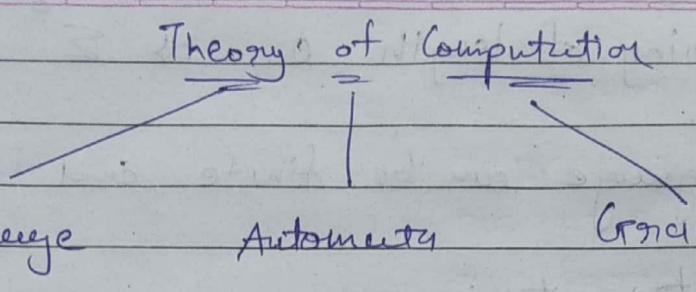
Theory of Computation



* Symbol : The building blocks of the language. (smallest unit)

{ a, b, c, 0, 1, 2, 3, ... }

* Alphabet (Σ) : Finite set of symbol.

Σ (a, b)

* String : Collection or Sequence of alphabet

Σ(a, b) : a, b, aa, bb, ab, ba are strings.

For length 2 : {aa, bb, ab, bb}
For length 3 : {aaa, aab, aba, abb, baa, bab, bba, bbb}

* Language : Collection of strings

Ex:- Σ (a, b) : Write a language : L_1 = strings of length 3, L_2 = (Infinite) All the strings start with a and end with a.

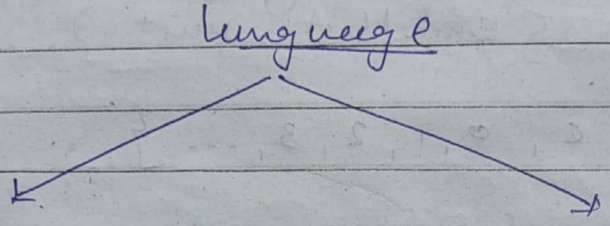
→ L_1 = { aaa, aab, aba, abb, baa, bab, bba, bbb }.

→ L_2 = { aa, aaa, aaaa, ... , aba, abba, abbba, ... }

* String of length 0. => $\Sigma^0 = \varepsilon$

* Language can be finite and infinite.

* Automata :



Finite
L_1 = length = 2
{aa, ab, ba, bb}

Infinite
L_2 = at least one a
{ a, aa, aaa, ...
ab, abb, abbb, ...
ab, abcab, ...
ba, baba, ... }

* Now, whether {abba} is a part of the language or not ?

* It is possible in finite language but very difficult in infinite language.

∴ Automata is a mathematical model or machine which is used to check whether a string is a part of any language or not.

Types of Automata
* Finite automata (FA)
Push Down Automata (PDA)
Linear Bound Automata (LBA)
Turing Machine

$\Sigma^* = \Sigma^+ + \Sigma^0$

↑ Identity element

Date
Page

* Power of Σ :- (for $\Sigma = \{a, b\}$)

Σ^0 = set of all strings with length '0'.
= λ , ε

Σ^1 = set of all strings with length '1'.
= $\{a, b\}$

Σ^2 = set of all strings with length '2'.
= $\Sigma \cdot \Sigma$
= $\{a, b\} \cdot \{a, b\}$
= $\{aa, ab, ba, bb\}$

Σ^3 = set of all strings with length '3'.
= $\Sigma \cdot \Sigma \cdot \Sigma$
= $\{aa, ab, ba, bb\} \cdot \{a, b\}$
= $\{aaa, aab, aba, abb, baa, bab, bba, bbb\}$.

Σ^4 = ;

Σ^5 = ;

⋮ ⋮

→ Σ^* = set of all strings of all length possible
↑ = $(a+b)^*$
(Kleene closure) = Infinite language.

= 2^n possible strings.

→ Σ^+ = set of all strings of all length possible except length 0

* Grammar :-

→ It is a standard way of representing a language.

→ It identifies whether a particular string is a part of language or not.

→ A grammar 'G' is defined as quadruple.

→ $G = \{ V, T, P, S \}$

V → Variable, T → Terminal, P → Production rule, S → Start symbol.

* $S \rightarrow aSb \mid \varepsilon$

S → start symbol. aSb → Terminal. ε → Null.

ε, aSb (ab), $aaSbb$ (a^2Sb^2, a^2b^2), $aaaSbbb$ (a^3Sb^3, a^3b^3), ..., a^nb^n $\boxed{n \geq 0}$

Putting $S = \varepsilon$

a^nb^n ⇐ 'a' followed by equal number of 'b'

→ Here, abab is not a part of language.

$F \subseteq Q$

(F is subset of Q.)

Date
Page

** $S \to SS$

$S \to aSb$

$S \to bSa$

$S \to \epsilon$

→ ϵ , $aSb \cdot bSa$, ab , ba , ~~abbbbbaa~~

$abba$ ($\because S = \epsilon$)

, $bSabSa$, $aSbaSb$, $aSb \cdot bSa$

$baba$ $abab$. $a \cdot aSb \cdot b \cdot b \cdot aSba$

$aabbbabab$

language $\Rightarrow$ $n_a(w) = n_b(w)$

↑ ↑

no. of 'a' no of 'b'.

* Regular Expression :-

→ Method to represent a language (L).

→ Let 'R' be a Regular Expression over Alphabet Σ if R is :

1) ε, is regular expression denoting the set $\{\varepsilon\}$.
2) ϕ, is regular expression denoting the empty state $\{\}$.
3) For each symbol $a \in \Sigma$, a is regular expression denoting set $\{a\}$.
4) Union of two RE is also regular.
5) Concatenation of two RE is also regular.
6) Kleene closure * of RE is also regular.
7) If R is regular, L(R) is also regular.
8) Nothing else, repeat 1 to 7 recursively.

→ $R = \varepsilon$, $L(R) = \{\varepsilon\}$
$R = \phi$, $L(R) = \{\}$
$R = a$, $L(R) = \{a\}$

→ $R_1 \cup R_2$ = Regular

$R_1 \cdot R_2$ = Regular

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

→ a^* = Regular, $(RE)^*$ = Regular

$a^* = a^n$, $n \geq 0$

$(a+b)^* = \{\epsilon, a, b, ab, aa, bb, ba, \ldots$

Date
Page

* Σ (a, b).

language denotation ↓ ← regular expression

1) No string $\{\}$, $\emptyset$
2) Length 0 $\{\epsilon\}$, ϵ, λ
3) Length 1 $\{a, b\}$, $(a+b)$
4) Length 2 $\{aa, ab, ba, bb\} \rightarrow (aa+ab+ba+bb)$
5) Length 3 $(a+b)(a+b)(a+b)$ ↓ $(a+b)(a+b)$.

6) Atmost 1 $\{\epsilon, a, b\} \rightarrow (\epsilon + a + b)$
7) Atmost 2 $\{\epsilon, ab, aa, ba, bb\} \rightarrow (\epsilon+a+b).(\epsilon+a+b)$

— length should be atmost 1 ⇒ 0 or 1.

* Regular expression for finite language :-

→ $a^* = \{\epsilon, a, aa, aaa, \ldots\}$
→ $a^+ = a^* - \epsilon$
$= \{a, aa, aaa, \ldots\}$
→ $a^+ = a \cdot a^*$

1) All strings having a single 'b'.
→ $a^* b a^*$

2) All strings having at least one 'b'.
→ $(a+b)^* b (a+b)^*$

3) All strings having 'bbbb' as substring.
→ $(a+b)^* bbbb (a+b)^*$

4) All strings end with 'ab'.
→ $(a+b)^* ab$

5) All strings start with 'ba'.

→ $ba \cdot (a+b)^*$

6) All strings beginning and end with 'a'.

→ $a \cdot (a+b)^* a$

7) All strings containing 'a'.

→ $(a+b)^* a (a+b)^*$

8) All strings starting and ending with different symbol.

→ $(a \cdot (a+b)^* b + b(a+b)^* a)$

9) All strings having two b's.

→ $a^* b a^* b a^*$

* IMP:-

Notation	Meaning	Language	RE
ϕ	Null set	$\{\}$	

~~[] Character set [1,2,3] [1,3,3]~~

[] Character set {1,2,3} [1,-3]

^ Reverse of set {0,1,3,4,5,6,8,9} [27]^

Practice

Date
Page

* Write Regular Expression for below languages.

→ String end with 0 : $(1+0)^*0$

→ String end with 11 : $(1+0)^*11$

→ String do not end with 01 :

⇒ which can end with 11, 10, 00

$\epsilon + 1 + 0 + (1+0)^*(11+10+00)$

↓

ϵ, 1, 0,
11, 10, 00,
111, 110, 100
011, 010, 000
⇒ starting with any combi. of 1, 0.

→ Next to last (Second last) symbol is 0.

∴ last can be strickly 0 or 1 not combination of 0 or 1.

01
101
100
00

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

↓ ↓

All Combination of 0 or 1 OR 1

→ String which has first and last letters different.

any combination of 0 or 1 in middle

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

↑ ↑ ↑ ↑

first letter | last letter | first letter | last letter

→ String begin or end with 00 or 11.

⇒ String can begin with 00 or 11 and end with 10 or 01.

⇒ String can end with 00 or 11 and begin with 10 or 01.

$$(00+11)(0+1)^*(10+01) + (10+01)(0+1)^*(00+11).$$

→ String containing 00 as substring.

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

→ String not containing 00 as substring

$$\epsilon, 1, 0, 01, 10, 11, 010, 101, 011, 110, \ldots$$

$$(0+\epsilon)(1+10)^* \quad | \epsilon 1 \Rightarrow 1$$

→ String end with 1 and not containing 00.

⇒ here 'and' is there ∴ no ϵ (null char)

$$1, 01, 101, 10101, 11, 011, 0101, \ldots$$

$$(1+01)^+$$

Any combination of 0 or 01 without null character (ϵ).

→ String containing both 11 and 010 as sub string.

$(0+1)^*(11)(0+1)^*(010)(0+1)^*$

In this case, 11 will appear always ahead of 010. What if 010 occurs before 11? Then, this soln fails.

∴ We need to modify above soln.

$(0+1)^*\left((11)(0+1)^*(010) + (010)(0+1)^*(11)\right)(0+1)^*$

→ String that contains 10 and 01 as substring.

$(0+1)^*\left((10)(0+1)^*(01) + (01)(0+1)^*(10)\right)(0+1)^*$

OR

$(0+1)^*\left(1^+0^+1^+ + 0^+1^+0^+\right)(0+1)^*$

1^+ ⇒ one or more occurences of 1

0^+ ⇒ one or more occurences of 0.

→ String not containing 0011 as substring.

$(0+10)^*$

→ String has exactly two 0's :

$(1)^*\,0\,(1)^*\,0\,(1)^*$

→ String has atleast two 0's : $(1+0)^*\,0\,(1+0)^*\,0\,(1+0)^*$

→ String has max one 00.

$$(01+1)^*(\epsilon+00+0)(10+1)^*$$

→ even number of 0's.

$$(1)^*(0(1)^*0(1)^*)^*$$

→ odd number of 1's

1, 10, 01, 001, 010, 100, 111, ...

$$(0)^*(1(0)^*1(0)^*1)^*(0)^*$$

In this case 1, 10, 01 will fail to satisfy the string's regular expression.

Therefore write 1 seperately.

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

Important

→ Even number of 0's and 1's

11, 00, 1100, 0011, 1010, 0101, 1111, 0000, ...

$$(00+11)^* + (00+11)^*(10+01)(00+11)^*(10+01)(00+11)^*$$

→ Strings in which every 0 is followed immediately by 11.

1011, 011011, 011, 11011, 011011, 01111, ...

$$(1)^*(011)^*$$

→ String of length 3 or, 1 plus a multiple of 3.

⇒ String can be of length 3 or multiple of 3 plus 1 length.

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

→ String of even length.

ϵ, 11, 00, 10, 01, 1100, 0011, 1010, 0101, 0010, 1110, 1111, 0000, ...

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

→ String of length 6 or less.

$$(\epsilon + 0 + 1)^6$$

→ $\wedge \in L$, if $x \in L$ then $001x$ and $x11$ are in L.

$$(001)^*(11)^*$$

→ $0 \in L$, if $x \in L$ then $001x$ and $x001$ and $x11$ are in L.

$$(001)^* 0 (001+11)^*$$

↑ to accept odd number of 0's

→ $\wedge \in L$, $0 \in L$, if $x \in L$ then $001x$ and $11x$ are in L.

→ language of C identifier

$$[a\text{-}zA\text{-}Z\_][a\text{-}zA\text{-}Z0\text{-}9\_]^*$$

→ Email.

$$[a\text{-}zA\text{-}Z0\text{-}9\_\text{-}\backslash\cdot]^+ [@][a\text{-}z]^+ [\backslash\cdot][a\text{-}z]_{\{2,3\}}$$

→ Unsigned real number

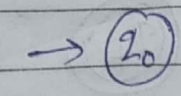
$$[0\text{-}9]^+ (.[0\text{-}9]^+)^* ([+\text{-}]e[0\text{-}9]^+)^*$$

→ Signed real number.

