]$



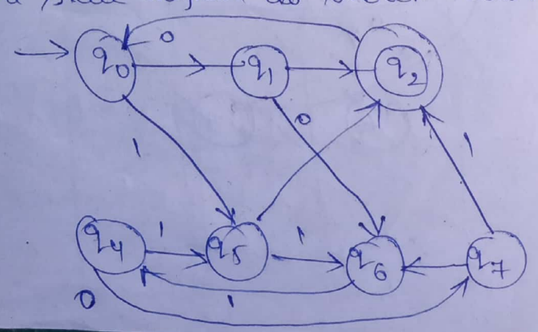
Minimise DFA S.D



Step 1 :-

$q_0\ q_1\ q_2$ (OR q_3) $q_4\ q_5\ q_6\ q_7$

After deleting the q_3 unreachable state, the resultant state diagram as shown below.



	0	1
→ q_0	q_1	q_5
q_1	q_6	q_2
* q_2	q_0	q_2
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

Step-3

0-equi

$[q_0\ q_1\ q_4\ q_5\ q_6\ q_7]$ $[q_2]$

1-equi

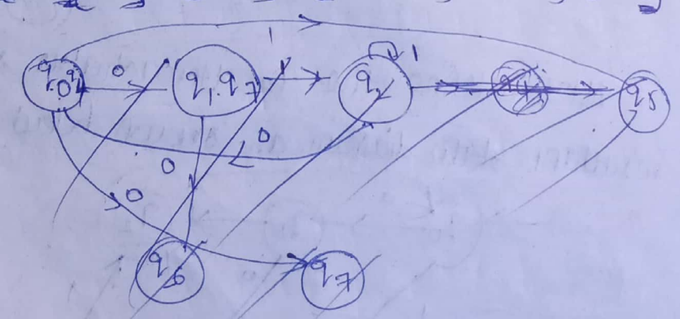
$[q_0\ q_4\ q_6]$ $[q_1\ q_7]$ $[q_2]$ $[q_5]$

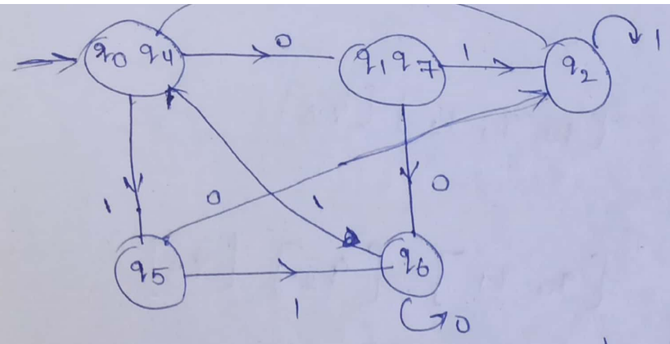
2-equi

$[q_0\ q_4]$ $[q_6]$ $[q_1\ q_7]$ $[q_2]$ $[q_5]$

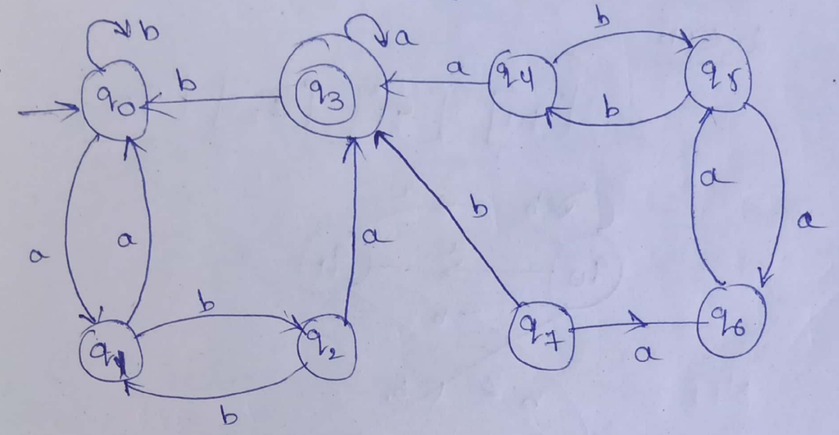
3-equi

$[q_0\ q_4]$ $[q_6]$ $[q_1\ q_7]$ $[q_2]$ $[q_5]$



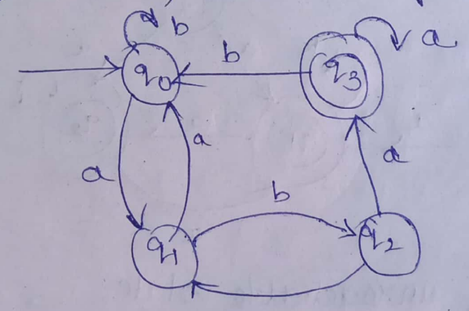


⇒



unreachable state q_4, q_5, q_6, q_7.

After deleting the state diag is

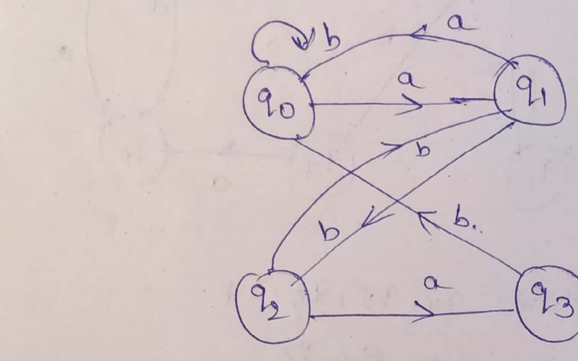


	a	b
→ q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
* q_3	q_3	q_0.

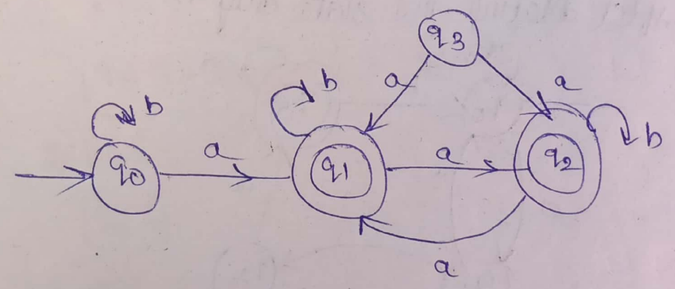
0-equi : $[q_0\ q_1\ q_2]\ [q_3]$

1-equi $[q_0\ q_1]\ [q_2]\ [q_3]$

2-equi $[q_0]\ [q_1]\ [q_2]\ [q_3]$

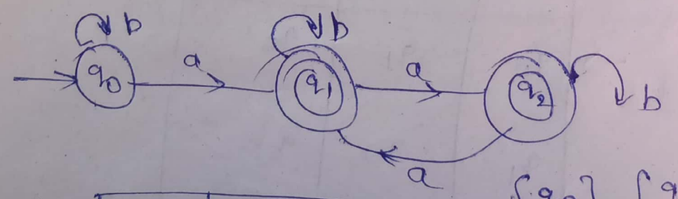


$\Rightarrow$



q_3 unreachable state

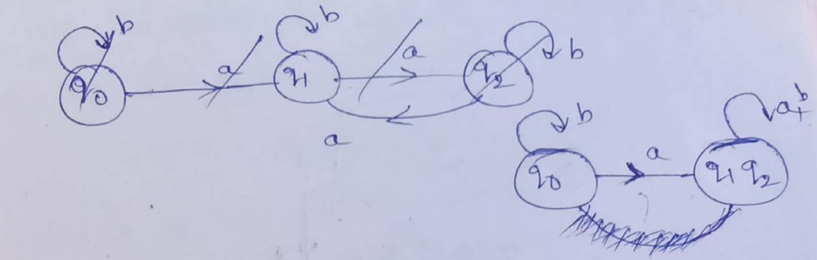
After deleting q_3 the state diagram is



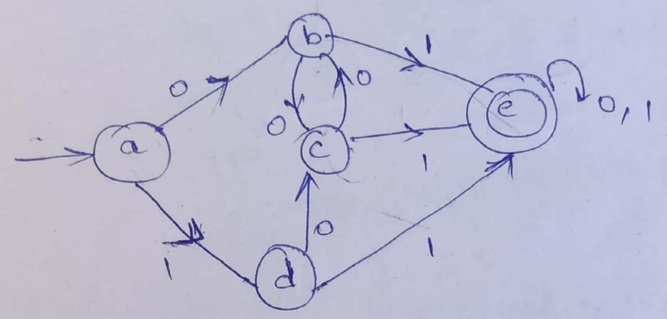
	a	b
→ q_0	q_1	q_0
* q_1	q_2	q_1
* q_2	q_1	q_2

0-ϵ $[q_0]\ [q_1\ q_2]$

1-ϵ $[q_0]\ [q_1\ q_2]$



⇒



No reachable state.

State transition table

	0	1
→a	b	d
b	c	e
c	b	e
d	c	e
*e	e	e

0-Eq :-

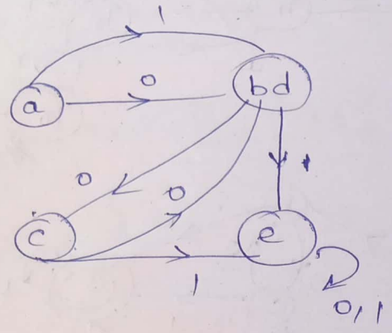
[a b c d] [e]

1-Eq :-

[a] [b d] [c] [e]

2-Eq :-

[a] [b d] [c] [e]



NFA :-

NFA stands for Non-deterministic finite Automata. The capabilities of both DFA and NFA are same i.e., the language which is accepted by DFA is also accepted by NFA and vice-versa.

$$L(DFA) = L(NFA)$$

⇒ In case of DFA every state should perform only one transition for every I/P symbol. In case of NFA every state should perform any number of transition (0 | 1 | >1)

⇒ The no. of states in NFA <= no. of states in DFA.

In case of NFA, there is no dead state but in case of DFA, there may be (or) may not be dead state.

⇒ we can represent the NFA in 5-tuple representation as shown below.

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

where Q is set of all sets

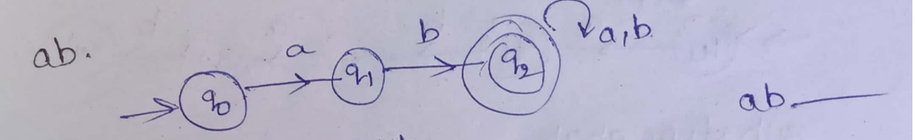
q_0 = Initial state

Σ = set of input symbols (or) I/p alphabet

δ = transition function, represented as $\delta : Q \times \epsilon \to 2^Q$

F = set of all final states, $F \subseteq Q$

Design NFA to accept ~~where~~ string starts with ab.

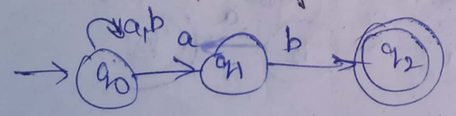


ab.

State transition table

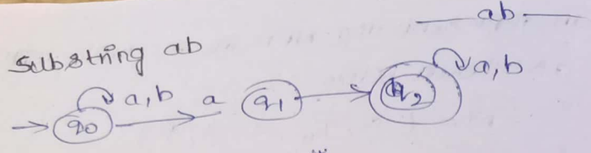
Q \ Σ	a	b
→ q_0	q_1	ϕ
q_1	ϕ	q_2
* q_2	q_2	q_2

where strings ends with ab.