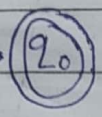
$$[+\text{-}][0\text{-}9]^+ (.[0\text{-}9]^+)^* ([+\text{-}]e[0\text{-}9]^+)^*$$

* **Basics for Conversion :** REGEX → DFA

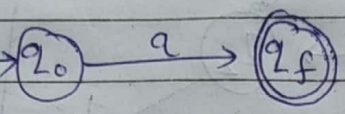
→ ϕ :



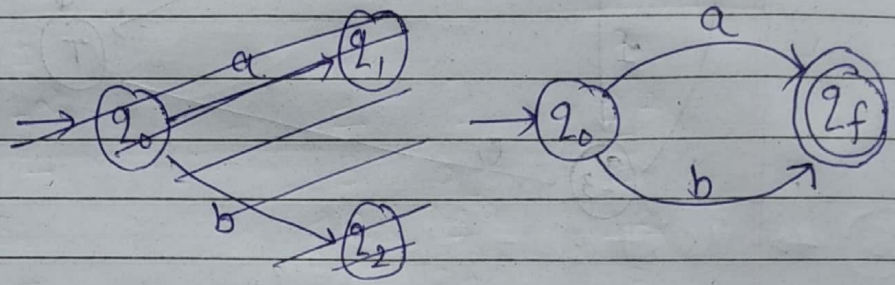
→ ϵ :



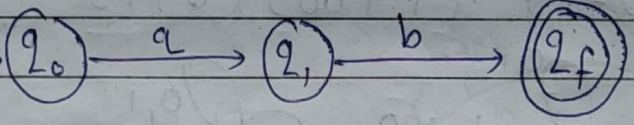
→ a :



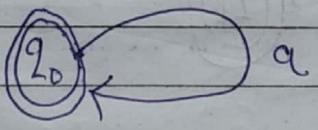
→ $a+b$:



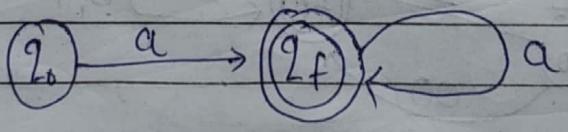
→ $a \cdot b$:



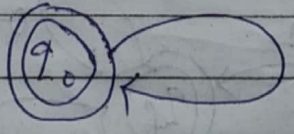
→ a^* :



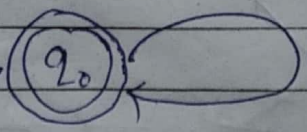
→ a^+ :



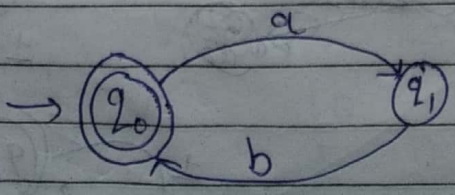
→ $(a+b)^*$:



→ $(a \cdot b)^*$:

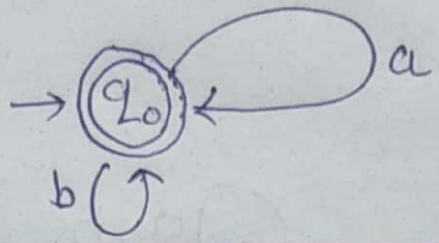


step 1

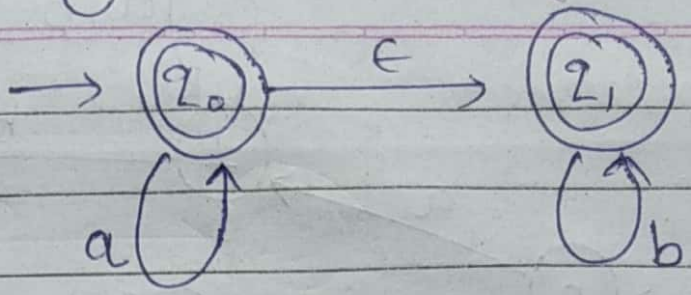


step 2
(Ans)

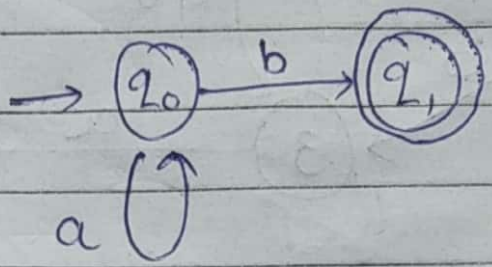
→ $(a^* + b^*)^*$:



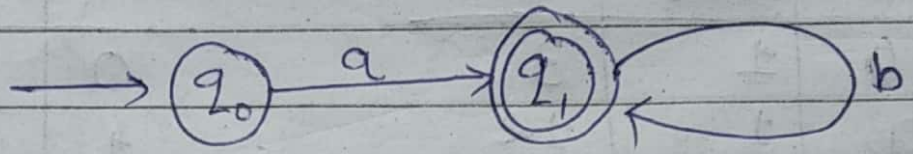
→ $a^* b^*$:



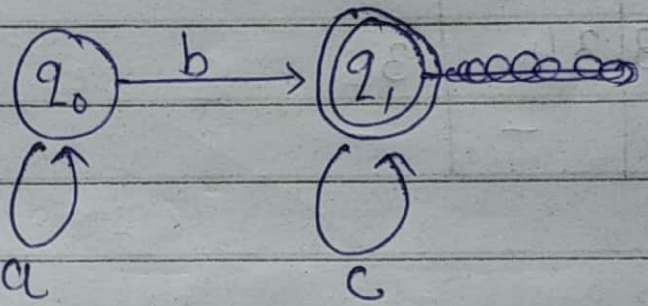
→ $a^* b$:



→ ab^* :

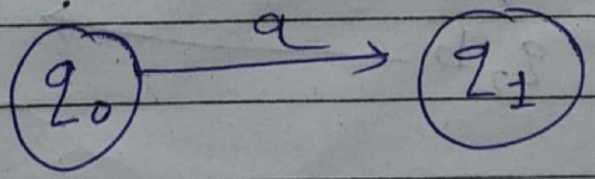


→ $a^* b c^*$:



* Deterministic Finite Automata :-
(DFA)

→ Deterministic : If alphabet 'a' is applied to any node 'q_0' then it will definitely gives another particular state called 'q_1'.

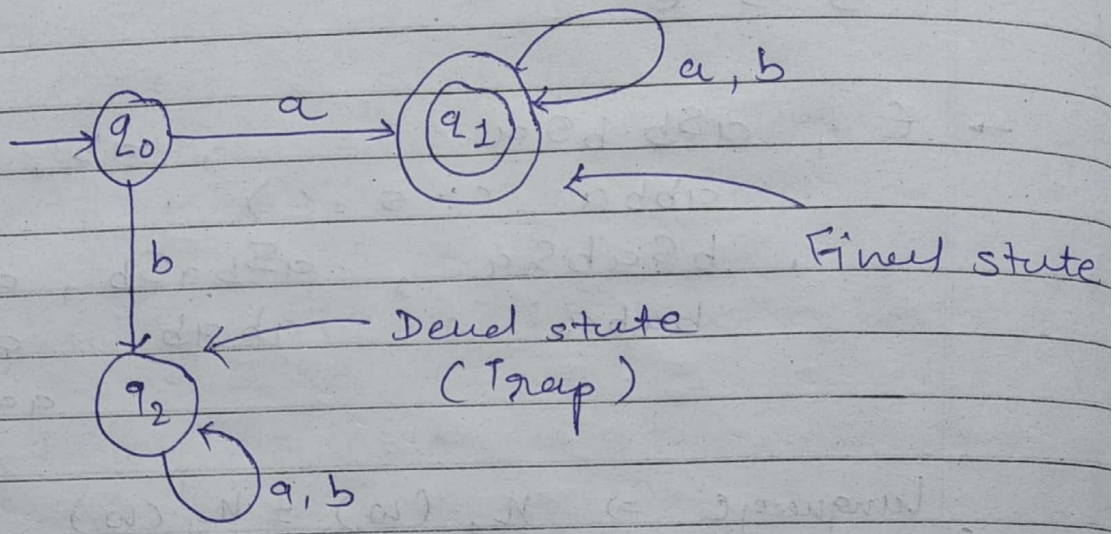


→ DFA (Q, Σ, δ, q_0, F)

- Q : Set of all finite states
- Σ : set of alphabet ex. {a, b}
- δ : Transition (Delta) from one node to another node.
- q_0 : start
- F : set of finite final state

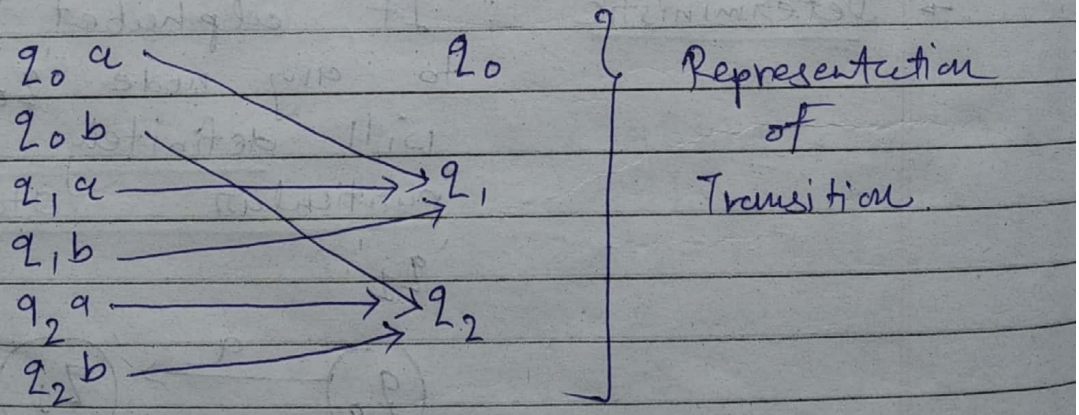
** Write a language starting with 'a'. and draw a DFA.

→ { a, aa, aaa, ab, ... }



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

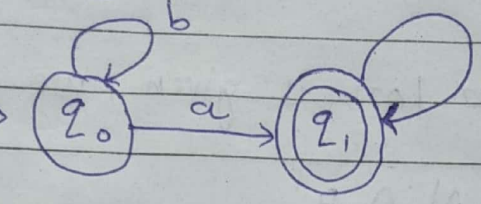
q_0 (a, b)
q_1
q_2



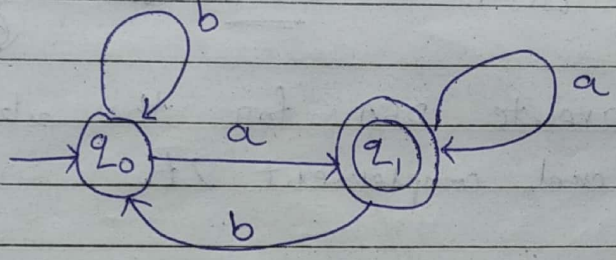
** Construct a DFA which accepts a language of all strings. $\Sigma = \{a, b\}$

→ Containing 'a'.
→ End with 'a'.

Ans:→ Containing 'a' : { a, aa, aaa, ba, ab, abab, baa, aab, ... }



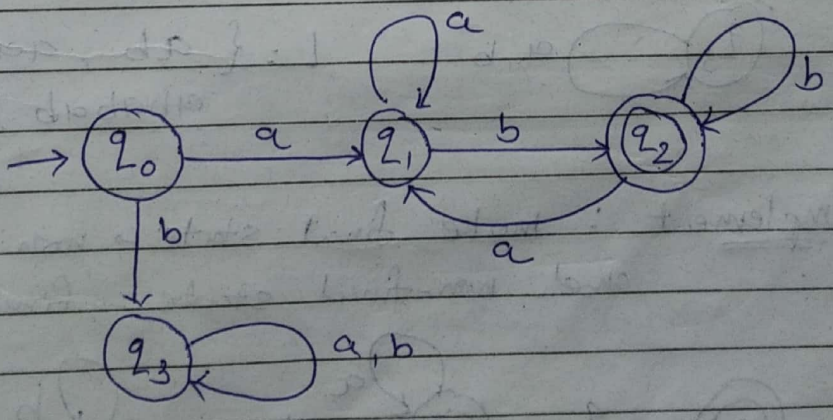
→ Ending with 'a' : { a, aa, aaa, aaaa, ...,
ba, bba, bbba, ...,
ababa, aba, ...
baba, bababa, ... }



DFA

* Construct a language which accepts all strings starting with 'a' and ending with 'b'. $\Sigma = \{a, b\}$

Ans:→ L : { ab, aabb, aaabbb, aaaabbbb, abab, ababab, aaaaab, aaaaaaab, ... }



* Construct a DFA which accepts a language of all strings not starting with 'a' or not ending with 'b'.

$L = \{ \varepsilon, a, b, \dots \}$

→ When not or non / or is given, use D'morgan's law.

$$(A \cup B)' = A' \cap B'$$

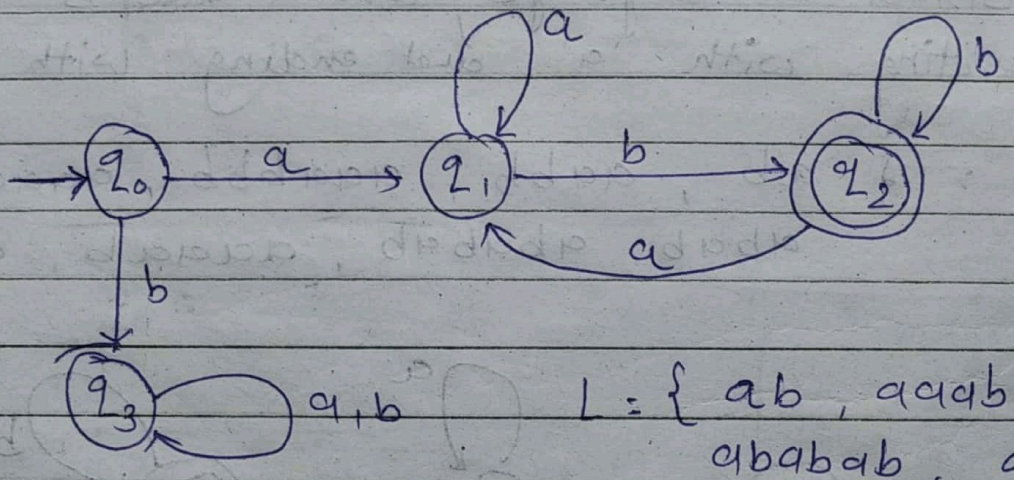
Not starting with 'a' or not ending with 'b'

↓

= starting with 'a' and not ending with 'b'.

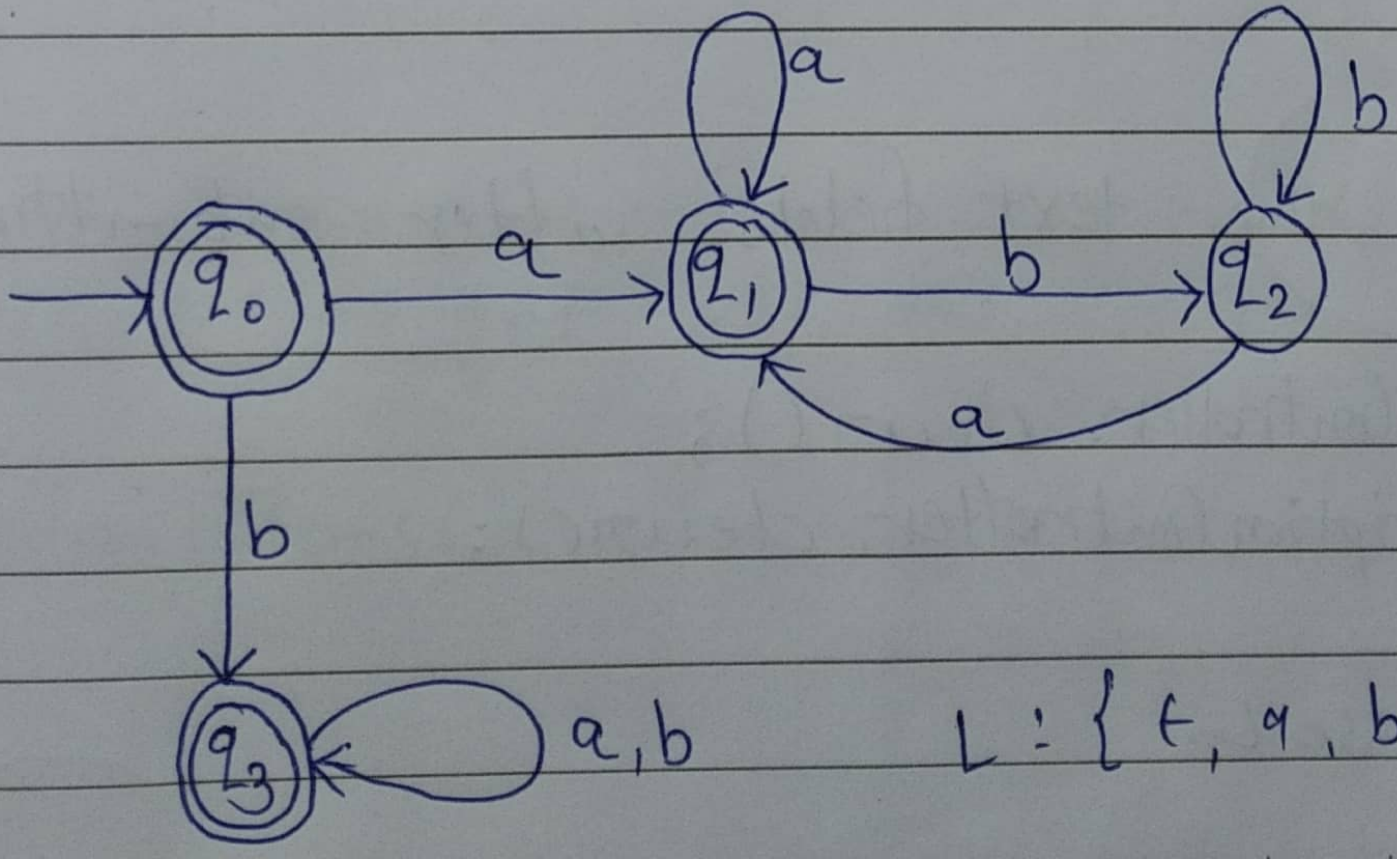
→ Therefore, create DFA for the above derived statement and complement it.

→ Starting with 'a' and ending with 'b'.



$L = \{ ab, aaab, abab, ababab, abbb, \dots \}$

→ Complement : make final state → non final and non-final state → final.



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

* Construct a DFA of all strings in which 2nd symbol is 0 and fourth symbol is 1. (0,1).

0/1 0 0/1 1

→ min length = 4

∴ min state in DFA = n+1

∴ 4+1 = 5 states

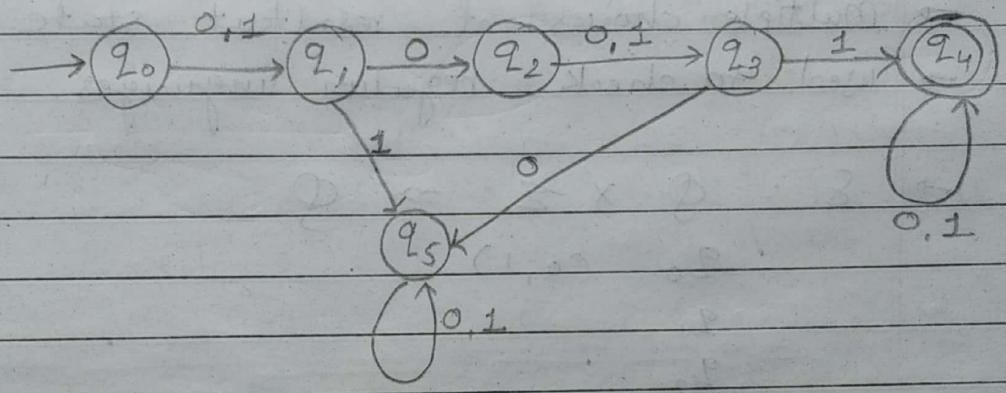
+1 trap state

∴ 6 states

→ 1 [0] 1 [1]

1st 2nd 3rd 4th

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



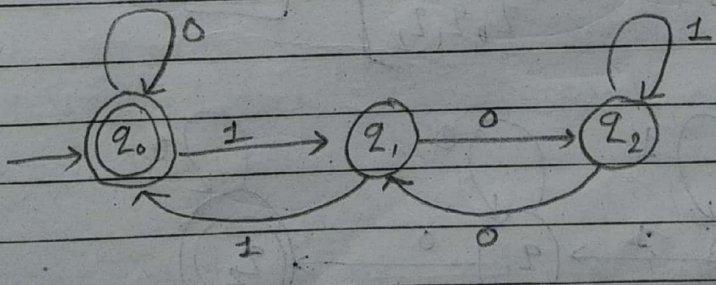
* Construct a DFA which accepts a language of all binary strings divisible by three over Σ(0,1).

* ending state — remainder

q_0 — 0

q_1 — 1

q_2 — 2



ending

000 ✓ q_0

001 × q_1

010 × q_2

011 ✓ q_0

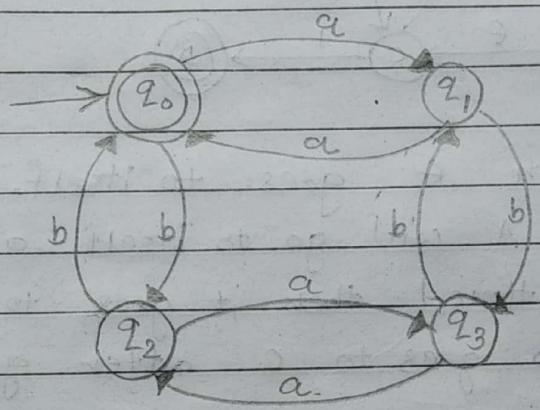
111 × q_1

ϵ exists :. starting state is ending state.

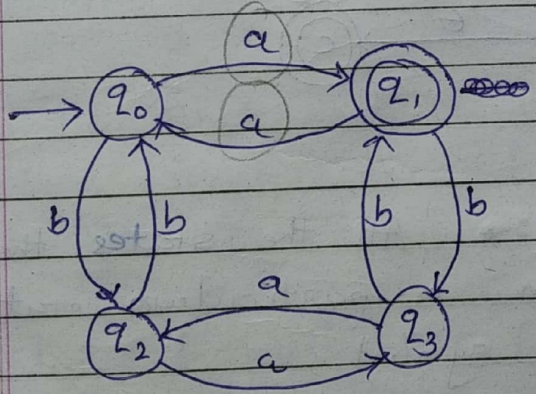
Date
Page

* Design DFA where $L = \{ w \mid w$ has even number of a's and even number of b's $\}$, $\Sigma(a, b)$.

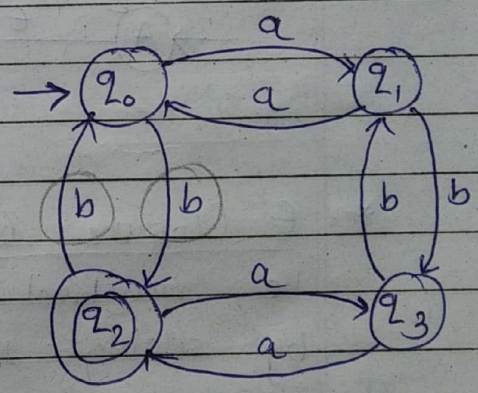
→ $L = \{ \epsilon, aa, bb, aabb, abab, bbaa, baba, abba, baab, \ldots \}$



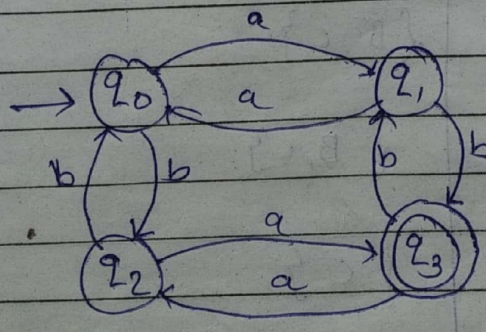
* odd a's and even b's.



* even a's and odd b's.



* odd a's and odd b's.

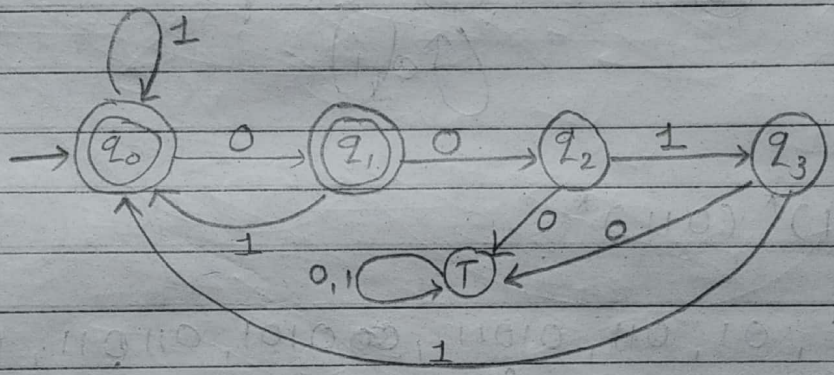


* Construct DFA :-

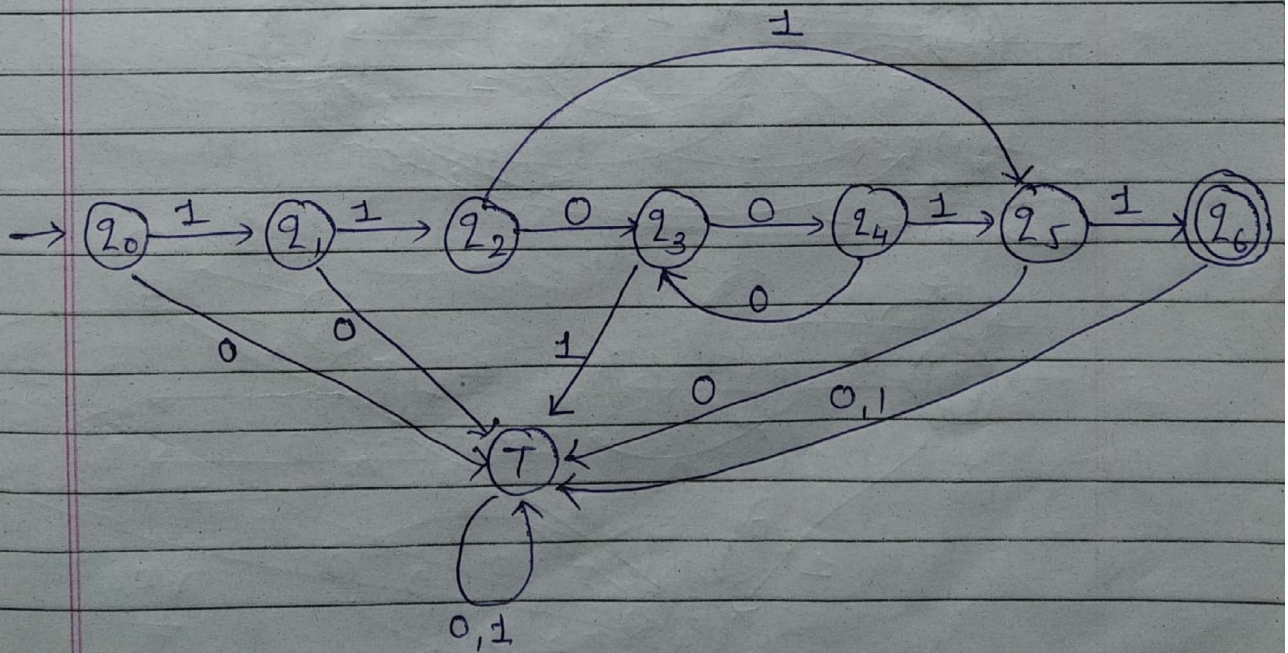
(1) ** Language accepting a string over {0,1} where every occurance of 00 is followed by 11.

{ ∈, 0, 1, 01, 10, 11, 0011, 10011, 110011, 0011 01 0011, 0011 11 0011, 0011 01 0011, ... }

R.E : (0011)* (01+1)* (0011)* (10+1)* (0011)* (0+ε)

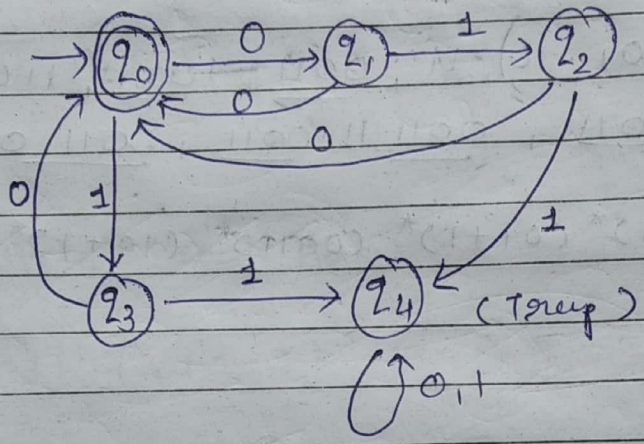


(2) ** 11(00)* 11 : { 11 0011, 11 0000 11, 1111, 11 0000 0011, ... }



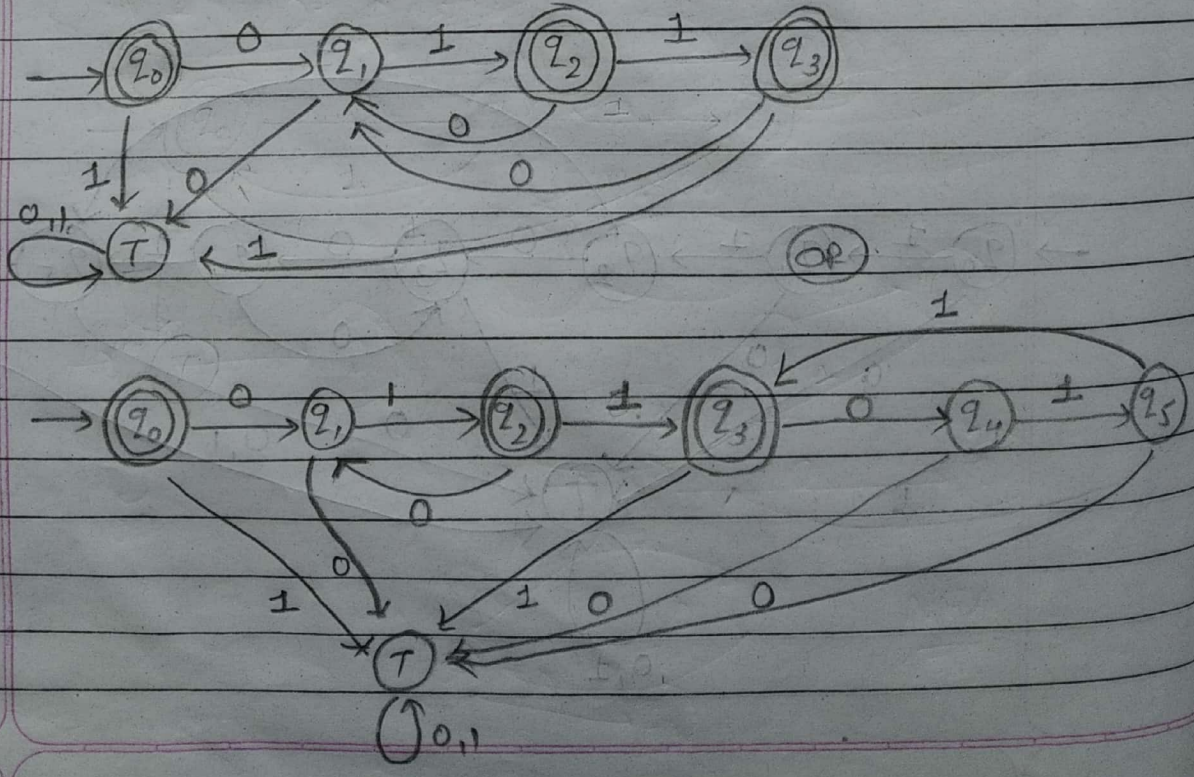
(3) ** $(010|00)^*(10)^* \Rightarrow (010+00)^*(10)^*$