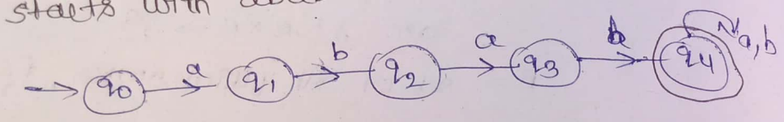
Q \ Σ	a	b
→ q_0	$\{q_0, q_1\}$	q_0
q_1	ϕ	q_2
* q_2	ϕ	ϕ

Substring ab

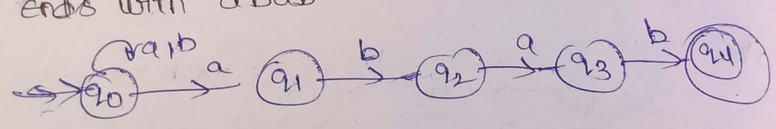


δ	a	b
q_0	$\{q_0, q_1\}$	q_0
q_1	ϕ	q_2
*q_2	q_2	q_2

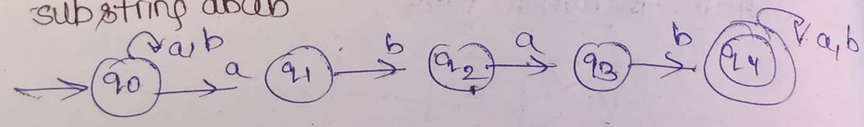
starts with abab



ends with abab



substring abab

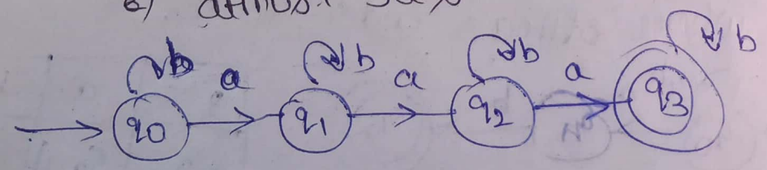


Design NFA over $\Sigma = \{a, b\}$ which every string contains a) exactly 3a's

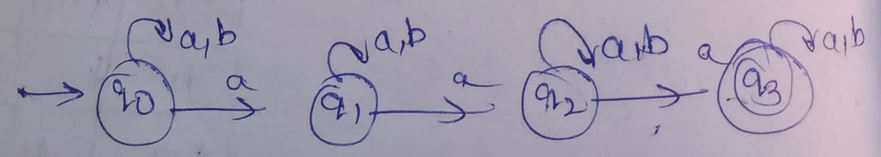
b) atleast 3a's

c) atmost 3a's

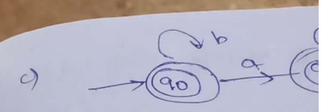
a)



b)



c)

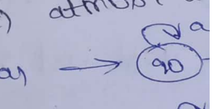


a) exactly ab's

b) atleast ab's

c) atmost ab's

a)



b)

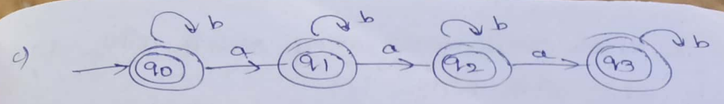
c)

design

ends with

sta

c)

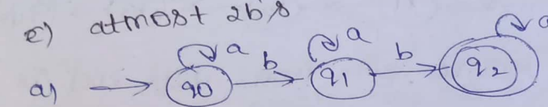


a) exactly 2b's

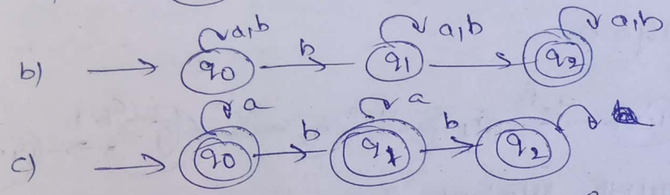
b) atleast 2b's

c) atmost 2b's

a)



b)



c)

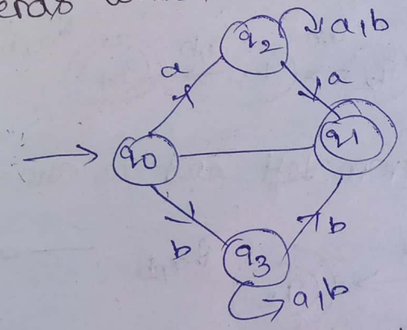
Design NFA, where every string starts and ends with same symbol.

a ——— a

b ——— b

a

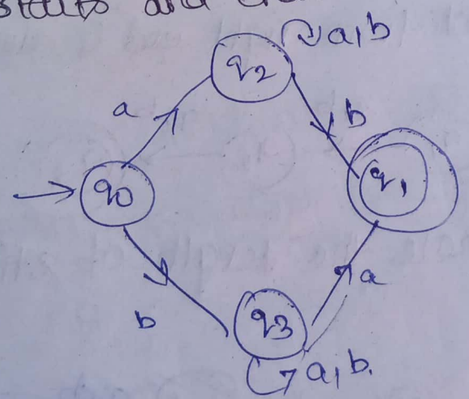
b



starts and ends with different symbol.

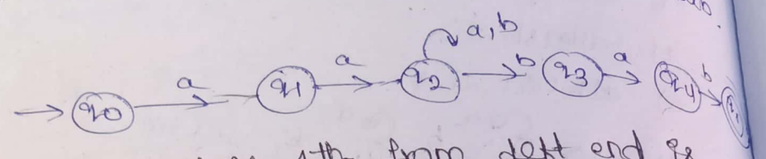
a ——— b

b ——— a



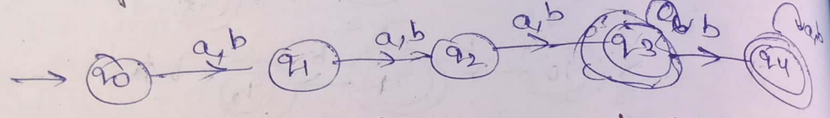
Design NFA where every string starts with aa and ends with bab.

aa ——— bab



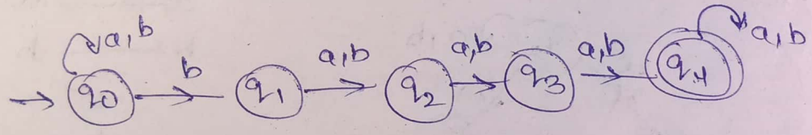
Design NFA whose 4th from left end is always b.

- - - b

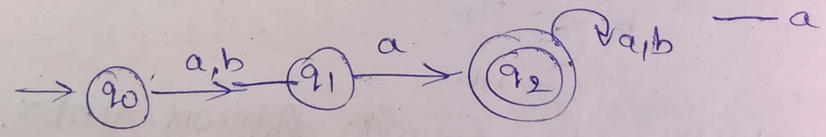


The fourth symbol from right end is always b.

b - - -

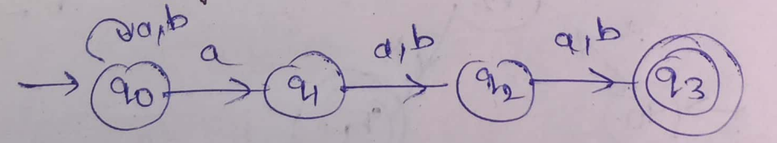


The second symbol from left end is always a

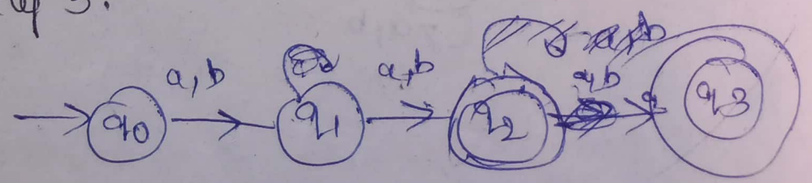


— a

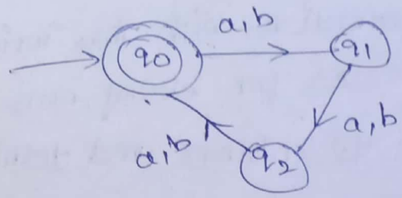
The 3rd symbol from right end is always a



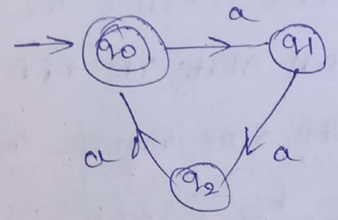
Design NFA whose the length of string is exactly 3.



Design NFA where the length of string is divisible by 3.



Design NFA where the no. of a's in a string is divisible by 3.



Equivalence between NFA and DFA :-

Here, equivalance b/w NFA and DFA is nothing but, we have to convert the NFA to DFA.

⇒ The process of converting NFA into DFA is called as subset construction.

⇒ W.K.T the no. of states in NFA is always lesser than equal to no. of states in DFA.

Procedure for conversion of NFA to DFA :-

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

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

1) Intial state is same for both DFA and NFA.

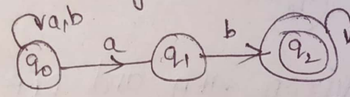
2) construction of δ' :-

$\delta'(q_0, q_1, q_2 \ldots q_n, a) = \delta(q_0, a) \cup \delta(q_1, a) \ldots \ldots \cup \delta(q_n, a)$

⇒ start the construction with the intial state & continue the process for every new state that appears in the I/P column and terminate the process whenever no new state appears in the input column.

3) every subset which contains the final state of NFA is a final state in DFA.

Convert the following NFA state diagram into DFA.



(1) W.K.T the intial state of NFA & DFA is same

(2) from the above NFA state diagram we can construct the DFA state transition table and then we can do the DFA state diagram from the derived DFA state transition diagram

(3) i.e., NFA SD → DFA STT → DFA SD.

DFA state transition table:

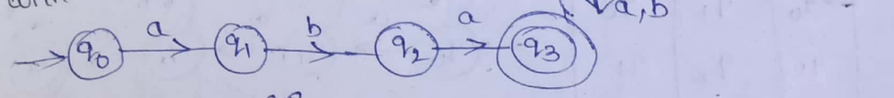
Σ / q	a	b
→ q_0	$\{q_0, q_1\}$	q_0
$q_0\, q_1$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$q_0\, q_2$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$
$q_0 q_1 q_2$	$\{q_0 q_1 q_2\}$	$\{q_0, q_2\}$

NFA:

Σ / q	a	b
q_0	$\{q_0, q_1\}$	q_0
q_1	ϕ	q_2
q_2	q_2	q_2

from the above DFA STT, we can design the state transition diagram as shown below:

Convert the NFA into DFA for the, every string starts with aba

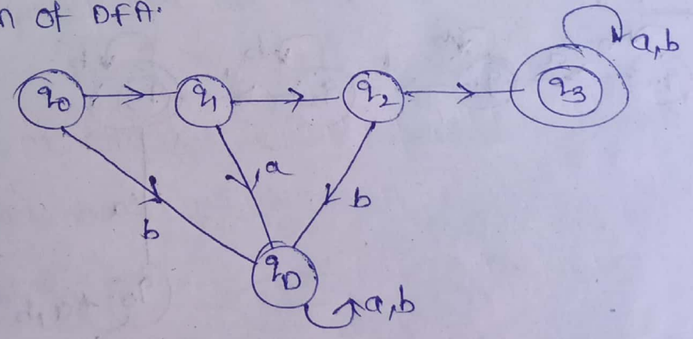


DFA state transition table

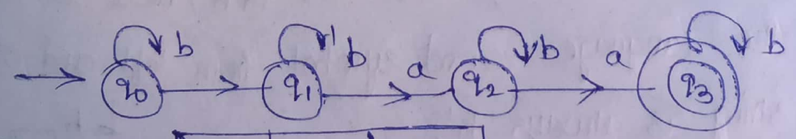
Q \ Σ	a	b
→ q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_0
* q_3	q_3	q_3
q_0	q_0	q_0

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

from above DFA STT, we can design the state diagram of DFA.



Conversion of NFA to DFA.

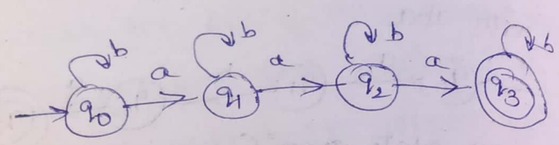


Q \ Σ	a	b
→ q_0	q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_2
* q_3	ϕ	q_3

Convert the following NFA STT into DFA SD.

NFA STT → NFA SD → DFA STT → DFA SD

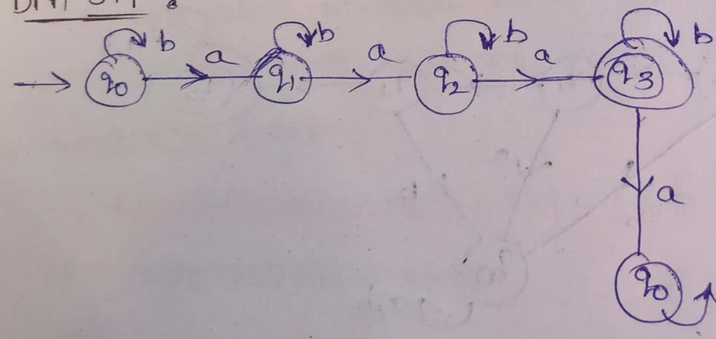
q / Σ	a	b
→ q_0	q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_2
* q_3	ϕ	q_3



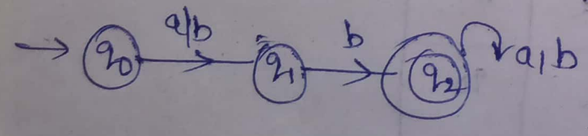
DFA STT

q / Σ	a	b
→ q_0	q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_2
* q_3	q_0	q_3
q_0	q_0	q_0

DFA STT :-



Convert the following NFA into DFA, here the NFA accept the language second symbol from left and of every string is always 'b'.