{ ∈, 00, 010, 10, 01000, 01010, 0010, ... }



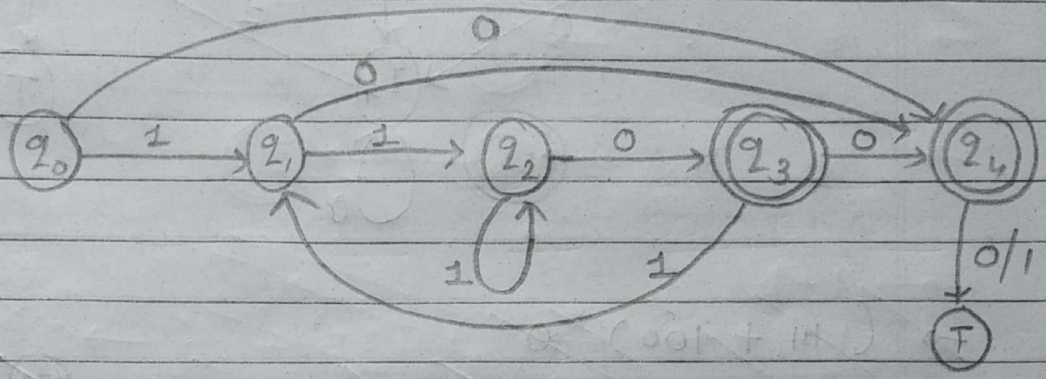
(4) ** $(01)^*(011)^*$

{ ∈, 01, 011, 01011, 0101, 011011, 01011011, 0101011, ... }



(5) ** $(1|110)^* 0$

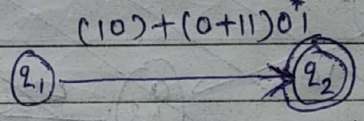
{ 0, 10, 110, 1110, 1110, 1100, 1101100, 110110100 }

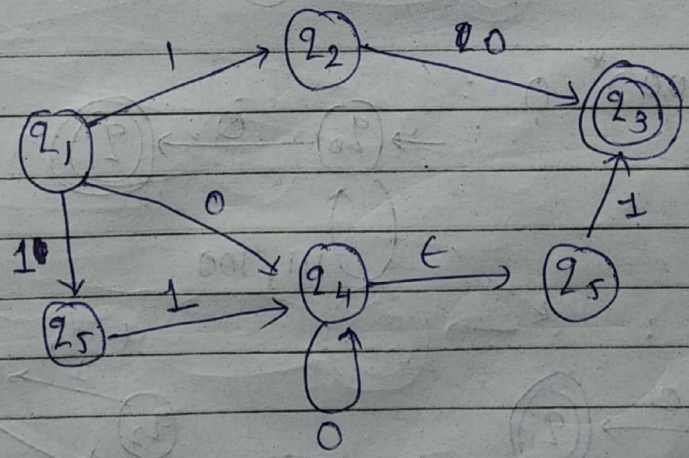


(6) ** $(10)+(0+11)0^*1$

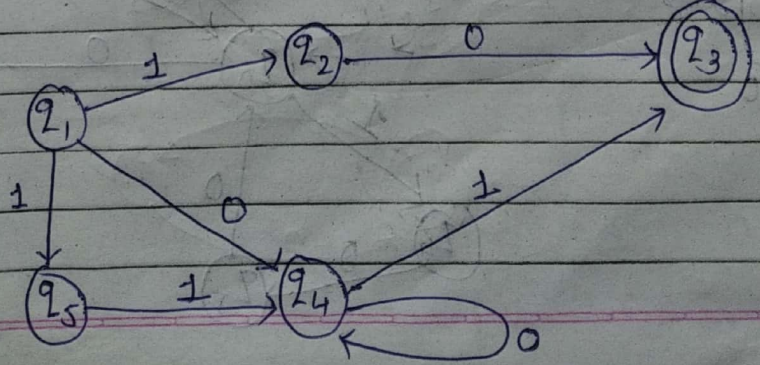
{ 101, 0001, 1101, 00001, 1100001, 1100001, 01, 11001, }

Stepwise :





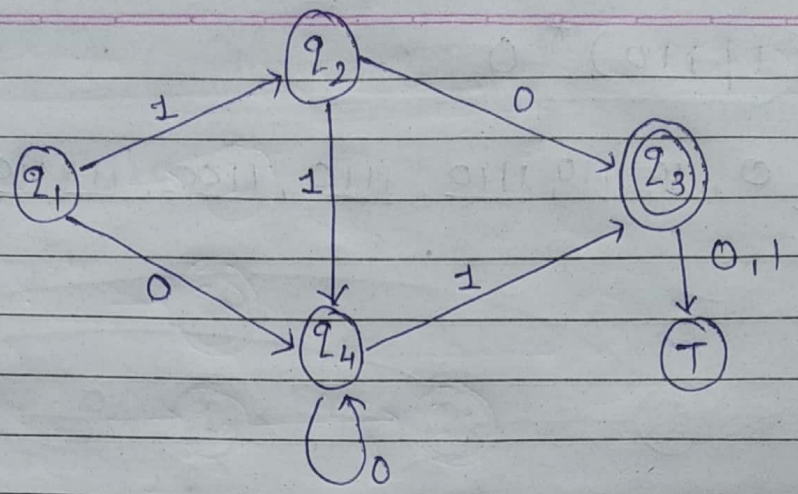
This is NFA →



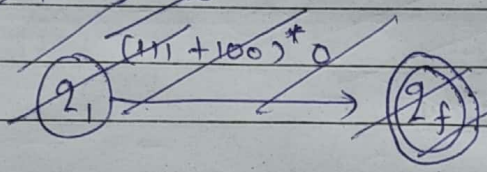
if * ⇒ Self loop. | Complex logic → Convert to NFA then Convert it to DFA.

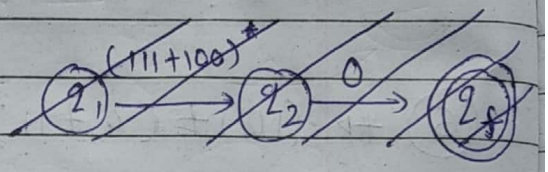
Date
Page

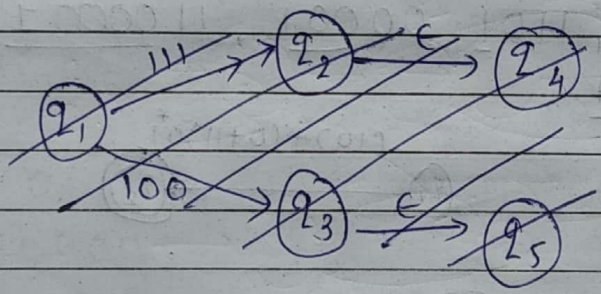
DFA →



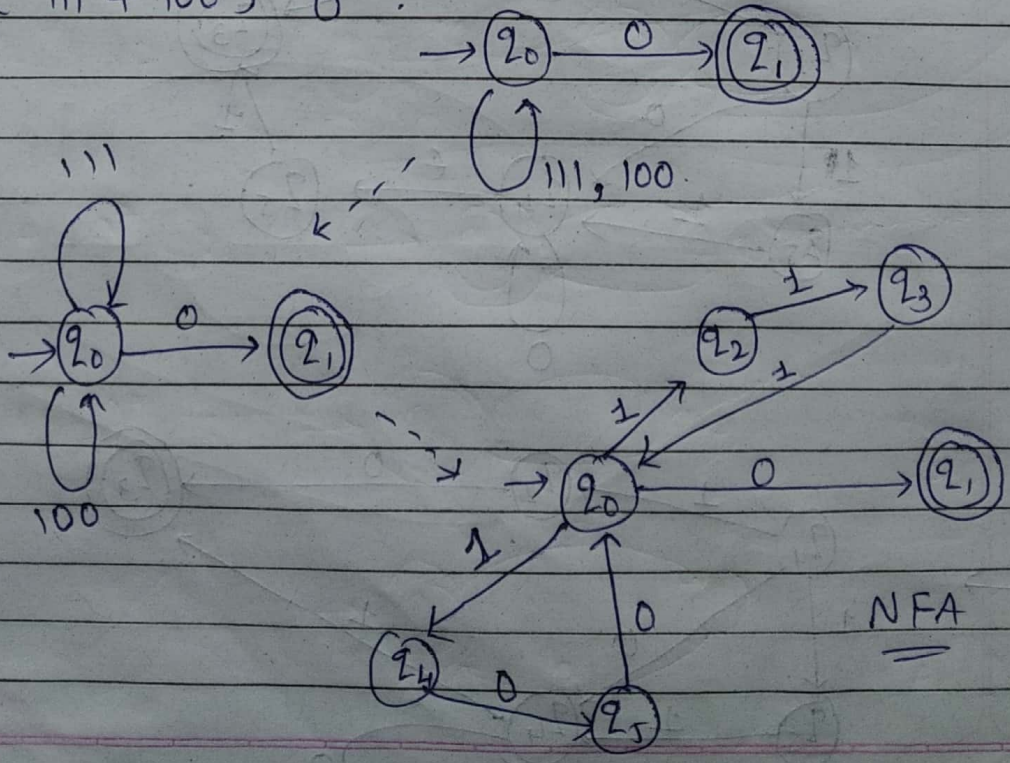
→ $(111 + 100)^* 0$

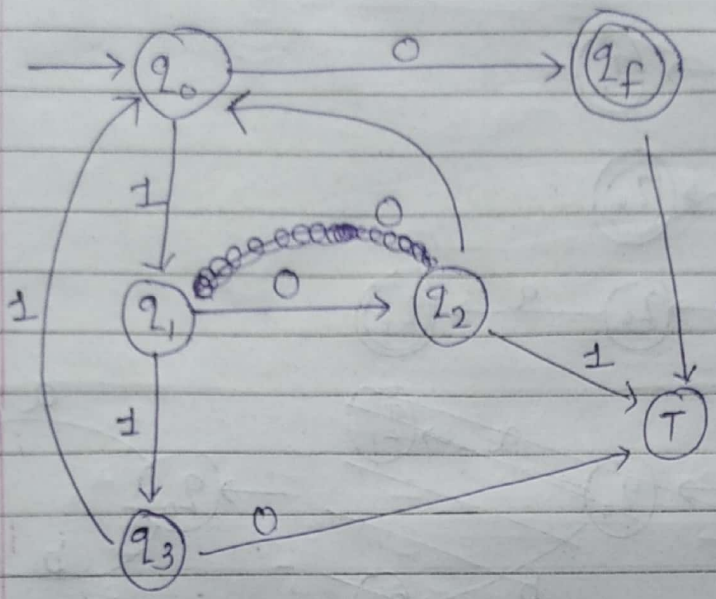






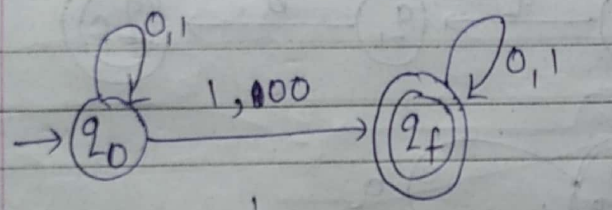
(7) ** $(111 + 100)^* 0$:



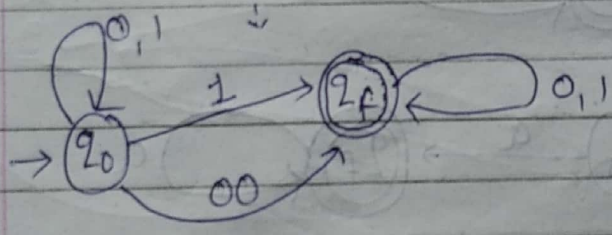


DFA

(8) ** $(0+1)^*(1+00)(0+1)^*$

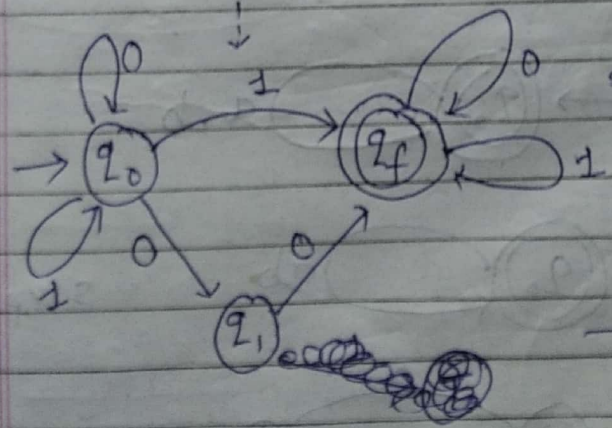


	0	1
→ q_0	q_0q_1	q_0q_f
q_1	q_f	–
q_f	q_f	q_f

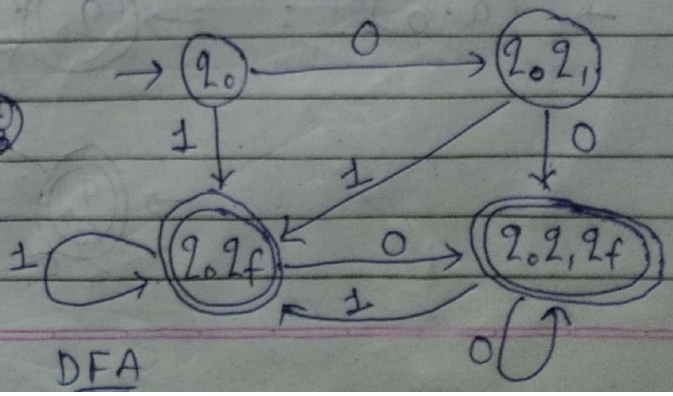


	0	1
→ q_0	q_0q_1	q_0q_f
q_0q_1	$q_0q_1q_f$	q_0q_f
final states q_0q_f	$q_0q_1q_f$	q_0q_f q_f
final states $q_0q_1q_f$	$q_0q_1q_f$	q_0q_f

as it includes q_f.



NFA



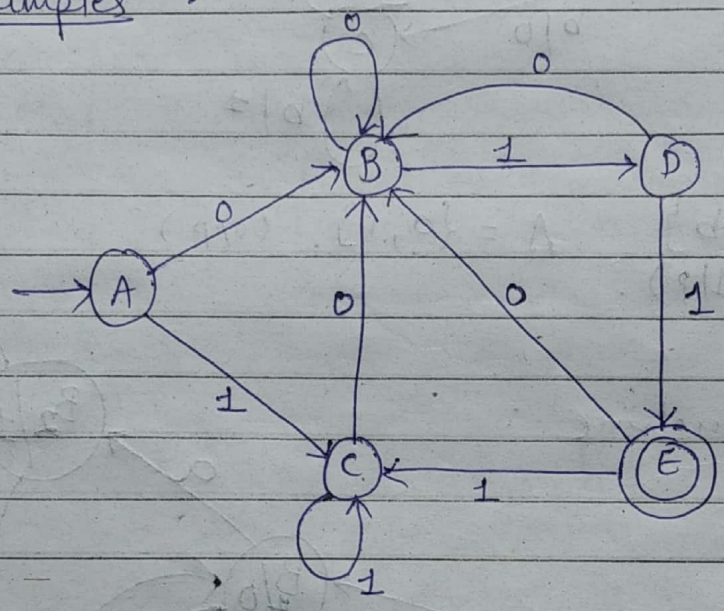
DFA

* Minimization of DFA :-

→ It is required to obtain the minimal version of DFA which consists of the minimum number of states possible.

* Examples :-

(1)



i) Draw state transition table for DFA.

state	0	1
→ A	B	C
B	B	D
C	B	C
D	B	E
(E)	B	C

← E does not belongs to non-final set. ∴ CD is not 1-equivalent.

ii) 0- Equivalence : { A, B, C, D } { E }
(Non final states) (Final states)

AB ✓
AC ✓
CD ✗

* 1- Equivalence : { A, B, C } { D } { E }

~~iii)~~ 2 Equivalence : {A, C} {B} {D} {E}

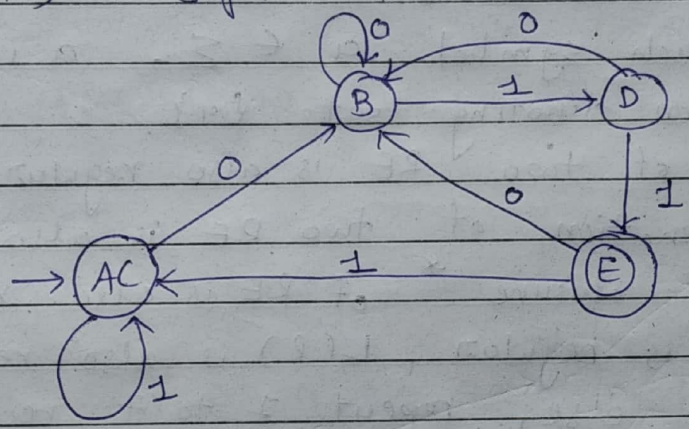
AB ✗
AC ✓
~~AB ✗~~

3 Equivalence : {A, C} {B} {D} {E}

AC ✓

{ If both are (0, 1) same then no need to check for the output to be from the same set. }

iii) Design DFA :-



** 1- equivalence :-

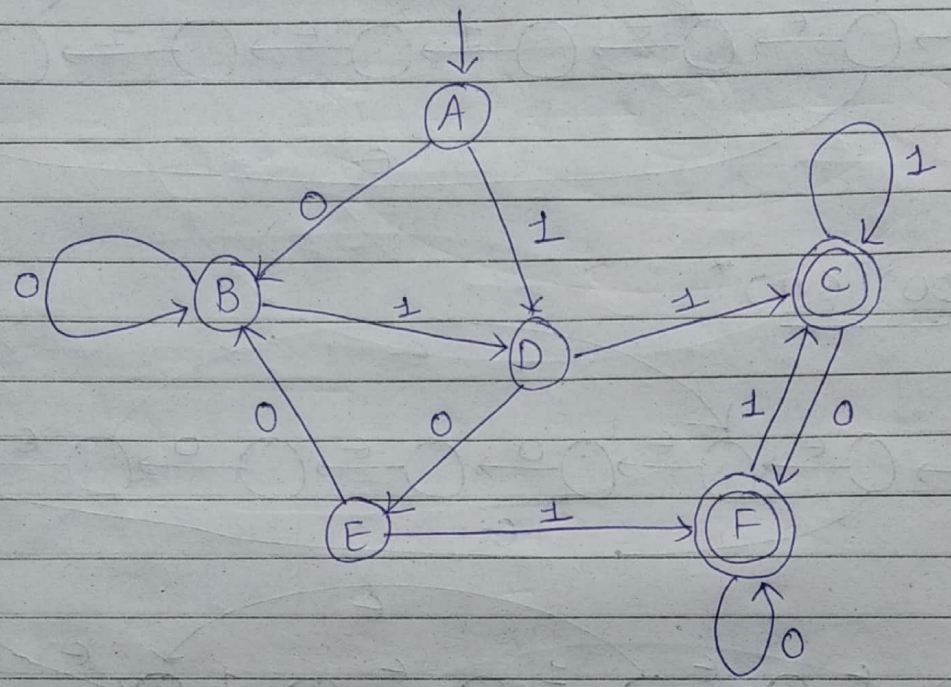
Take the two states from the first bracket and check if their results fall in the same set ⇒ It's 1-equivalence.

↓

It is not mandatory for them to fall in the set they belong from.

* Minimize DFA :-

(1) **



** Transition table :-

State	0	1
A	B	D
B	B	D
(C)	F	C
D	E	C
E	B	F
(F)	F	C

0- equivalence : π_0 { A, B, D, E } { C, F }
(Non final) (final)

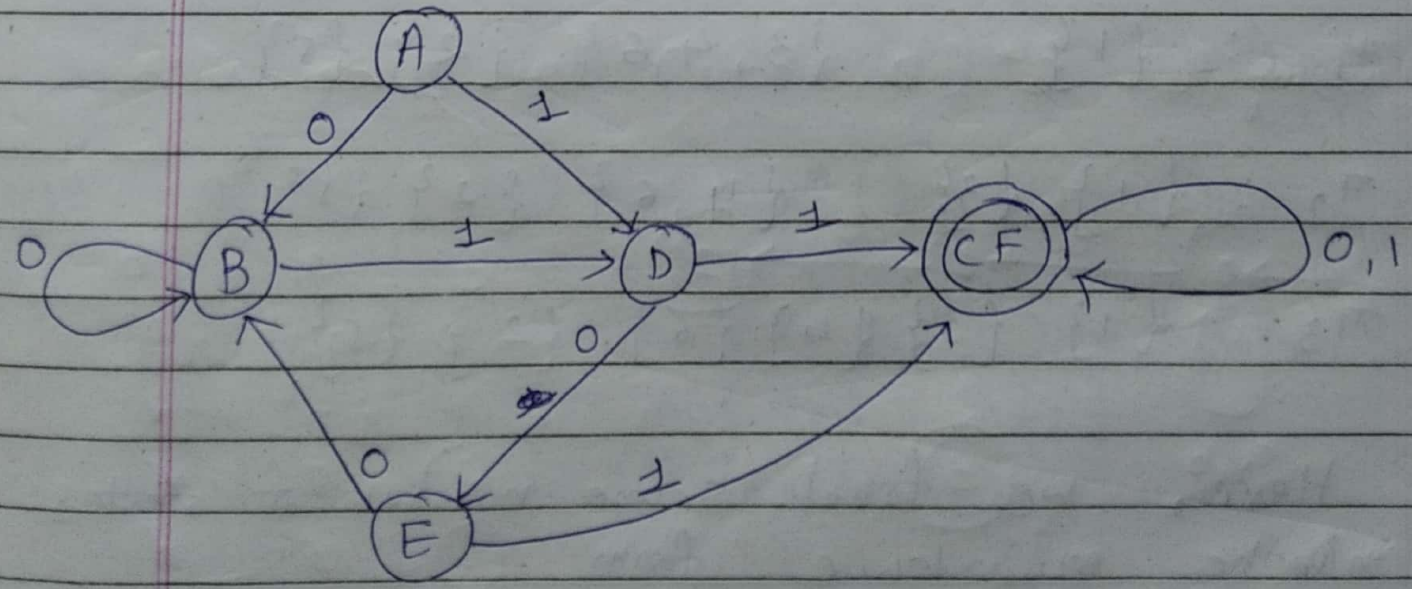
AB ✓
AD ✗
AE ✗
DE ✓
CF ✓

1 - equi. : π_1 { A, B } { D, E } { C, F }

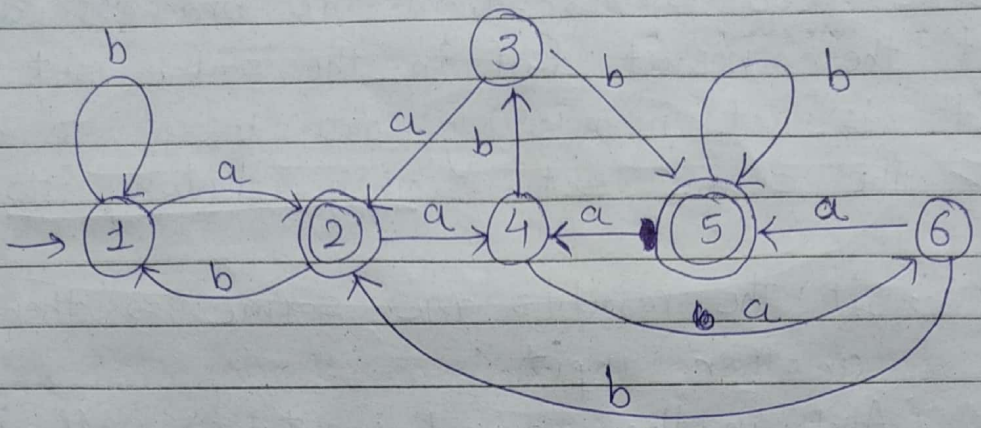
AB ✕ CF ≡ → π_2 : {A, B} {D} {E} {C, F}

CF ≡ → π_3 : {A} {B} {D} {E} {C, F}

→ π_4 : {A} {B} {D} {E} {C, F}



(2) **



→ Transition table :-

State	a	b
1	2	1
(2)	4	1
3	2	5
4	6	3
(5)	4	5
6	5	2

1,3 X
1,4 X
1,6 X
2,5 X
3,4 X
3,6 X
4,6 X

π_0 : {1, 3, 4, 6} {2, 5}

π_1 : {1} {3, 4, 6} {2} {5}

π_2 : {1} {3} {4, 6} {2} {5}

π_3 : {1} {3} {4} {6} {2} {5}

Here, we found same number of states in the equivalence form.

∴ Give DFA is already minimized.

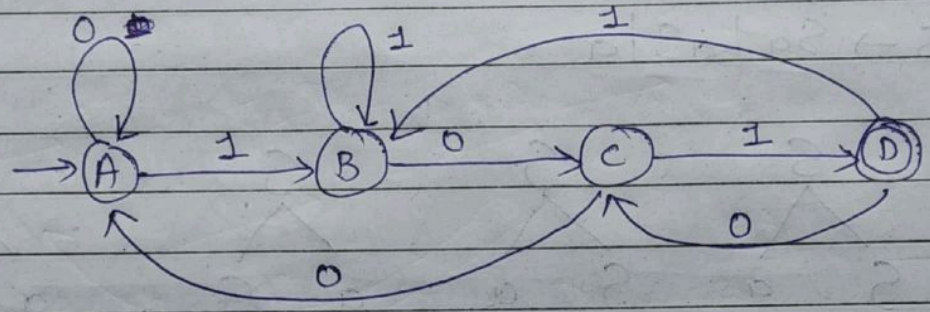
IMP

* Construct DFA for belogw given language L1 and L2. Construct the DFA for L1 ∪ L2 , L1 ∩ L2 , L2 - L1.

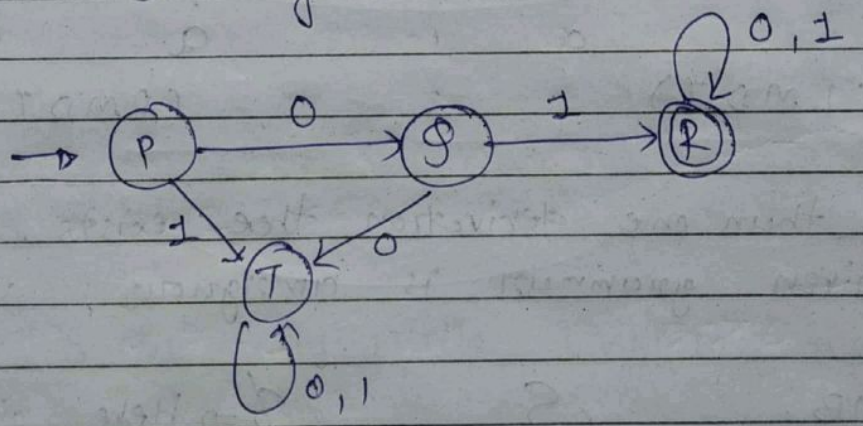
L1 : string over {0,1} , end with 101.
L2 : string over {0,1}, start with 01.

→ L1 : end with 101.

00101 , 101 , 11101 , 10101 , 11101 , ...



→ L2 : starting with 01.



→ States of L1 and L2

A	P
B	Q
C	R
D	T

State	0	1
AP	AQ	BT
AQ	AT	BR
AR	AR	BR
AT	AT	BT
BP	CQ	BT
BQ	CT	BR
BR	CR	BR
BT	CT	BT
CP	AQ	DT
CQ	AT	DR
CR	AR	DR
CT	AT	DT
DP	CQ	BT
DQ	CT	BR
DR	CR	BR
DT	CT	BT

→ $L1 \cup L2$ => final states are those which include final state D OR R.

∴ final states in $L1 \cup L2$ will be : AR, BR, CR, DP, DQ, DR, DT.

→ $L1 \cap L2$ => final states are those which include final state D AND R.

∴ final states for $L1 \cap L2$ will be only : DR.

→ $L2 - L1$

↑	↑
final state	Non-final state
R	A, B, C

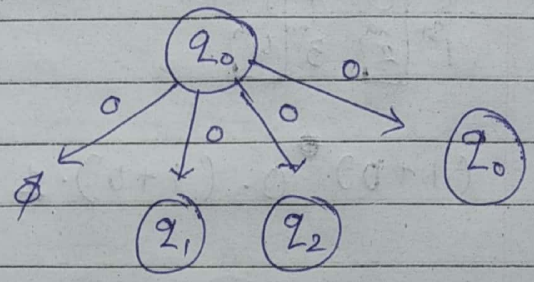
=> RA, RB, RC will be final states of $L2 - L1$

→ Then create DFA according to the above transition table.

* Non Deterministic Finite Automata :-
(NDFA or NFA)

→ NDFA (Q , Σ , q_0 , F , δ)

→ Non deterministic :



→ Multiple choices of resultant state exists.