DFA STT :-

q / Σ	a	b
→ q_0		
q_1		
* q_2		
q_0		

DFA St

∈-N

→ NF

UC

'e

→

→

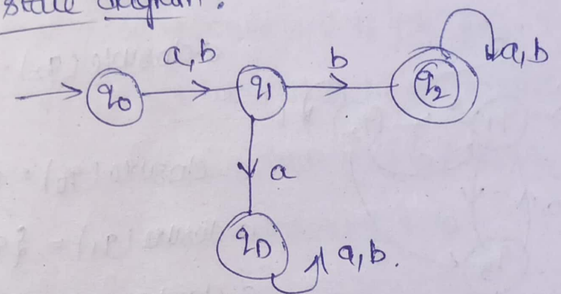
DFA STT :-

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

NFA STT :-

q	a	b
→ q_0	q_1	q_1
q_1	ϕ	q_2
* q_2	q_2	q_2

DFA state diagram :

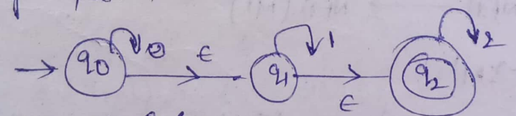


ϵ-NFA :-

→ NFA with ϵ-moves is called an ϵ-NFA that means we can move from one state to another state with 'ϵ' symbol but not with i/p symbol of the given

$$L = \{ 0^m 1^n 2^p \mid m, n, p \geq 0 \}$$

→ The above language it is represented in ϵ-NFA state diagram, as shown below.



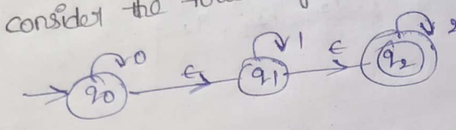
→ ϵ-NFA satisfies the all the properties of NFA along with ϵ-transition.

The capabilities of ϵ-NFA, NFA and DFA are same that means the language which is accepted by DFA also accepted by remaining NFA and ϵ-NFA, vice-versa. It can be denoted as

$$L(\epsilon\text{-NFA}) = L(\text{NFA}) = L(\text{DFA})$$

ϵ-closer :- The set of all the states that can be reached from the state q along ϵ labeld transition paths is called as ϵ-closel (q)

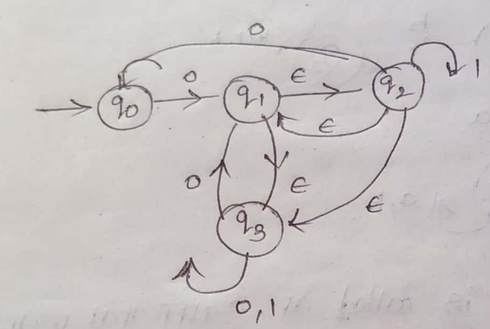
Consider the following ϵ-NFA state diagram.



ϵ-closure $(q_0) = \{q_0, q_1, q_2\}$

ϵ-closure $(q_1) = \{q_2, q_1\}$

ϵ-closure $(q_2) = \{q_2\}$



ϵ-closure $(q_0) = \{q_0\}$

ϵ-closure $(q_1) = \{q_1, q_2, q_3\}$

ϵ-closure $(q_2) = \{q_2, q_1, q_3\}$

ϵ-closure $(q_3) = \{q_3, q_2, q_1\}$

Conversion of ϵ-NFA into NFA

→ No change in the initial state

→ No change in the total number of states

→ May be change in the final states.

Processor :-

1. $(M)\ \epsilon - NFA \longrightarrow NFA\ (M')$

$M \rightarrow M'$

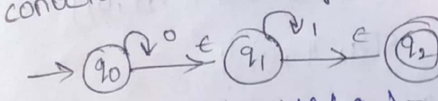
$M = (Q, \Sigma_0, F, \Sigma, \delta) \rightarrow M' = (Q', \Sigma_0', F', \Sigma, \delta')$

i. $\Sigma_0' = \Sigma_0$

2. $\delta'(q, x) = \epsilon\text{-closure}\{\delta(\epsilon\text{-closure}(q), x)\}$

3. final state : every state whose ϵ-closure contains the final state of ϵ-NFA is a final state in NFA.

convert the following ϵ-NFA state diagram to NFA.



ϵ-closure(q_0) = $\{q_0, q_1, q_2\}$

ϵ-closure(q_1) = $\{q_1, q_2\}$

ϵ-closure(q_2) = $\{q_2\}$

$\delta'(q_0, 0) = \epsilon$-closure $\{\delta(\epsilon\text{-closure}(q_0), 0)\}$

$= \epsilon$-closure $\{\delta(\{q_0, q_1, q_2\}, 0)\}$

$= \epsilon$-closure $\{q_0\} = \{q_0, q_1, q_2\}$

$\delta'(q_0, 1) = \epsilon$-closure $\{\delta(\epsilon\text{-closure}(q_0), 1)\}$

$= \epsilon$-closure $\{\delta(\{q_0, q_1, q_2\}, 1)\}$

$= \epsilon$-closure $\{q_1, q_2\}$

$= \{q_1, q_2\}$

$\delta'(q_0, 2) = \epsilon$-closure $\{\delta(\epsilon\text{-closure}(q_0), 2)\}$

$= \epsilon$-closure $\{\delta(\{q_0, q_1, q_2\}, 2)\}$

$= \epsilon$-closure $\{q_2\}$

$= q_2$

$\delta'(q_1, 0) = \epsilon$-closure $\{\delta(\epsilon\text{-closure}(q_1), 0)\}$

$= \epsilon$-closure $\{\delta(\{q_1, q_2\}, 0)\}$

$= \epsilon$-closure

$= \phi$

$\delta'(q_1, 1) = \epsilon$-closure $\{\delta(\epsilon\text{-closure}(q_1), 1)\}$

$= \epsilon$-closure $\{\delta(\{q_1, q_2\}, 1)\}$

$= \epsilon$-closure $\{q_1, q_2\}$

$= q_1, q_2$

$\delta'(q_1, 2) = \epsilon$-closure $\{\delta(\epsilon\text{-closure}(q_1), 2)\}$

$= \epsilon$-closure $\{\delta(\{q_1, q_2\}, 2)\}$

$= \epsilon$-closure $\{q_2\}$

$= q_2$

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

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

$= \epsilon\text{-closure}$

$= \phi$

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

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

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

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

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

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

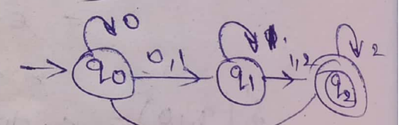
$= \phi$)*

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

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

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

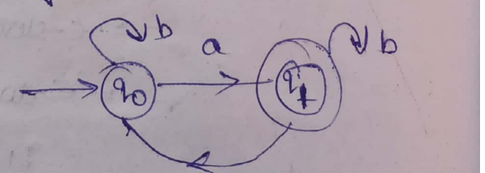
$= \{q_2\}$



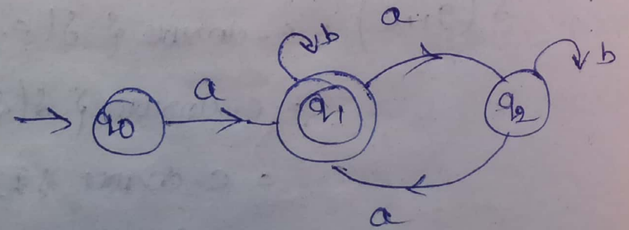
Equivalence b/w two finite Automata:

The two finite automata are equal when they are excepting the same language.

consider the following two finite automata M_1 and M_2. M_1 is



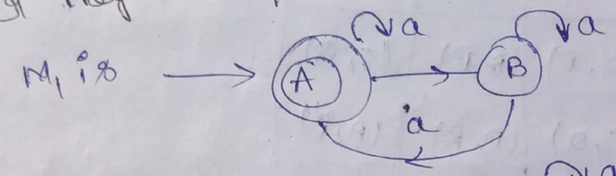
M_2 is



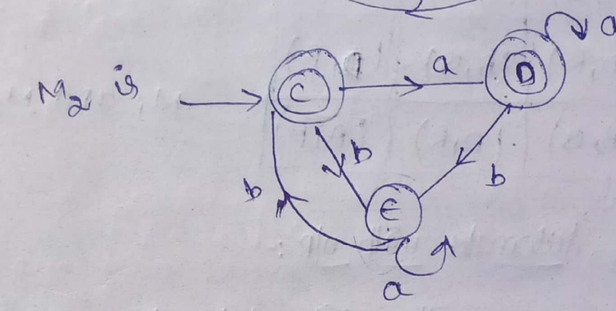
Here FA M_1 and FA M_2, both are equal bcz both are accepting the same language. Here in this language is the every string contains odd no of a's.

* consider the following FA M_1 and M_2 and check whether they are equal or not.

M_1 is



M_2 is

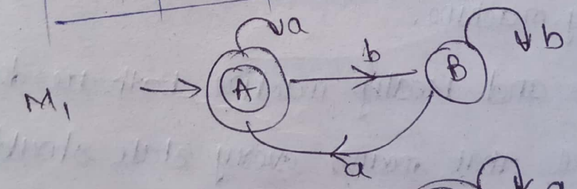


	a	b
(A, C)	(A, D)	(B, E)
(A, D)	(A, D)	(B, E)
(B, E)	(B, E)	(A, C)

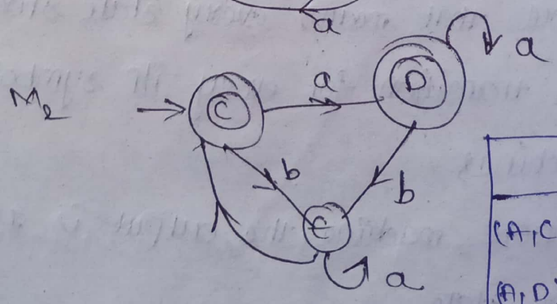
(f, f) } a, b should be this pair
(Nf, Nf)

M_1 and M_2 are equal.

$\Rightarrow$ M_1

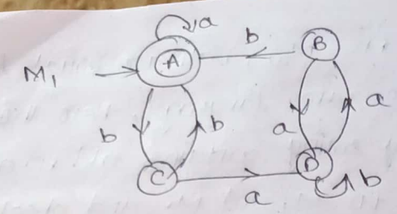


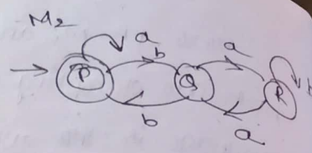
M_2



	a	b
(A, C)	(A, D)	(B, E)
(A, D)	(A, D)	(B, E)
(B, E)	(A, E)	(B, C)

M_1 and M_2 are not equal.





	a	b
(A, P)	(A, P)	(C, Q)
(C, Q)	(D, R)	(A, P)
(D, R)	(B, Q)	(D, R)
(B, Q)	(D, R)	(A, P)

M_1 and M_2 are equal.

Finite Automata with o/p :-

The finite automata which is having o/p without final states is called as finite automata with o/p.

~~A~~ with o/p classified with 2 categories

1) Moore machine
2) mealy machine.

⇒ Moore and Mealy machine, both are deterministic in nature that means every state should perform exact one transition for every i/p symbol in both the machines

⇒ In moore machine the output is associated with the state

→ In mealy machine the output is associated

with the i/p sym

⇒ Both moore an

6-tuple repres

the 6-tuple

M = (

where q_0 - i

Σ = input

Δ = o/p

λ is the

as λ : Q

The 6-tup

M = (q_0,

q_0 - initi

λ is t

as sho

Examples

no. of

when

indica

string

Σ = {

with the i/p symbol in the time of transition.