→ Used to check regular languages.

→ δ : Q × Σ ⇒ Q

q_0 (0,1)

q_1

q_2

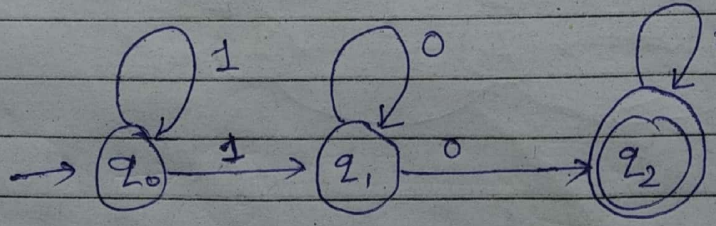
From the diagram given below:

q_0 0	q_0
q_0 1	q_1
q_1 0	q_2
q_1 1	q_0q_1
q_2 0	q_0q_2
q_2 1	q_1q_2
	$q_0q_1q_2$
	ϕ

2^n transitions will be there.

∴ in this case,
$2^3 = 8$ transitions exist.

2^Q



* NFA of all binary strings in which 2nd last bit is 1. $\Sigma\{0,1\}$.

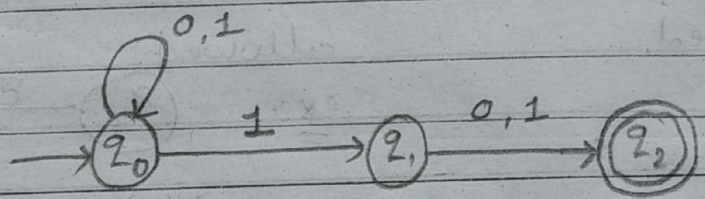
→ any ... 1 0/1

... (0+1)* 1 (0+1)

Any combination of 0 or 1.

Total no. of states
= min string length + 1
= 2+1
= 3

→ 10, 11, 010, 110, 111, 1110, 0110, ...

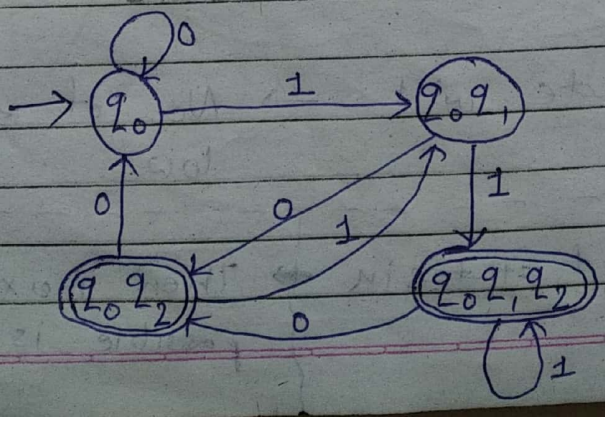


** Convert NFA to DFA :-
(for above NFA)

	0	1
→ q_0	q_0	q_0q_1
q_1	q_2	q_2
(q_2)	-	-

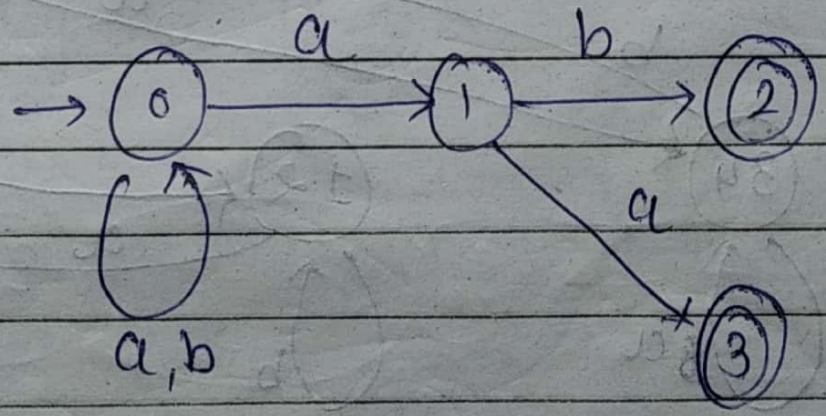
	0	1
→ q_0	q_0	q_0q_1
q_0q_1	q_0q_2	$q_0q_1q_2$
(q_0q_2)	q_0	q_0q_1
($q_0q_1q_2$)	q_0q_2	$q_0q_1q_2$

final states as it consists of q_2.



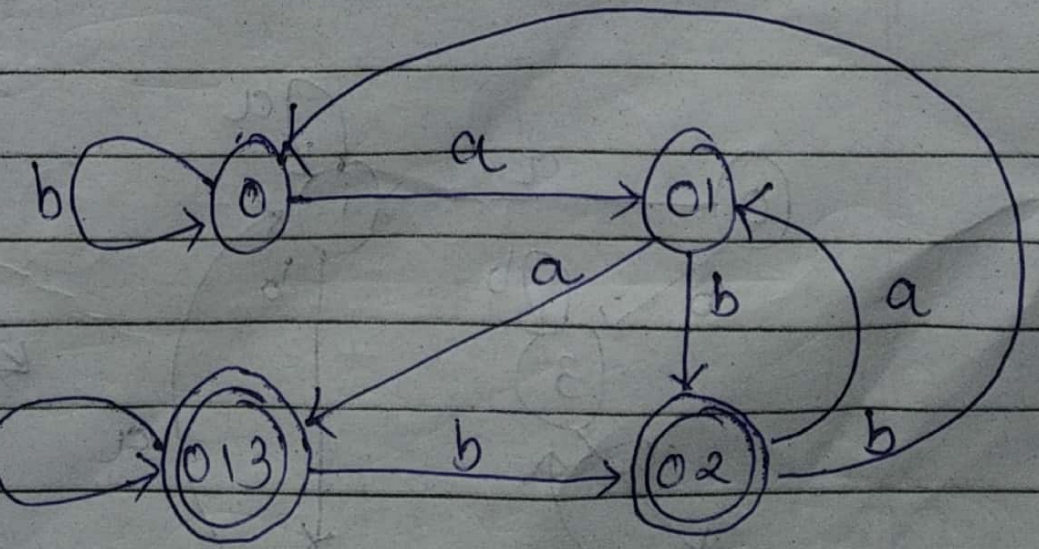
* Convert NFA to DFA :-

(1) **



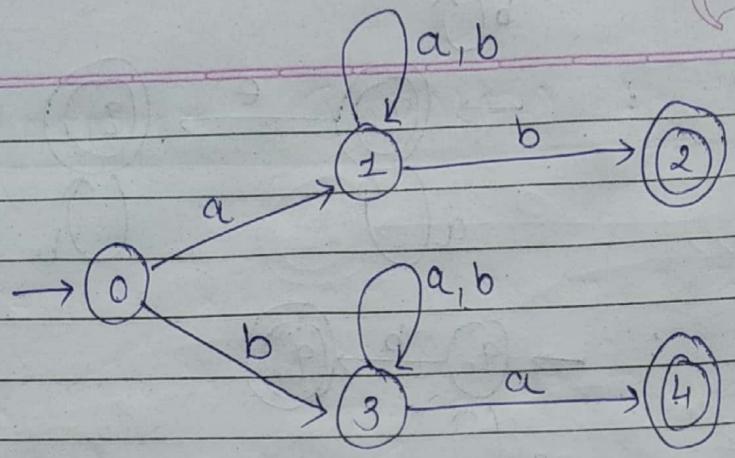
	a	b
0	01	0
1	3	2
2	–	–
3	–	–

	a	b
0	01	0
01	013	02
final states. (013)	013	02
final states. (02)	01	0



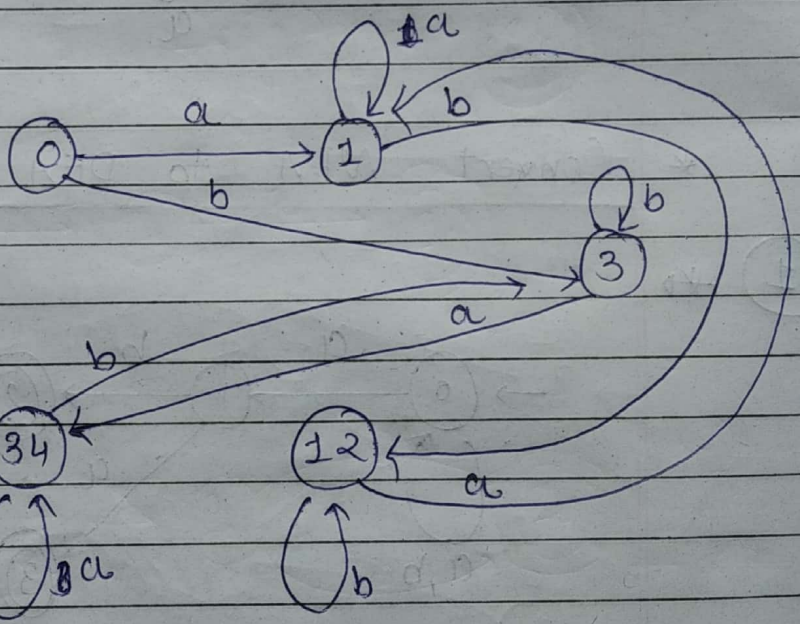
DFA

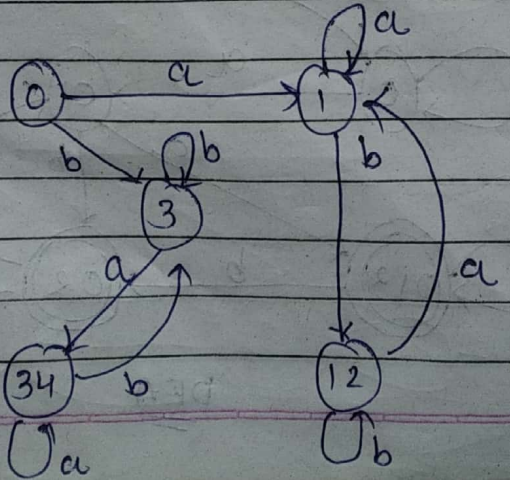
(2) *



	a	b
→ 0	1	3
1	1	12
(2)	–	–
(3)	34	3
(4)	–	–

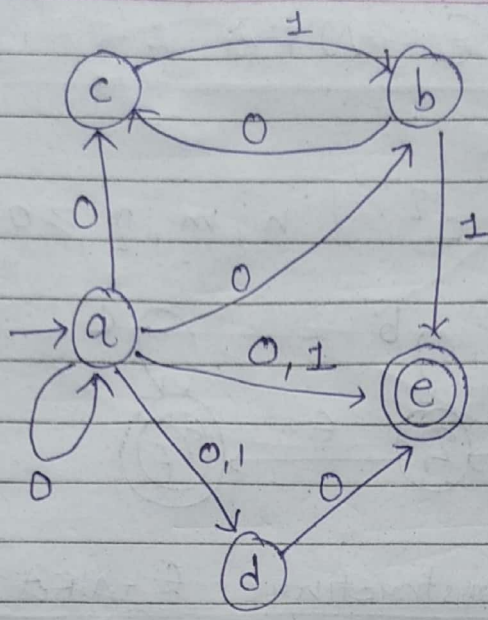
	a	b
0	1	3
1	1	12
3	34	3
12	1	12
34	34	3





done constructed in a good way.

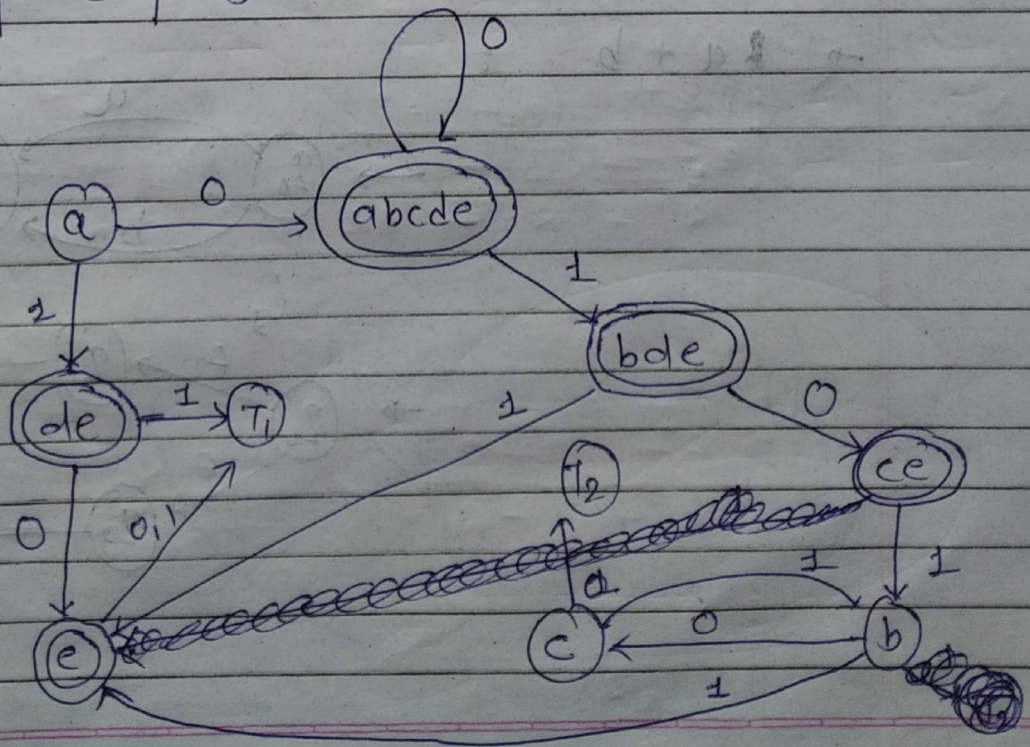
(3) *



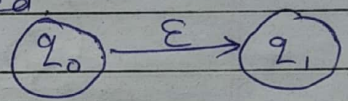
	0	1
→ a	abcde	de
b	ce	e
c	–	b
d	e	–
(e)	–	–

	0	1
a	abcde	de
(abcde)	abcde	bde
(de)	e	–
(bde)	ce	e
(e)	–	–
(ce)	–	b
b	c	e
c	–	b

Final states



* Diff. b/w DFA and NFA :-

DFA	NFA
→ Dead configuration is NOT allowed.	→ Dead configuration is allowed.
→ Multiple choices are NOT available corresponding to an input.	→ Multiple choices are available corresponding to an input.
→ ε (null) move is NOT allowed.	→ ε (null) move is allowed. ex :- 
→ Digital computers are deterministic.	→ Non deterministic feature is not associated with real computers
→ Designing and understanding is difficult. (As we have to create states for each symbol).	→ Designing and understanding is easy. (As it allows dead configuration and null moves).
→ No. of state will be high.	→ No. of state will be low.
→ If n = no. of states in NFA.	→ Then max no. of states posible is 2^n in DFA.

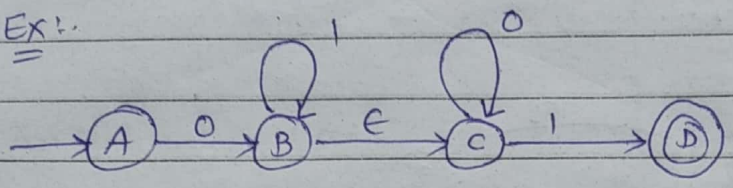
state	ϵ^* ex: (0) input	ϵ^*
A	A — A	A, B
B	B — ∅	B, C
C	C — C	C

Ex:-

* ε-NFA (Epsilon NFA) : Eliminating ε moves

→ $\delta : Q \times \Sigma \cup \epsilon \rightarrow 2^Q$

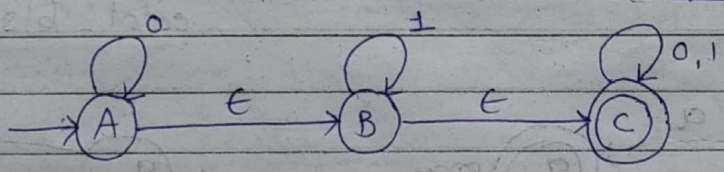
→ Ex:-



→ Every state on ε goes to itself.
For instance, A will go to itself after getting ε. (If mentioned then it goes to the mentioned state. ex:. B goes to C after getting ε.)

* Convert ε-NFA to it's equivalent NFA :-

NFA :-



* ε Closure (ϵ^*) :→ All the states that can be reached from a particular state only by seeing the ε symbol.

	States	0	1
Initial state →	A	{A, B, C}	{B, C}
	B	{C}	{B, C}
	C	{C}	{C}

	ϵ^* 0 ϵ^*
(A)	A — A — A, B, C
	B — ∅ — -
	C — C — C

{A, B, C}.

	ϵ^* 1 ϵ^*
(A)	A — ∅ — -
	B — B — B, C
	C — C — C

{B, C}.

	ϵ^* 0 ϵ^*
(B)	B — ∅ — -
	C — C — C

{C}.

	ϵ^* 1 ϵ^*
(B)	B — B — B, C
	C — C — C

{B, C}.

	ϵ^* 0 ϵ^*
(C)	C — C — C

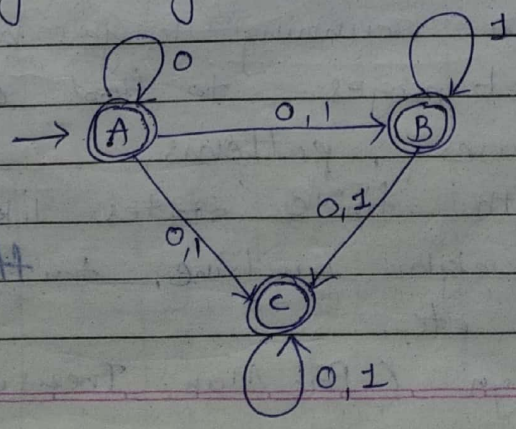
{C}

	ϵ^* 1 ϵ^*
(C)	C — C — C

{C}

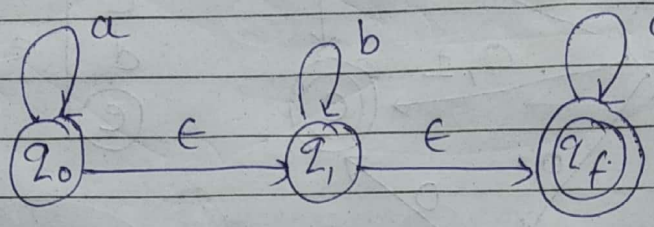
→ The final state will be any state that can reach the final state only by seeing ϵ. (Epsilon.)

→ In this case, A, B and C will be final states as it can reach the final state only by seeing ϵ.



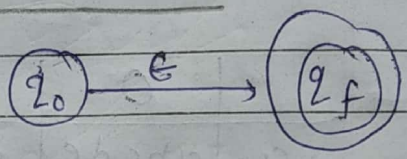
* Construct ϵ-NFA :-

→ $L = \{ a^m b^n c^q \mid n, m, q \geq 0 \}$

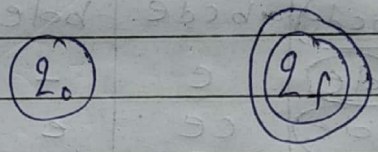


* Basics for constructing ϵ-NFA :-

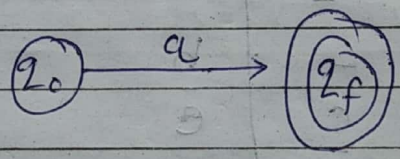
→ RE = ε :



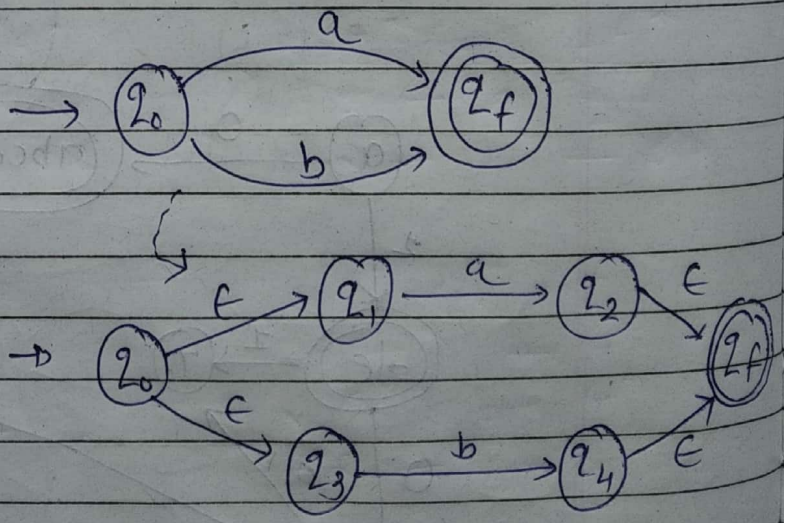
→ RE = ϕ :



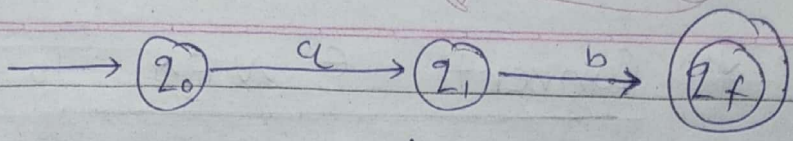
→ a :

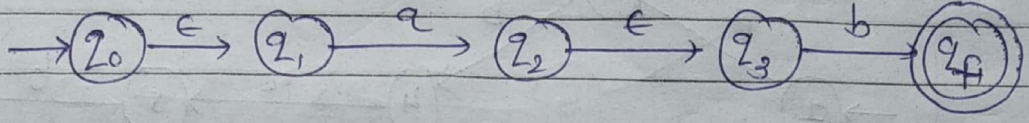


→ a + b :

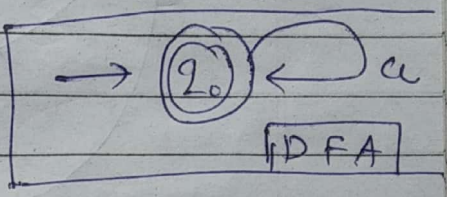


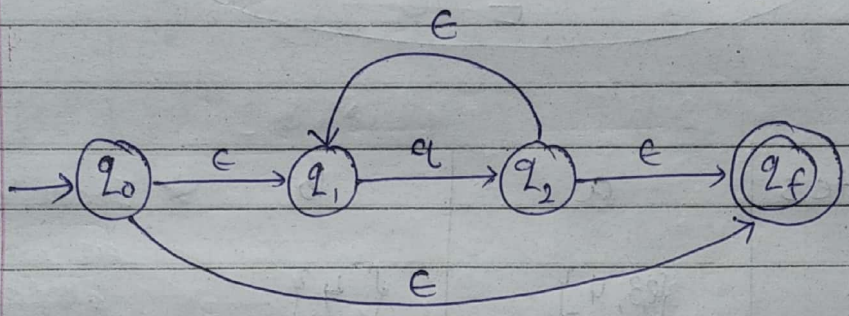
→ ab :





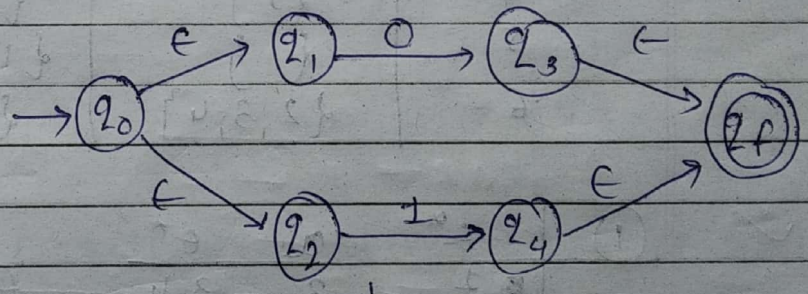
* → a^* :





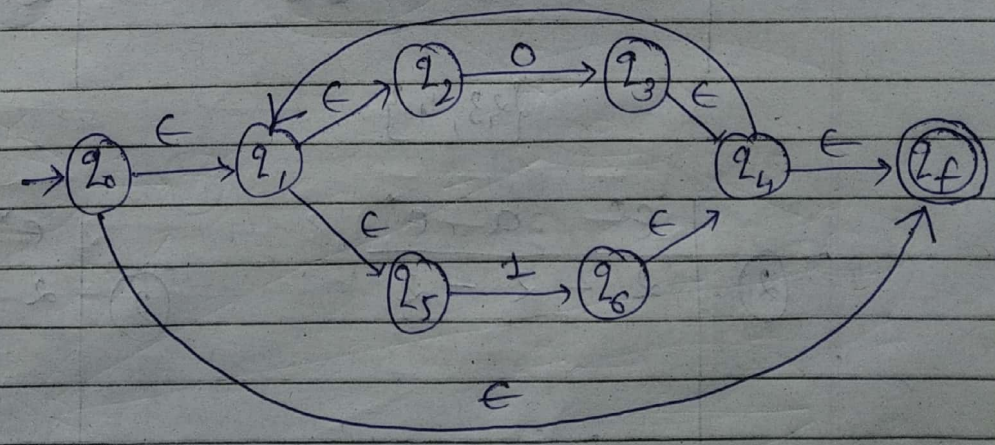
{ ϵ, R, RR, RRR, ... } accepted.
{ ϵ, a, aa, aaa, ... }

(1) * → $(0+1)^*$:



This is of $(0+1)$

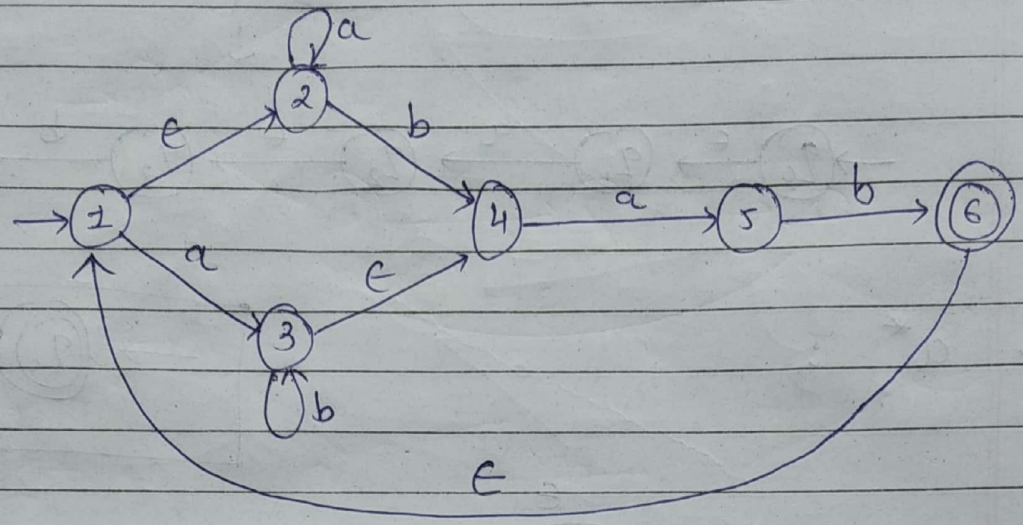
Connect starting & ending states Then add two new states: starting & ending. Connect them and connect last with first.



ϵ-NFA

* Convert ϵ-NFA to NFA :-

(1) **



States	a	b
1	{3,4}	{4}
2	{2}	{4}
3	{5}	{4}
4	{5}	{ }
5	{ }	{1,2,6}
6	{2,3,4}	{4}

* Include itself as well in ϵ* (Epsilon-closure)

✓ *

(1)

ϵ*	a	ϵ*
1	3	3,4
2	2	2

{2,3,4}

(1)

ϵ*	b	ϵ*
1	∅	—
2	4	4

{4}

(2)

ϵ*	a	ϵ*
2	2	2

{2}

(2)

ϵ*	b	ϵ*
2	4	4

{4}

	ϵ^*	a	ϵ^*
3	3	∅	-
	4	5	5
	{5}		

	ϵ^*	b	ϵ^*
3	3	3	4
	4	∅	-
	{4}		

	ϵ^*	a	ϵ^*
4	4	5	5
	{5}		

	ϵ^*	b	ϵ^*
4	4	∅	-
	{ }		

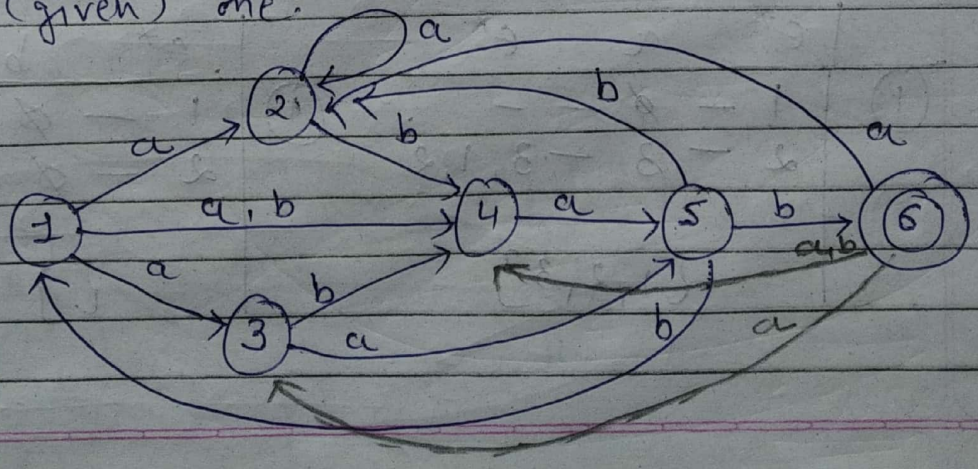
	ϵ^*	a	ϵ^*
5	5	∅	-
	{ }		

	ϵ^*	b	ϵ^*
5	5	6	1,2,6
	{1,2,6}		

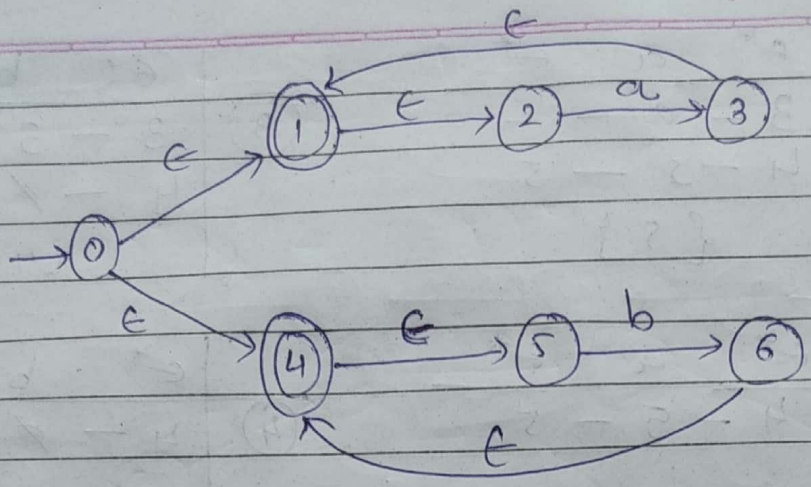
	ϵ^*	a	ϵ^*
6	1	3	3,4
	2	2	2
	6	∅	-
	{2,3,4}		

	ϵ^*	b	ϵ^*
6	1	∅	-
	2	4	4
	6	∅	-
	{4}		

→ 6 is the final state. Further, no state can reach the final state only by seeing ϵ.
∴ No other state is final state except default (given) one.