⇒ Both moore and mealy machine is represented with 6-tuple representation.

the 6-tuple representation of moore machine is

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

where q_0 - intial state, Q = set of all states,

Σ = input alphabet, δ = state transition function represented as $\delta : Q \times \Sigma \to Q$.

Δ = o/p alphabet

λ is the o/p transition function, is represented as $\lambda : Q \to \Delta$

The 6-tuple representation of mealy machine is

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

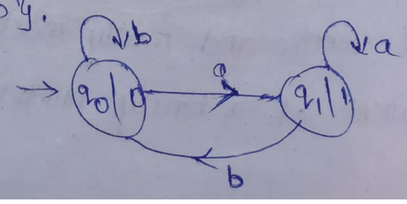
q_0 - intial state, Q = set of all states ↑ same above

λ is the o/p transition function, it is represented as shown below $\lambda : Q \times \Sigma \to \Delta$

Examples on Moore machine :-

Q- Design Moore machine to count the no. of a's in a string of a language.

when ever 'a' is occured in this string it is indicated with 1 whenever b is occured in the string it is indicated with 0 over the alphabet $\Sigma = \{a, b\}$.



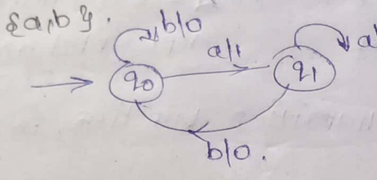
Q \ Σ	a	b	Δ
→ q_0	q_1	q_0	0
q_1	q_1	q_0	1

aaabba = 1.

$\rightarrow q_0 \xrightarrow{a/0} q_1 \xrightarrow{a/1} q_1 \xrightarrow{a/1} q_1 \xrightarrow{b/0} q_0 \xrightarrow{b/0} q_0 \xrightarrow{a/0} q_1$

Example of mealy

Design mealy machine to count the no. of a's in a string of a language over the alphabet $\Sigma = \{a, b\}$.



δ	a	1	b	1
$\rightarrow q_0$	q_1	1	q_0	0
q_1	q_1	1	q_0	0

Design moore and mealy machines to count the occurances of a b in a string over the alphabet $\Sigma = \{a, b\}$, $\Delta = \{0, 1\}$

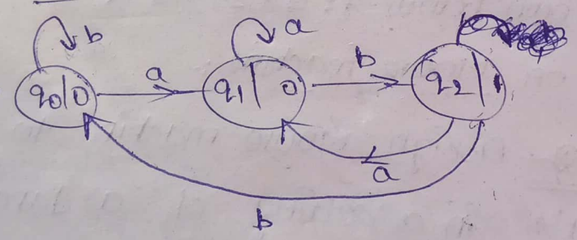
ab ab ba = 2 string

ab-1

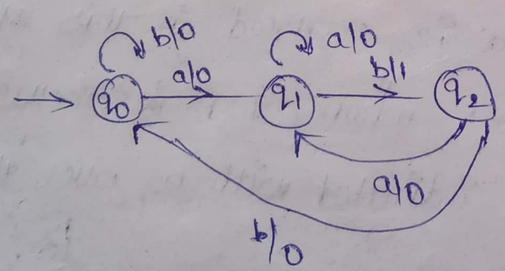
abe

ab ab a ab a = 3 string 2.

Moore :-



Mealy :-



Design the moore and mealy machine to implement the complement of a binary number over $\Sigma = \Delta = \{0, 1\}$

MOORE

mealy

The eq

ie., both the

for the sam

1) convers

2) convers

Convers

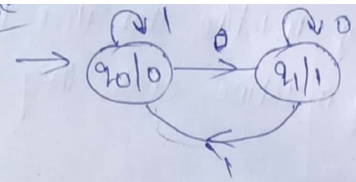
moore

mealy

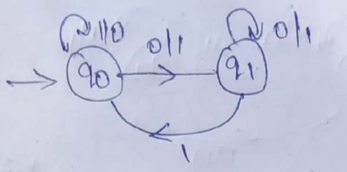
in moo

conver

Moore



mealy



The equivalent of moore and mealy machine i.e., both the machines generate the same o/p stream for the same i/p stream.

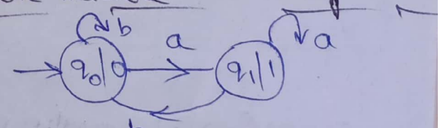
1) conversion from moore to mealy machine
2) conversion from mealy to moore machine

Conversion from moore to mealy machine :-

While we are performing conversion from moore to mealy machine, how many no. of states in moore machine the same no. of states are present in mealy machine i.e., there is no change in no. of states in moore machine to mealy machine.

Convert the following moore machine into mealy machine

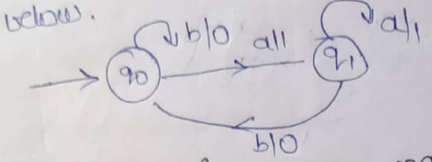
moore ⇌ mealy



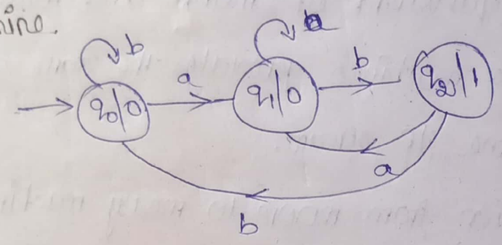
from the above moore machine diag we can draw the state transition table for mealy machine. as shown below.

	a →		b →	
→q_0	q_1	1	q_0	0
q_1	q_1	1	q_0	0

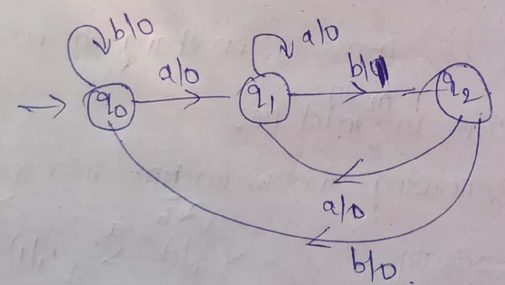
from the above mealy machine transition table we can draw the mealy machine state diag as shown below.



Convert the following moore machine into mealy machine.



	a		b	
	q_1	0	q_0	0
$\rightarrow q_0$	q_1	0	q_2	1
q_1	q_1	0	q_2	1
q_2	q_1	0	q_0	0

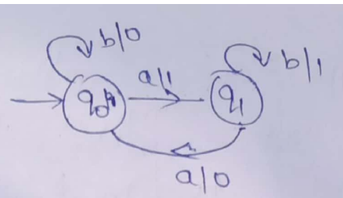


Conversion of mealy machine to moore machine:

Here the no. of states may be change from mealy machine to moore machine

(1) Design the moore machine state diag from the below mealy machine state diag.

	a		b	
→q_0	q_1	1	q_0	0
q_1	q_0	0	q_1	1

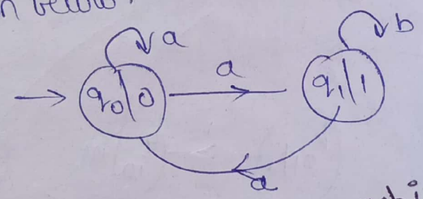


from the above state diag we can derive the mealy machine state transition table

→ from the above mealy state transition table, we can construct the moore machine state transition table,

	a	b	
→q_0	q_1	q_0	0
q_1	q_0	q_1	1

from the moore state transition table we can draw the moore machine state diag as shown below.



(2) Draw the moore machine by using the below mealy state transition table.

mealy

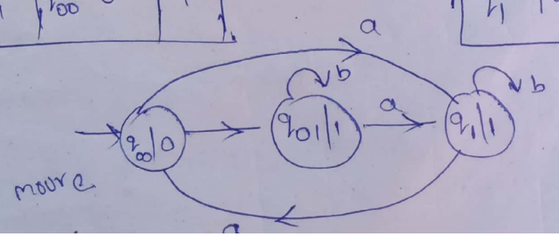
δ	a	λ	b	λ
→q_0	q_1	1	q_{01}	1
q_1	q_{00}	0	q_1	1

$q_0 \rightarrow q_{00}$, q_{01}

$q_0 = q_{00} = q_{01}$

moore

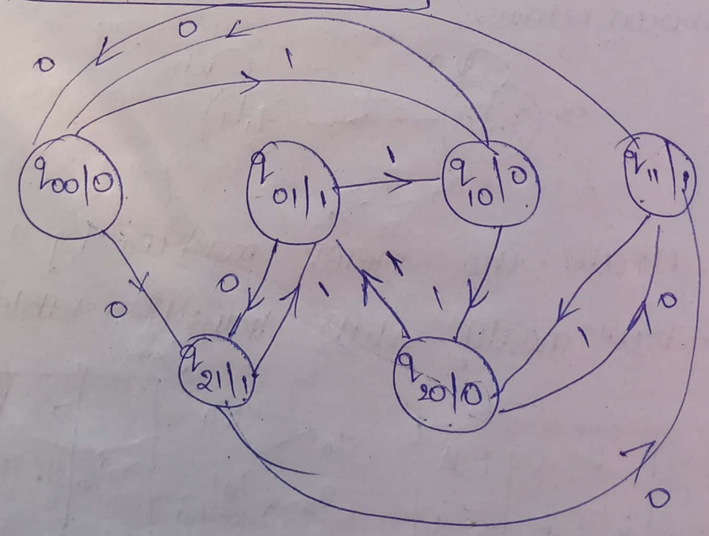
	a	b	λ
→q_{00}	q_1	q_{01}	0
q_{01}	q_1	q_{01}	1
q_1	q_{00}	q_1	1



(3)

	0	1	1	1
$\rightarrow q_0$	q_{21}	1	q_{10}	0
q_1	q_{00}	0	q_{20}	0
q_2	q_{11}	1	q_{01}	1

	0	1	1
$\rightarrow q_{00}$	q_{21}	q_{10}	0
q_{01}	q_{21}	q_{10}	1
q_{10}	q_{00}	q_{20}	0
q_{11}	q_{00}	q_{20}	1
q_{21}	q_{11}	q_{01}	1
q_{20}	q_{11}	q_{01}	0



UNIT-II

REGULAR EXPRESSION (RES)

The expression which generates regular languages is called Regular expression.

For constructing the regular expression, we can use the following three operators.

Here '*' indicate the kleendouble values are ranging from 0, 1, 2 . . . .

'+' indicates the union operation

'.' indicates the concatenation operations

The values of + ranging from 1, 2 . . . .

Regular expression is indicated with the letter capital 'R'.

consider the following RE's and generates the corresponding Regular language

1) $R = a+b$

$L(R) = \{a, b\}$

2) $R = (a+b)b = ab+bb$

$L(R) = \{ab, bb\}$

3) $R = a^*$

$= a^0 + a^1 + a^2 + a^3 + \ldots$

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

4) $R = (ab)^*$

$R = (ab)^0 + (ab)^1 + (ab)^2 + (ab)^3 + \ldots$

$L(R) = \{\epsilon, ab, abab, ababab\}$

5) $R = (a+b)^*$

$= (a+b)^0 + (a+b)^1 + (a+b)^2 + \ldots$

$L(R) = \{\epsilon, a, b, aa, ab, ba, bb \ldots\}$

Construct the Regular expression that generates all the strings of a's and b's.

a) Including ϵ

b) excluding ϵ.

Including ϵ $R = (a+b)^*$

excluding ϵ $R = (a+b)^+$

construct RE that generates all the strings of a's and b's where every string starts with a.

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

a ——— a(a+b)*

construct RE that generates all the strings of a's and b's where every strings ends with ba.

——— ba

$R = (a+b)^* ba$

where every string starts and ends with a

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

a ——— a, a

where every string starts and ends with same symbol.

a, b, a ——— a, b ——— b

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

where every string starts and ends with different symbols.

a ——— b, b ——— a

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

where the length of string is exactly 3.

$R = (a+b)(a+b)(a+b) = (a+b)^3$

a/b a/b a/b

where the length of the string is atleast 3.

$R = (a+b)^3 (a+b)^*$

where the length of the string atleast 3.

$$R = (a+b)^3.$$

where the length of the string is divisible by 3.

$$R = (a+b)^{3*}$$

(or) $\left((a+b)^3\right)^*$

Length of the string is 2 mod 3.

$$R = (a+b)^2\left((a+b)^3\right)^*$$

where the third symbol from last end is always a.

$$R = (a+b)^2 a\,(a+b)^*$$

The third symbol from right end is always b.

$$R = (a+b)^*\, b\,(a+b)^2$$

where every string starts with ab and ends with ba

$$R = ab\,(a+b)^*\,ba$$

where every strings contains exactly 2a's.

$$R = b^*\,a\,b^*\,a\,b^*$$

atleast 2a's

$$R = (a+b)^*\,a\,(a+b)^*\,a\,(a+b)^*$$

Atmost 2a's

$$R = b^* + b^*a\,b^* + b^*\,a\,b^*\,a\,b^*$$

Identity Rules :-

Let 'P', 'Q', and 'R' are Regular expressions then the identity Rules are given below.

(RE)

1. $\epsilon \cdot R = R \cdot \epsilon = R$
2. $\phi \cdot R = R \cdot \phi = \phi$
3. $\phi + R = R$
4. $R + R = R$
5. $(P+Q) \cdot R = P \cdot R + Q \cdot R$
6. $\epsilon^* = \epsilon$
7. $\phi^* = \epsilon$
8. $R \cdot R^* = R^* \cdot R = R^+$
9. $\epsilon + R \cdot R^* = R^*$
10. $[R^*]^* = R^*$
11. $(R+\epsilon)^* = R^*$
12. $\epsilon + R^* = R^*$
13. $R^* \cdot R + R = R^*$
14. $(P \cdot Q)^* \cdot P = P(Q \cdot P)^*$
15. $(P+Q)^* = (P^* \cdot Q^*)^* = (P^* + Q^*)^*$
16. $R^* \cdot (\epsilon + R) = (\epsilon + R) \cdot R^* = R^*$

Equivalence between two Regular Expressions :-

If two RE's representing same set of strings then they are equal.

Problems :-

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

L.H.S $= \epsilon + \underbrace{1^*(011)^*}_{R}\left(\underbrace{1^*(011)^*}_{R}\right)^*$

$= (1^*(011)^*)^*$ (from $\epsilon + RR^* = R^*$)

$= (1+011)^*$ (from $(P^*.Q^*)^* = (P+Q)^*$)

2. $(1+00^*1) + (1+00^*1)(0+10^*1)^*(0+10^*1)$

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

$= (1+00^*1)[\epsilon + \frac{(0+10^*1)(0+10^*1)}{R}]$ ($\epsilon + R^*R = R^*$)

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

$= \cancel{\epsilon + 0}\ (\epsilon + 00^*)1(0+10^*1)^*$

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

3. S.T $(0^*1^*)^* = (0+1)^*$

$= (0^*.1^*)^*$

$(0+1)^* = \{\epsilon, 0, 1, 01, 10, 00 \ldots\}$

$\epsilon \rightarrow$

$0 - (0^*1^*)'$ $\quad 1 - (0^*1^*)'$

$= 0'1^\circ$ $\quad - 0^\circ 1'$

$= 0.\epsilon$ $\quad - \epsilon.1$

$= 0$ $\quad = 1$

The same set of the strings generate by the regular expression,

4. $(R+S)^* \neq (R^* + S^*)$

$(R+S)^* = \{\epsilon, R, S, RR, RS \ldots\}$

$R^* + S^* = \{\epsilon, RR, RRR, \ldots\} + \{\epsilon, S, SS, SSS \ldots\}$

$=$

In R.H.S RE we can't get the strings with the combinations of R and S. But these type of ~~exp~~ strings are obtained from R.E. the strings which are obtained from one regular

expression is not obtained from another RE, so that RE's are not equal.

(1) Finite Automata to Regular exp conversion :-

Here we can convert the FA into RE with the help of Arden's theorem

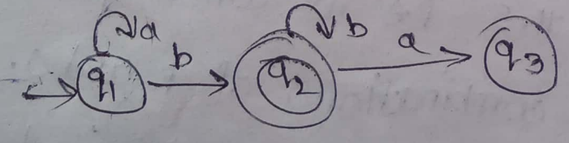
Def. of Arden's theorem

consider P and Q be the two RE's over the set Σ, the RE are 'R' is obtained as

$$R = Q + RP$$

$$\Rightarrow R = QP^*$$

Construct the RE from given DFA



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

$RE = q_2$, This is the Required RE

should be derived from the given DFA.

$q_1 = q_1 a + \epsilon$

$\boxed{q_1 = \epsilon + q_1 a} \Rightarrow R = \epsilon + RP$

$\Rightarrow R = \epsilon P^*$

$q_1 = \epsilon \cdot a^*$

$= a^*$

$q_2 = q_2 b + q_1 b$

$q_2 = \underset{\epsilon}{a^* b} + \underset{RP}{q_2 b}$

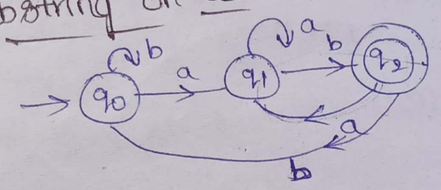
$= a^* b b^*$ (from 8)

$q_2 = a^* b^+$

$RE = a^* b^+$

ab

substring on ab



$RE = q_2$

$q_2 = q_1 b$

$q_1 = q_0 a + q_1 a + q_2 a$

$q_0 = q_0 b + q_2 b + \epsilon$

$q_2 = q_1 b$

Obtain the RE for the following DFA

$RE = q_1 + q_2$

$q_1 = q_1 0 + \epsilon$

$\frac{q_1}{R} = \underset{Q}{\epsilon} + \underset{RP}{q_1 0}$ $(R = QP^*)$

$q_1 = \epsilon \cdot 0^* = 0^*$

$\boxed{q_1 = 0^*}$

$q_2 = q_1 1 + q_2 1$

$\frac{q_2}{R} = 0^* 1 + \frac{q_2}{RP} 1$

$q_2 = 0^* 1 . 1^* = 0^* 1^+$

$\boxed{q_2 = 0^* 1^+}$

$RE = q_1 + q_2$

$= 0^* + 0^* 1^+$

$\boxed{RE = 0^* (\epsilon + 1)^+}$

Regular expression to FA conversion:-

Here, we can convert the RE into FA in the following scenario.

There are two methods to convert the RE into FA

RE → ENFA
→ NFA
→ DFA

1) Method of synthesis
2) Method of decomposition (or) direct method (or) subset.

Method of synthesis

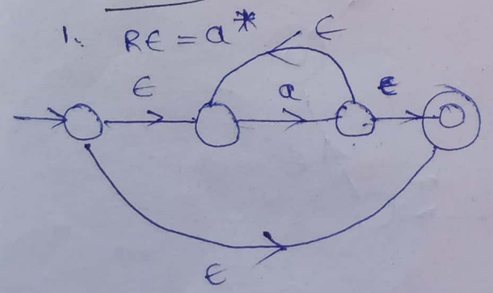
In method of synthesis we can consider the ENFA by using the following 3 properties

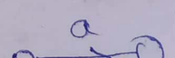
1. Kleen closure (*)
2. union (+)
3. concatination (.)

Examples:-

R.E → NFA

1. $RE = a^*$



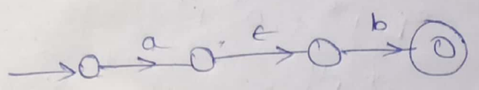
1)


2)

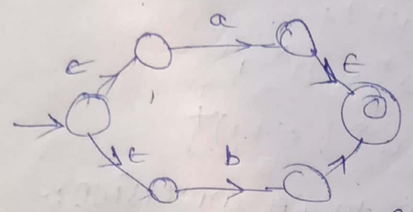

3)

4)

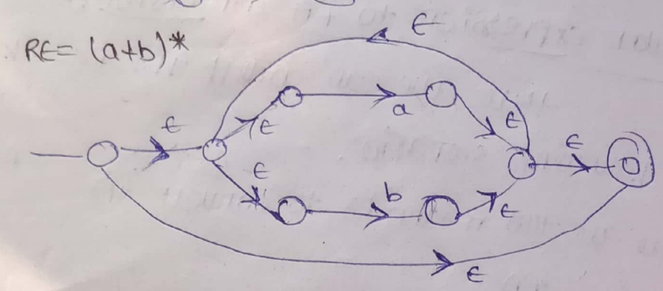

2) RE = a.b.



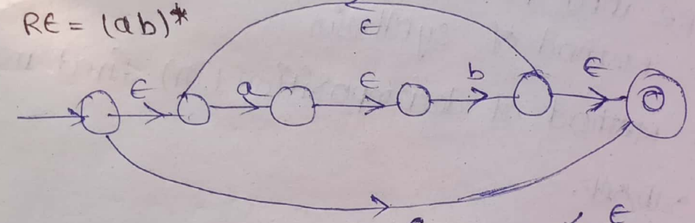
3) RE = a+b



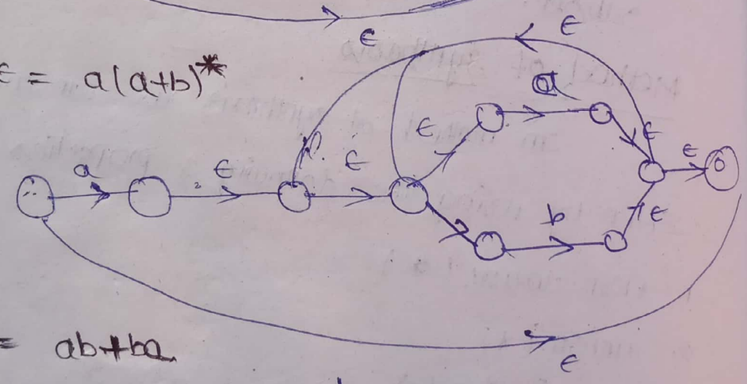
4) RE = (a+b)*



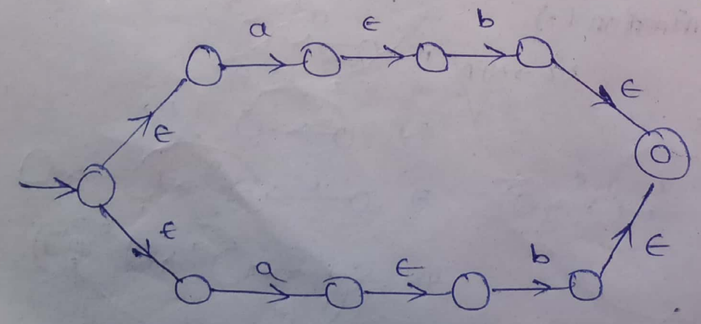
5) RE = (ab)*



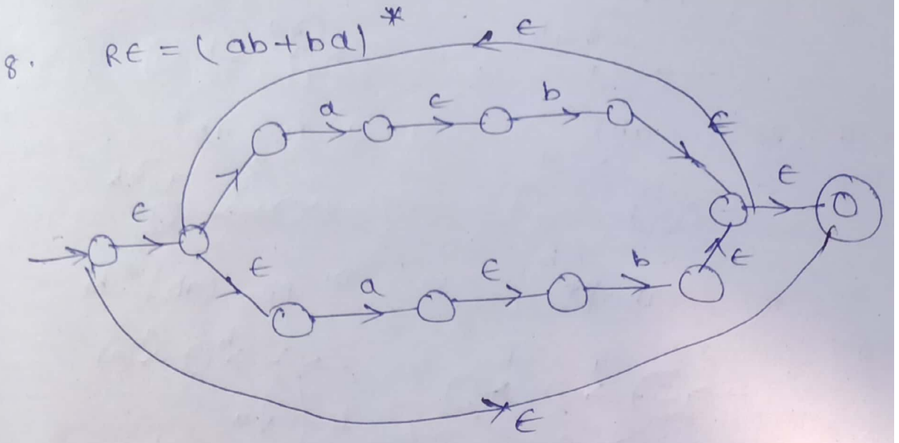
6) RE = a(a+b)*



7) RE = ab+ba



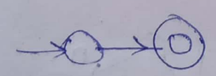
8. RE = (ab+ba)*



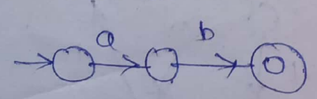
Method of decomposition

By using this method we can convert the regular expression into NFA and then convert into DFA.

Ex:- 1. RE = a

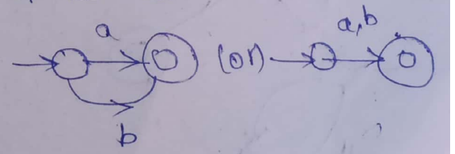


2. RE = ab

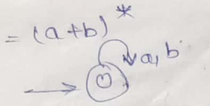


3. RE = a*

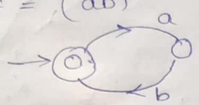
4. RE = a+b



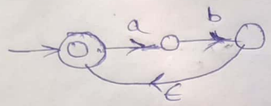
5. RE = (a+b)*



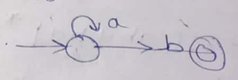
6. RE = (ab)*



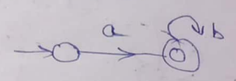
7. RE = (abc)*



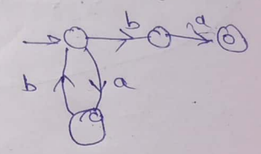
8. RE = a*b



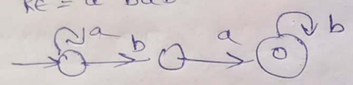
9. RE = ab*



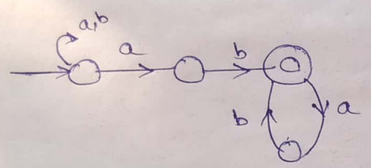
10. RE = (ab)* ba



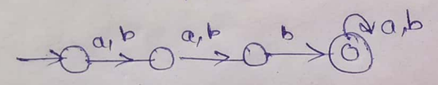
11. RE = a*bab*



12. RE = (a+b)*ab(ab)*

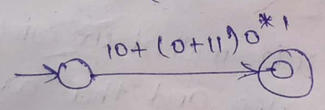


13. RE = (a+b)(a+b)b(a+b)*

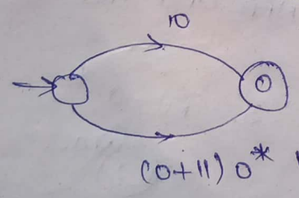


14. RE = 10 + (0+11)0* 1

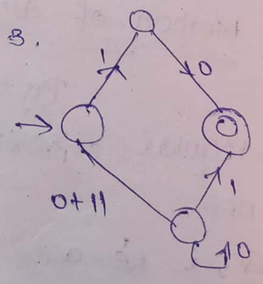
1.



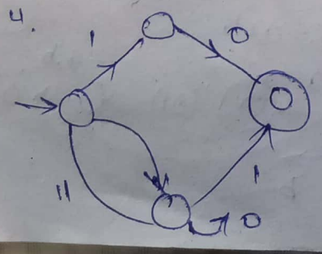
2.



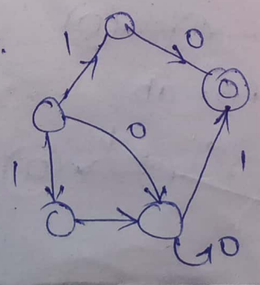
3.



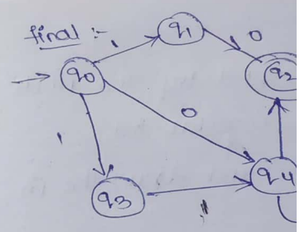
4.



5.



final :-



Closure Properties

We can

Performing an

regular langu

1) The regular

of two regu

2) The regular

- on of two

3) The regu

- ent of RL

4) we can

operation

5) The conca

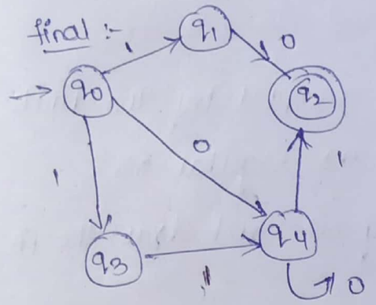
6) The reple

7) The

8) The c

RL.

final :-



closure properties of regular language :-

We can obtain a regular language by performing an operation on one (or) more regular languages

1) The regular language 'L' is obtained by union of two regular languages i.e., $L = L_1 \cup L_2$

2) The regular language 'L' is obtained by intersecti -on of two regular lang i.e., $L = L_1 \cap L_2$

3) The regular language 'L' is obtained by complim -ent of RL i.e., $L = \overline{L_1}$

4) we can obtain the RL by performing difference operation i.e., $L = L_1 - L_2$ (or) $L = L_2 - L_1$

5) The concatination of RL is regular. i.e., $L = L_1 \cdot L_2$

6) The reverse of RL is Regular i.e., $L = Rev(L_1)$

Ex :- $L_1 = \{ab, ba\}$ - RL

$\Rightarrow L = Rev(L_1) = \{ba, ab\}$ - RL

7) The homomphism of RL is also a RL

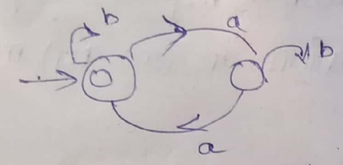
8) The inverse homorphism of RL is also a RL.

Regular Set :-

The set which is accepted by the finite Automata (FA) is called as Regular set.

Q- consider the following set and draw the FA which accepts the set.

$RS = \{ \epsilon, aa, aaaa, aaaaaa \ldots \}$



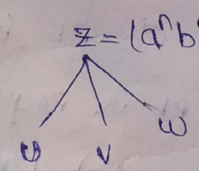
✱ Pumping ~~Lemma~~ Lemma of RL :-

Pumping Lemma is used for the given language is not RL. to prove

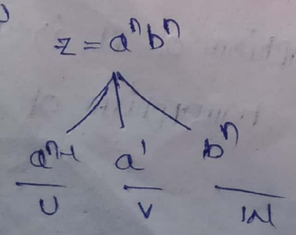
Ex :- $L = \{ a^n b^n \mid n \geq 1 \}$

→ Initially, the pumping lemma consider the given language as RL and finally the language is proved as not RL.

→ The string 'z' is split into 3 parts u, v, w



i.e., $z = u.v.w$ It can be explained as shown below



If we prove $uv^iw \notin L$; $i \geq 2$ (for atleast any value of i).

$a^{n-1}(a^1)^i b^n$

$a^{n-1} a^i b^n$

let $i=2 \Rightarrow a^{n-1} a^2 b^n$

$\Rightarrow a^{n+1} b^n$

let n = any value

n=1

$a^2 b^1$

$aab \notin L$.

Regular Grammar:

The grammar which generates the RL and recognised by the FA is called as RG.

It is represented as 'G'

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

V – Set of non-terminals

T – set of terminals

P – Production rules

S – start symbol.

Note:-

→ In RG, capital letters indicates the non-terminals and terminals are indicated by lower case letters, symbols and epsilon 'ε'.

→ only one non-terminal is present on the left of the production.

→ The non-terminal can be present either left side or right side or both.

Terminal can be present only in the right side of the production.

→ Non-terminal is also called as Variable.

ex:- $S \to 0S$

$S \to 1B$

$B \to \epsilon$

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

$V = \{S, B\}$

$T = \{0, 1, \epsilon\}$

$P: S \to 0S,\ S \to 1B\ ;\ B \to \epsilon$

$S : S$

ex:- $A \to BC$

$B \to 0C \mid 1B \mid \epsilon$

$C \to 01D \mid \epsilon$

$V = \{A, B, C\}$

$T = \{0, 1, \epsilon\}$

$P = A \to BC\ ;\ B \to 0C \mid 1B \mid \epsilon\ ;\ C \to 01D \mid \epsilon$

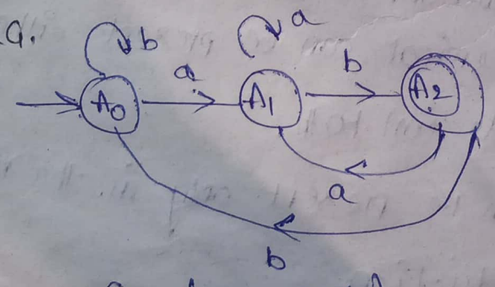
$S = A$

Regular Grammar and finite Automata:

We can convert the RG into FA as well as FA into RG.

FA to RG :-

consider the following state diag and convert it into RG.



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

$V = \{A_0, A_1, A_2\}$

$T = \{a, b\}$

$S = A_0$

P: $A_0 \to aA_1$

$A_0 \to bA_0$

$A_1 \to aA_1$

$A_1 \to bA_2$ final state

$A_1 \to b$

$A_2 \to aA_1$

$A_2 \to bA_0$.

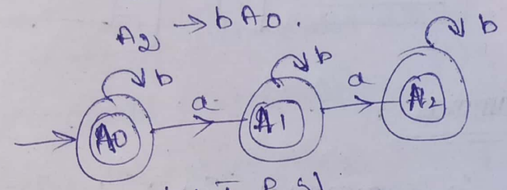
P

$\Rightarrow A_0 \to aA_1 | bA_0$

$A_1 \to aA_1 | bA_2 | b$

$A_2 \to aA_1 | bA_0$.

$\Rightarrow$



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

$V = \{A_0, A_1, A_2\}$

$S = A_0$

$P = A_0 \to aA_1$

$A_0 \to bA_0$

$A_1 \to aA_2$

$A_1 \to bA_1$

~~$A_2 \to aA_2$~~

$A_2 \to bA_2$

$A_0 \to a$

$A_0 \to b$

$A_\phi \to a$

$A_\phi \to b$

~~$A_\phi \to a$~~

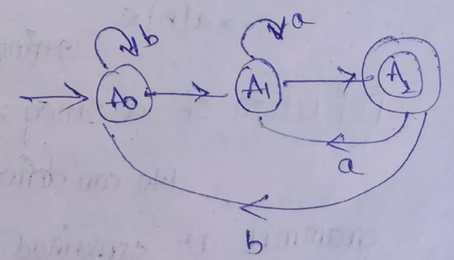
$A_2 \to b$.

RG to FA

$A_0 \to aA_1 | bA_0$

$A_1 \to aA_1 | bA_2 | b$

$A_2 \to aA_1 | bA_0$



Regular expression and regular Grammar :-

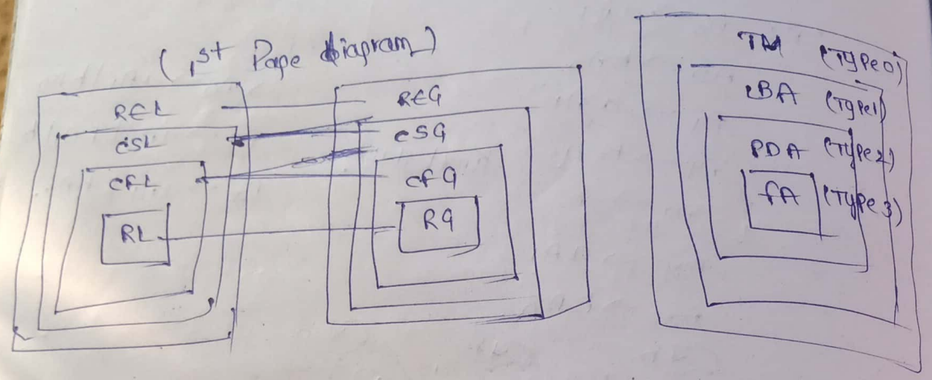
$RE \rightleftharpoons RG$

1) $RE \to RG$ $\quad RE \to FA \to RG$

2) $RG \to RE$ $\quad RG \to FA \to RE$.

UNIT-3

CONTEXT FREE GRAMMAR



Context free Grammar :-

CFG is represented with 'G'

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

V = set of variables (or) non-terminals

T = set of terminals

P = set of Production rules

S = starting symbol.

Ex :-

$S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow a|b|\epsilon$

~~string :- aabbabba~~

Derivation of a string :-

We can derive the string from the grammar as explained below.

Consider the above grammar and derive the string

string :- aabbabba.

W.K.T the string derivation is always start

with starting symbol only.

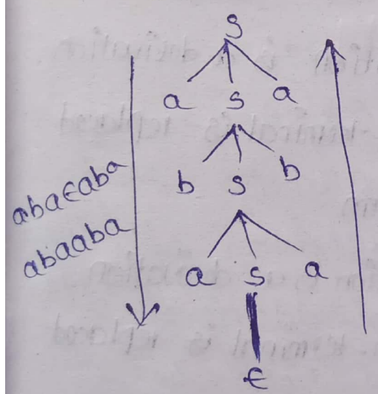
1) String :- abaaba.

⇒ S
⇒ aSa [S→aSa]
⇒ abSba [S→bSb]
⇒ abasaba [S→aSa]
⇒ abaεaba [S→ε]
⇒ abaaba.

Parse Tree :-

It is also called as derivation tree. It is the graphical representation of how we are deriving the string from the given grammar.

ex :- CFG G is S→aSa
S→bSb
S→a|b|ε



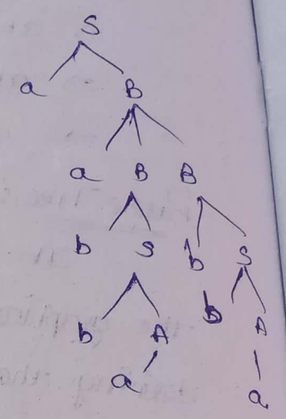
S
aSa [S→aSa]
abSba [S→bSb]
abasaba [S→aSa]
abaεaba [S→ε]
abaaba.

Properties Parse String :-

1) The root node indicates the start symbol
2) The derivation is read from Left to right.
3) The leaf nodes are always terminal nodes.
4) The internal node is always will be the non-terminal nodes.

2). $S \to aB \mid bA$
$A \to a \mid aS \mid bAA$
$B \to b \mid bS \mid aBB$
String :- aabbabba

$\Rightarrow S$
$\Rightarrow aB$ ($S \to aB$)
aaBBB ($B \to aBB$)
aabSB ($B \to bS$)
aabbAB ($S \to bA$)
aabbaB ($A \to a$)
aabbabS ($B \to bS$)
aabbabbA ($S \to bA$)
aabbabba ($A \to a$)



Left Most derivation & Right Most derivation :-

The left most derivation is a derivation in which the left most non-terminal is replaced first from the contentiallyial form

The Right most derivation is a derivation in which the right most non-terminal is replaced first from the sentential form.

1) String : aba

$S \Rightarrow XYX$
$X \to a$
$Y \to b$

LMD	RMD
$\Rightarrow S$	$\Rightarrow S$
$\Rightarrow XYX$	$\Rightarrow XYX$
$\Rightarrow aYX$	$\Rightarrow XYa$
$\Rightarrow abX$	$\Rightarrow Xba$
$\Rightarrow aba$	$\Rightarrow aba$

2) $S \to AA$
$A \to aB$
$B \to bB$
String : a

3) $S \to$
$A \to$
$B \to$

$\Rightarrow$ aabbS
$\Rightarrow$ aabbaB
$\Rightarrow$ aabbabA
aabbaba

2) S → AA
A → aB
B → bB | ϵ
string : abba

LMD

⇒ S
⇒ AA [A → aB]
⇒ aBA
⇒ abBA [B → bB]
⇒ abbBA [B → bB]
⇒ abbϵA [B → ϵ]
⇒ abbaB [A → a]
⇒ abbaϵ [B → ϵ]
⇒ abba

RMD

⇒ S
⇒ AA
⇒ AaB
⇒ AabB
⇒ AabbB
⇒ Aabbϵ
⇒ Aa
⇒ aBa
⇒ abBa
⇒ abbBa
⇒ abbϵa
⇒ abba.

3) S → aB | bA
A → a | as | bAA
B → b | bs | aBB

string aa bbabba

LMD

⇒ S
⇒ aB [B → aBB]
⇒ aaBB [B → bS]
⇒ aabSB [S → bA]
⇒ aabbAB [A → aS]
⇒ aabbaSB [S → bA]
⇒ aabbabAB [A → bAA]
⇒ aabbabbAAB [A → a]
⇒ aabbabbaAB
⇒ aabbabbaa
⇒ aabbS
⇒ aabbaB
⇒ aabbabS
⇒ aabbabbA
⇒ aabbabbaa

RMD

⇒ S
⇒ aB
⇒ aaBB
⇒ aaBbS
⇒ aaBbbA
⇒ aaBbba
⇒ aabSbba
⇒ aabbAbba
⇒ aabbabba

4) S → TOOT
T → 0T | 1T | ϵ
string: 1000111

LMD
⇒ S
⇒ TOOT
⇒ 1TOOT
⇒ 10TOOT
⇒ 10ϵOOT
⇒ 1000T
⇒ 1000 1T
⇒ 10001 1T
⇒ 100011 T
⇒ 1000111 T
⇒ 1000111 ϵ
⇒ 1000111

RMD
⇒ S
⇒ TOOT
⇒ TOO 1T
⇒ TOO11T
⇒ TOϕ11T
⇒ TO0111ϵ
⇒ T00111
⇒ 1T00111
⇒ 10T00111
⇒ 10ϵ00111
⇒ 1000ϕ11

5) ϵ → +ϵϵ | *ϵϵ | -ϵϵ | x | y
string : +*-xyxy.

⇒ S
⇒ +ϵϵ
⇒ +*ϵϵϵ
⇒ +*-ϵϵϵϵ
⇒ +*-xϵϵϵ
⇒ +*-xyϵϵ
⇒ +*-xyxy

RMD
⇒ S
⇒ +ϵϵ
⇒ +ϵ+ϵϵ
⇒ +ϵ+ϵy
⇒ +ϵxϵy
⇒ +ϵy
⇒ +*ϵϵy
⇒ +*ϵxy
⇒ +*-ϵϵxy
⇒ +*-xyxy

6) $E \rightarrow I \mid E + E \mid E * E \mid (E)$

$I \rightarrow a \mid b \mid Ia \mid Ib \mid Io \mid I1$

string a*b(

a*(a+boo)

Ambigious in Grammar :-

The Grammar for which more than one parse tree is possible for the same input string stream is called as ambigious Grammar.

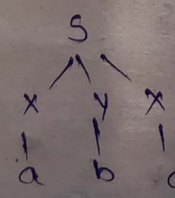
1) ex :- $S \rightarrow XYX$

$X \rightarrow a$

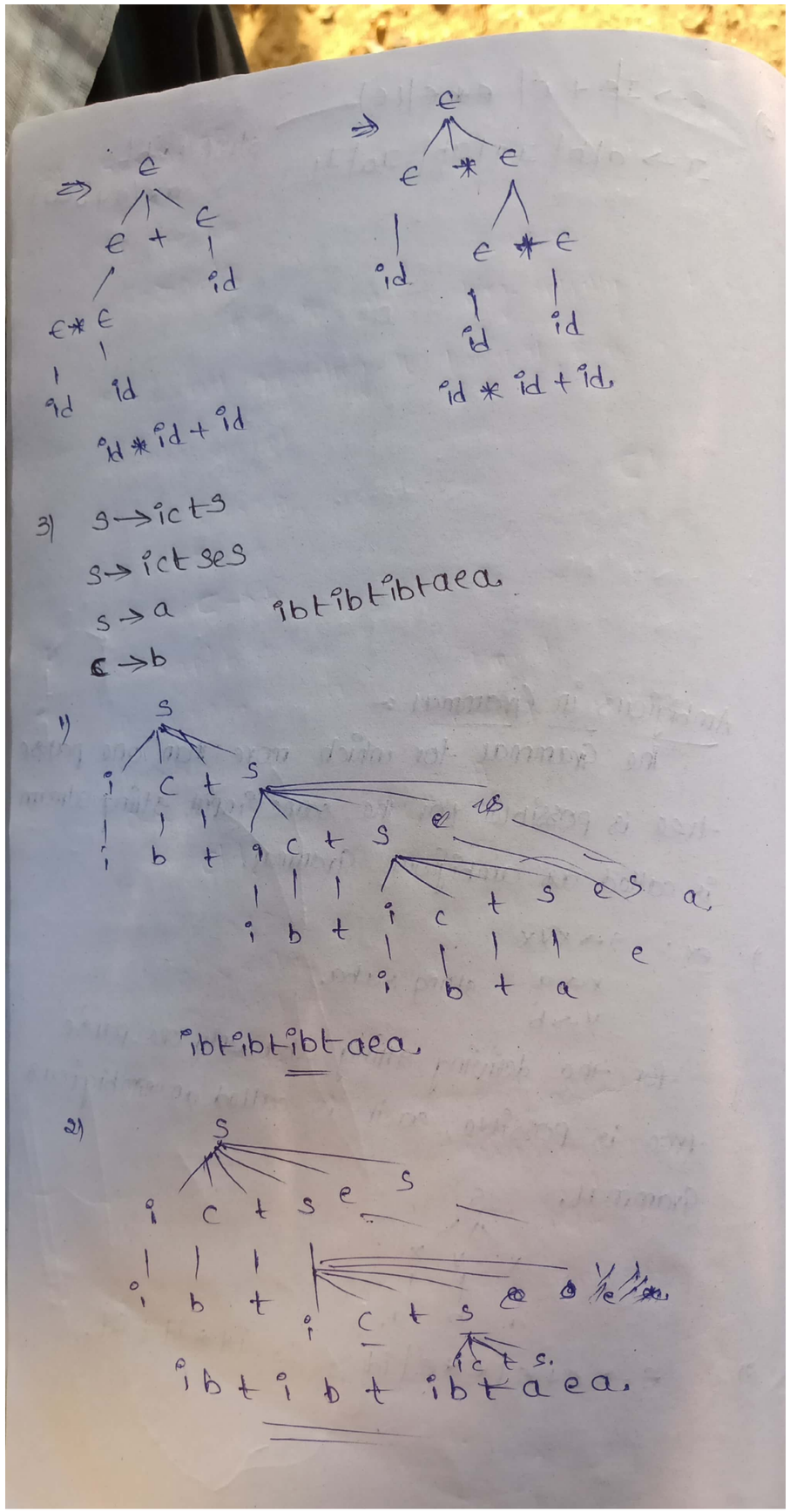
$Y \rightarrow b$

string :- aba.

for the deriving string aba only one parse tree is possible, so it is called an unambigious Grammar.



2) $E \rightarrow E+E \mid E*E \mid id$ S :- id * id + id.



Simplification of context free Grammars :-

The simplification of context free Grammars is involved in three phases.

1) Removal of useless symbols
2) Elimination of epsilon(ε) - productions.
3) Removal of unit productions.

1) Elimination of useless symbols :-

The variable which is not generating the terminal, which is not involving in the generation of a string from the start symbol.

1) $S \rightarrow 0T \mid 1T \mid X \mid 01$

$T \rightarrow 00$

Here T generates the terminal, useful symbol

X doesnot generate the any terminal, useless symbol

$S \rightarrow 0T \mid 1T \mid 01$

$T \rightarrow 00$

2) $S \rightarrow A11B \mid 11A$

$S \rightarrow 1B \mid 11$

$A \rightarrow 0$

$B \rightarrow BB$

from the above grammar 'B' doesn't generating the any terminal, so we can delete the entire Production which is generated by as well as we can delete the entire part of the production which is not generated by 'B'. $\Rightarrow$

$S \rightarrow 11A$

$S \rightarrow 11$

$A \rightarrow 0$

3) $S \to AB \mid CA$

$A \to a$

$B \to BC \mid AB$

$C \to aB \mid b$

from the above diag. Initially we found that 'B' is not generating the terminal, B is called as useless symbol

$S \to CA$

$A \to a$

$C \to b$

4) $S \to aA \mid bB$

$A \to aA \mid a$

$B \to bB$

$D \to ab \mid ea$

$E \to aE \mid d.$

Terminals	
Generated	Not generated
A	B, C
D	
E	

$S \to aA$

$A \to aA \mid a$

$D \to ab \mid ea$

$E \to d.$

In the above grammar D and E are not reachable from the start symbol, these are also called as useless symbol.

⇒ The resultant grammar after elimination of useless Symbol D and E is

$S \to aA$

$A \to aA \mid a$

found that
ted as

ated

Elimination of ϵ-Productions :-

1) Eg: $S \to 0S \mid 1S \mid \epsilon$

Sol: $S \to 0S \mid 1S \mid 0 \mid 1$

In the above Grammar we eliminate the ϵ-Productions $S \to \epsilon$.

2) $S \to aSa$

$S \to bSb$

$S \to \epsilon$

$S \to aSa \mid aa$

$S \to bSb \mid bb$

$\Rightarrow S \to aSa \mid aa \mid bSb \mid bb$

3) $A \to 0B1 \mid 1B1$

$B \to 0B \mid 1B \mid \epsilon$ $\quad B \to \epsilon.$

$A \to 0B1 \mid 1B1 \mid 01 \mid 11$

$B \to 0B \mid 1B \mid 0 \mid 1$

4) $S \to a \mid Ab \mid aBa$

$A \to b \mid \epsilon$ $\quad A \to \epsilon$

$B \to b \mid A$ $\quad B \to \epsilon$

$S \to a \mid Ab \mid aBa \mid b \mid aa$

$A \to b$

$B \to b$

Removing unit Production :-

$Var \to Var$, is called as unit production

Eg: $X \to Y$

1) eliminate the unit productions from the following CFG.

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

$A \to 0S|00$

$B \to 1|A$

$C \to 01$

$S \to C \to 01$

$B \to A \to 0S|00$

$S \to 0A|1B|01$

$A \to 0S|00$

$B \to 1|0S|00$

$C \to 01$

Optimize the CFG given below $S \to A|0C1$

$A \to B|01|10$

$C \to e|CD$

A B C D

X X

Useless

$S \to A|0C1$

$A \to 01|10$

$C \to \in$

$\in$ Production

$S \to A|0C1|01$

$A \to 01|10$.

In the above simplified Grammar, 'C' is the useless symbol, we can delete it.

$S \to A|01$

$A \to 01|10$

Unit Production

In the above simplified Grammar $S \to A$ is the unit production.

$S \to 01|10$

$A \to 01|10$

In the above simplified Grammar, 'A' is the useless symbol. The optimized Grammar after deletion of useless symbol $S \to 01|10$

Normal forms :- The context free grammar should be in either CNF (or) GNF

Chomsky Normal form (CNF):-

The production of the CFG should be in the form of

$NT \rightarrow NT \cdot NT$

$NT \rightarrow T$

Eg:- $X \rightarrow YZ$

$A \rightarrow a$

convert the following CFG into chomsky Normal form.

i) $S \rightarrow aaaaS$

$S \rightarrow aaaa$

The given Grammar is already in optimised (or) reduced CFG.

$S \rightarrow aaaaS$

Let $P_1 \rightarrow a$

$S_1 \rightarrow P_1P_1P_1P_1S$ (P_2)

$S \rightarrow P_1P_2$

$P_2 \rightarrow P_1P_1P_1S$

$P_1 \rightarrow a$

$S \rightarrow P_1P_2$

$P_2 \rightarrow P_1P_3$

$P_3 \rightarrow P_1P_1S$ (P_4)

$P_1 \rightarrow a$

$\Rightarrow S \rightarrow P_1P_2$

$P_2 \rightarrow P_1P_3$

$P_3 \rightarrow P_1P_4$

$P_4 \rightarrow P_1S$

$P_1 \rightarrow a$

$S \rightarrow aaaa$

$S \rightarrow P_1P_1P_1P_1$ (P_5 P_5)

$\Rightarrow S \rightarrow P_5P_5$

$P_5 \rightarrow P_1P_1$

$S \rightarrow P_1P_2$

$P_2 \rightarrow P_1P_3$

$P_3 \rightarrow P_1P_4$

$P_4 \rightarrow P_1S$

$S \rightarrow P_5P_5$

$P_5 \rightarrow P_1P_1$

$P_1 \rightarrow a$

2) $S \to ASA$
$A \to a$
$S \to BSB$ ✗

$S \to ASA$ ($\bar{P_1}$)
~~$P_1 \to A$~~
~~$S \to P_1 S P_1$~~

~~$S \to ASA$~~
$P_1 \to SA$
$A \to a$
$S_1 \to AP_1$

There is no Production for B
$S \to ASA$
$A \to a$

<u>GNF</u>:-
The production is in the form of $NT \to T.(NT)^*$

<u>Ex</u>:- $A \to a$ ✓
$A \to aBc$ ✓
$A \to BacD$ ✗ (Bcz first Variable must be Terminal)

1) The Grammar should be in ~~spl.~~ simplified CFG.
2) The Grammar should be in CNF
3) change the names of the non-terminal symbol into same A_i in ascending order.
4) Non-terminals are in ascending order
5) If the production is of the form $A_i \to A_j \gamma$

then i< ...
sho...
Remo...
5)
1) $S \to$ ...
$B \to$ ...
$C \to$ ...
$A \to$ ...
The ...
well ...
Acc ...
4)

then $i<J$ and

should never be $i \geq J$

b) Remove left recursion, introduce a new variable

1) $S \rightarrow CA \mid BB$

$B \rightarrow b \mid SB$

$C \rightarrow b$

$A \rightarrow a$

The given Grammar is already in reduced form as well as CNF form.

Acc to step-3 the non-terminals are named as

$S \rightarrow A_1$

$C \rightarrow A_2$

$A \rightarrow A_3$

$B \rightarrow A_4$

$A_1 \rightarrow A_2 A_3 \mid A_4 A_4$

$A_4 \rightarrow b \mid A_1 A_4$

$A_2 \rightarrow b$

$A_3 \rightarrow a,$

4) $A_1 \rightarrow A_2 A_3 \mid A_4 A_4$

$A_4 \rightarrow b \mid A_2 A_3 A_4 \mid A_4 A_4 A_4$ ($i<J$)

$A_2 \rightarrow b$

$A_3 \rightarrow a$

We can re-write the above Grammar as

$A_1 \rightarrow bA_3 \mid A_4 A_4$ ✓

$A_4 \rightarrow b \mid b A_3 A_4 \mid A_4 A_4 A_4$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

$A A$

$A_4 \to b \mid bA_3A_4 \mid A_4A_4A_4$

$Z \to A_4A_4Z \mid A_4A_4A_4$

$A_4 \to b \mid bZ \mid bA_3A_4 \mid bA_3A_4Z$

$Z \to A_4A_4Z \mid A_4A_4A_4$

$A_1 \to bA_3 \mid A_4A_4$

$A_4 \to b \mid bZ \mid bA_3A_4 \mid bA_3A_4Z$

$Z \to A_4A_4A_4 \mid A_4A_4$

$A_2 \to b$

$A_3 \to a$

(continue on next page)

Closure properties of context free language (CFL) :-

1) The CFL are closed under union.
2) The CFL are closed under concatenation
3) The CFL are closed under Kleen closure
4) The CFL are not closed under intersection and complement.

Union :- If L_1 and L_2 are two context free languages, their union $L_1 \cup L_2$ will be context free. for example

$L_1 = \{ a^n b^n c^m \mid m >= 0 \text{ and } n >= 0 \}$ and

$L_2 = \{ a^n b^m c^m \mid n >= 0 \text{ and } m >= 0 \}$

$L_3 = L_1 \cup L_2$

$= a^n b^n c^m \cup a^n b^m c^m \mid n >= 0 \text{ and } m >= 0 \}$ is also context free.

L_1 says number of a's should be equal to number to number of b's and L_2 says number of b's should be equal to number of c's. Their union says either of two conditions to be true. so it is also CFL.

Concatenation :- If L_1 and if L_2 are two context free languages, their concatenation. L_1, L_2 will also be context free. for example,

$L_1 = \{ a^n b^n \mid n >= 0 \}$ and

$L_2 = \{ c^m d^m$

$L_3 = L_1 \cdot L_2 =$

also context

L_1 says

of b's and

number of

of a's s

of c's s

create a

push to

by pu

Kleene

L_1^* wi

$L_1 =$

L_1^*

Inter

their

exam

L_2

{

a

$L_2 = \{ c^m d^m \mid m >= 0 \}$

$L_3 = L_1 \cdot L_2 = \{ a^n b^n c^m d^m \mid m >= 0 \text{ and } n >= 0 \}$ is also context free.

L_1 says number of a's should be equal to number of b's and L_2 says number of c's should be equal to number of d's. their concatenation says first number of a's should be equal to number of b's, then number of c's should be equal to number of d's. so, we can create a PDA which will first push for a's, pop for b's, push for c's then pop for d's. So it can be accepted by pushdown automata, hence context free.

Kleene closure :-

If L_1 is context free, its kleene closure L_1^* will also be context free. for example,

$L_1 = \{ a^n b^n \mid n >= 0 \}$

$L_1^* = \{ a^n b^n \mid n >= 0 \}^*$ is also context free.

Intersection and complementation :-

If L_1 and if L_2 are two context free languages, their intersection $L_1 \cap L_2$ need not be context free. for example, $L_1 = \{ a^n b^n c^m \mid n >= 0 \text{ and } m >= 0 \}$ and $L_2 = \{ a^m b^n c^n \mid n >= 0 \text{ and } m >= 0 \}$ $L_3 = L_1 \cap L_2 = \{ a^n b^n c^n \mid n >= 0 \}$ need not be context free.

L_1 says number of a's should be equal to number of b's and L_2 says number of b's should be equal to number of c's. their intersection says both conditions need to be true, but push down automata

can compare only two. so, it cannot be accepted by pushdown automata, hence not context free.
Similarly, complementation of CFL L which $\leq *$ - L, ~~and~~ need not be context free.

Applications of CFL :-

1) for defining programming languages.
2) for parsing the program by constructing syntax trees.
3) for translation of programming languages.
4) for describing arithmetic expressions.
5) for construction of compilers.

GNF - problem - continue :-

$A_1 \to bA_3 \mid bA_4 \mid bZA_4 \mid bA_3A_4A_4 \mid bA_3A_4ZA_4$

$A_4 \to b \mid bZ \mid bA_3A_4 \mid bA_3A_4Z$

$Z \to bA_4A_4 \mid bZA_4A_4 \mid bA_3A_4A_4A_4 \mid bA_3A_4ZA_4A_4$

$Z \to bA_4Z \mid bZA_4Z \mid bA_3A_4A_4Z \mid bA_3A_4ZA_4Z$

$A_2 \to b$

$A_3 \to a$

2) $S \to XB \mid AA$

$A \to a \mid SA$

$B \to b$

$X \to a$

The given Grammar is already in CNF

Acc. to steps, non-terminals are named as

$S \to A_1$

X → A2

B → A3

A → A4

A1 → A2A3 | A4A4

A4 → a | A1A4

A3 → b

A2 → a

4) A1 → A3 | A4A4 ✓

A4 → a | A2A3A4 | A4A4A4 ✗

A3 → b ~

A2 → a ✓

A4 → a | aA3A4 | A4A4A4

Z → A4A4Z | A4A4A4

A4 → a | aZ | aA3A4 | aA3A4Z

A1 → aA3 | A4A4

A4 → a | aZ | aA3A4 | aA3A4Z

Z → A4A4A4 | A4A4Z

A3 → b

A2 → a,

A1 → aA3 | aA4 | aZA4 | aA3A4A4 | aA3A4ZA4

A4 → a | aZ | aA3A4 | aA3A4Z

Z → aA4A4 | aZA4A4 | aA3A4A4A4 | aA3A4ZA4A4

Z → aA4Z | aZA4Z | aA3A4A4Z | aA3A4ZA4Z

A3 → b

A2 → a

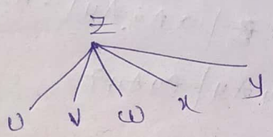
Pumping lemma for CFL :-

It is the technique which proves the given language is not CFL

If 'L' be a CFL and $z \in L$ such that $|z| \geq n$ then $uv^iwx^iy \in L \quad \forall\ i \geq 0$, where z is a string i.e., $z = uvwxy$, $|vwx| \leq n$; $|vx| \geq 1$

Process of pumping lemma :-

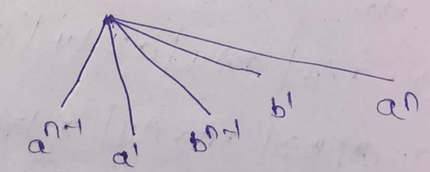
1) Assume that 'L' is a CFL
2) choose the string 'z' which belongs to 'L' i.e., $z \in L$, $|z| \geq n$
3) split the string 'z' into five parts as shown below



4) If there exists atleast one value for i such that $uv^iwx^iy \notin L$ then 'L' is not CFL

ex:- S.T the given language $L = \{a^n b^n a^n \mid n \geq 1\}$ is not CFL.

$z = a^n b^n a^n$



$z = u.v.w.x.y$

$u.v^i.w.x^i.y = a^{n-1}.a^i.b^{n-1}.b^i.a^n$

$i = 2$

$= a^{n-1}.a^2.b^{n-1}.b^2.a^n$

$= a^{n+1}.b^{n+1}.a^n \notin L$