(2) **



States	a	b
0	{1,2,3}	{4,5,6}
1	{1,2,3}	{ }
2	{1,2,3}	{ }
3	{1,2,3}	{ }
4	{ }	{4,5,6}
5	{ }	{4,5,6}
6	{ }	{4,5,6}

	ϵ^*	a	ϵ^*
(0)	0	— ϕ —	-
	1	— ϕ —	-
	2	— 3 —	3,1,2
	4		
	5		
	{3,1,2}		

	ϵ^*	b	ϵ^*
(0)	0	— ϕ —	-
	1	— ϕ —	-
	2	— ϕ —	-
	4		
	5	— 6 —	6,4,5
			{4,5,6}

	ϵ^*	a	ϵ^*
(1)	1	— ϕ —	-
	2	— 3 —	3,1,2
	{1,2,3}		

	ϵ^*	b	ϵ^*
(1)	1	— ϕ —	-
	2	— ϕ —	-
	{ }		

	ϵ^*	a	ϵ^*
(2)	2	3	3,1,2

$\{1,2,3\}$

	ϵ^*	b	ϵ^*
(2)	2	ϕ	-

$\{\}$

	ϵ^*	a	ϵ^*
(3)	1	ϕ	-
	2	3	1,2,3
	3	ϕ	-

$\{1,2,3\}$

	ϵ^*	b	ϵ^*
(3)	1	ϕ	-
	2	ϕ	-
	3	ϕ	-

$\{\}$

	ϵ^*	a	ϵ^*
(4)	4	ϕ	-
	5	ϕ	-

$\{\}$

	ϵ^*	b	ϵ^*
(4)	4	ϕ	-
	5	6	6,4,5

$\{4,5,6\}$

	ϵ^*	a	ϵ^*
(5)	5	ϕ	-

$\{\}$

	ϵ^*	b	ϵ^*
(5)	5	6	4,5,6

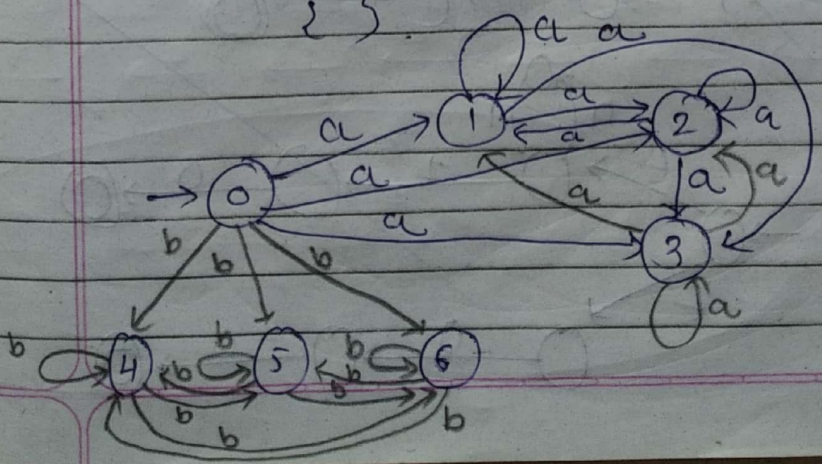
$\{4,5,6\}$

	ϵ^*	a	ϵ^*
(6)	6	ϕ	-
	4	ϕ	-
	5	ϕ	-

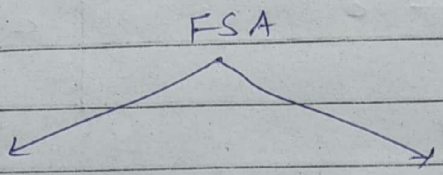
$\{\}$

	ϵ^*	b	ϵ^*
(6)	6	ϕ	-
	4	ϕ	-
	5	6	4,5,6

$\{4,5,6\}$



* Limitations and applications of FSA :
(Finite State Automata)



without o/p	with o/p
↓	↓
DFA	Mealy Machine
NFA	Moore Machine
∈-NFA	

* Limitations :-

→ limited memory
→ strings without comparisons
→ linear power

* Applications :-

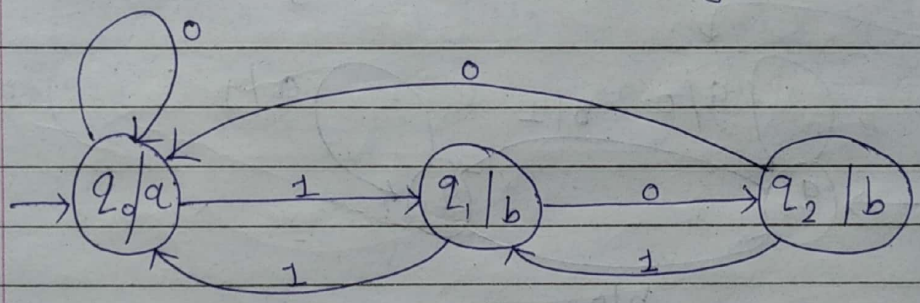
→ word processor programs
→ digital logic design
→ lexical analyzer
→ switching circuit design
→ Text editors, etc.
→ Software for scanning large bodies of text such as web pages to find occurance of words, phrases, patterns.
→ Software with finite states like vending machines, weigh machine, traffic lights, Toll machine etc.
→ Game design (Pac Man, Treasure Hunt etc.)

* Moore machine :-

→ FSA with output.

→ $M_0 = \{ Q, \Sigma, \Delta, \delta, \lambda, q_0 \}$

Q = finite set of states
Σ = input symbol (alphabet)
δ = transition function
$Q \times \Sigma$
q_0 = start state
Δ = Output symbol (a, b)
λ = Output function
$\lambda : Q \rightarrow \Delta$



→ input length = n
∴ o/p length = n+1

→ I/p = 00110

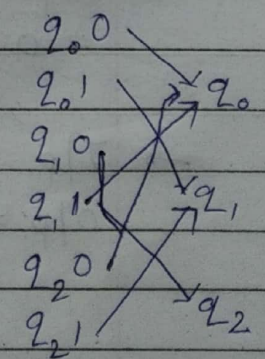
o/p = a aabaa

By default starting state o/p

Transition Table	Current	Next state		o/p
		0	1	
	q_0	q_0	q_1	a
	q_1	q_2	q_0	b
	q_2	q_0	q_1	b

$\lambda : Q \rightarrow \Delta$

q_0 (0,1) $q_0 0$, $q_0 1$, $q_1 0$, $q_1 1$, $q_2 0$, $q_2 1$
q_1
q_2



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

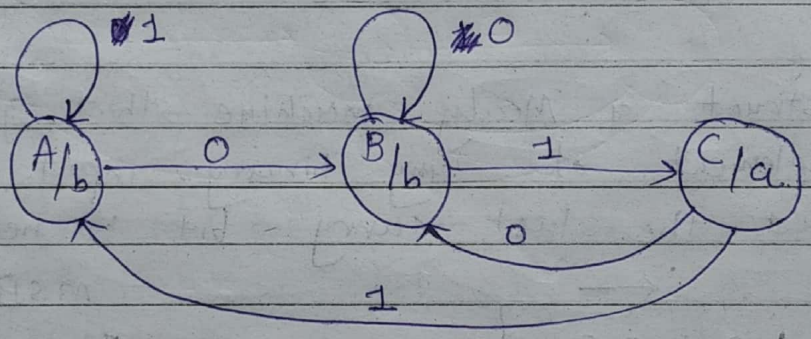
* Construction of Moore machine :-

(1) ** Construct a moore machine that prints 'a' whenever the sequence '01' is encountered in any input binary string. $\Sigma = \{0, 1\}$, $\Delta = \{a, b\}$

→ Moore machines consists of result in it's own states.

01

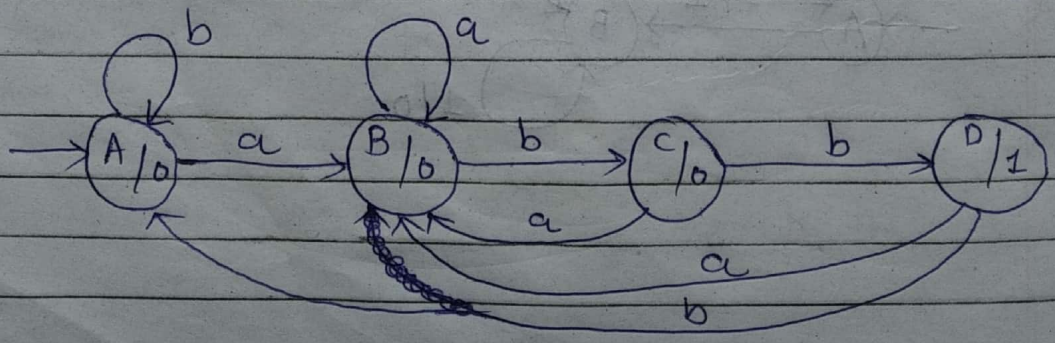
→ Here, firstly creating DFA and then assigning 'a' where 01 sequence is achieved and writing 'b' elsewhere.



(2) ** Construct a Moore machine that counts the occurences of the sequence 'abb' in any input string over {a, b}.

→ $\Sigma = \{a, b\}$, $\Delta = \{0, 1\}$.

abb → 1
else → 0



(3) ** For the following Moore machine the input alphabet is $\Sigma = \{a, b\}$ and the output alphabet $\Delta = \{0, 1\}$. Run the following input sequence and find the respective output:

(1) aabab (2) abbb (3) ababb.

states	a	b	outputs
q_0	q_1	q_2	0
q_1	q_2	q_3	0
q_2	q_3	q_4	1
q_3	q_4	q_4	0
q_4	q_0	q_0	0

→ i)

	a	a	b	a	b
q_0	q_1	q_2	q_4	q_0	q_2
0	0	1	0	0	1

ii)

	a	b	b	b
q_0	q_1	q_3	q_4	q_0
0	0	0	0	0

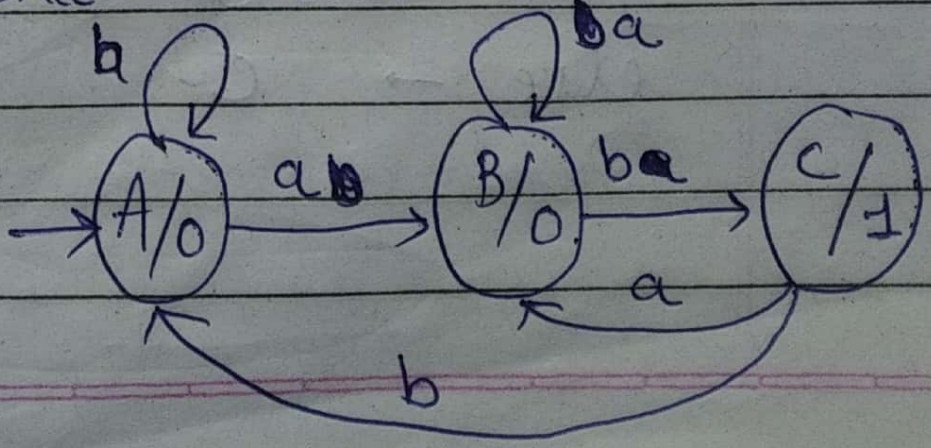
iii)

	a	b	a	b	b
q_0	q_1	q_3	q_4	q_0	q_2
0	0	0	0	0	1

* Construct Moore Machine :-

(1) * Construct Moore machine that takes set of all strings over {a, b} as input and prints 1 as output for every occurence of ab.

→ a, b, ab, ba, aa, bb,
0 0 01 00 00 00

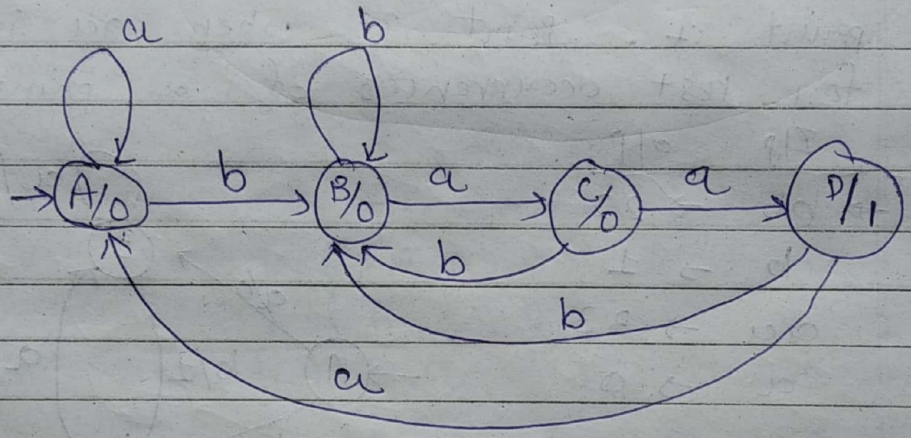


(2) ** Construct a Moore machine that takes set of all strings over {a, b} and count number of occurences of string 'baa'.

Steps: Construct DFA which accepts all the strings:

a, b, aa, bb, aab, baa, baaaa, baabb, bbbaa, abbaa, bbbabaa

Then assign results to the states.

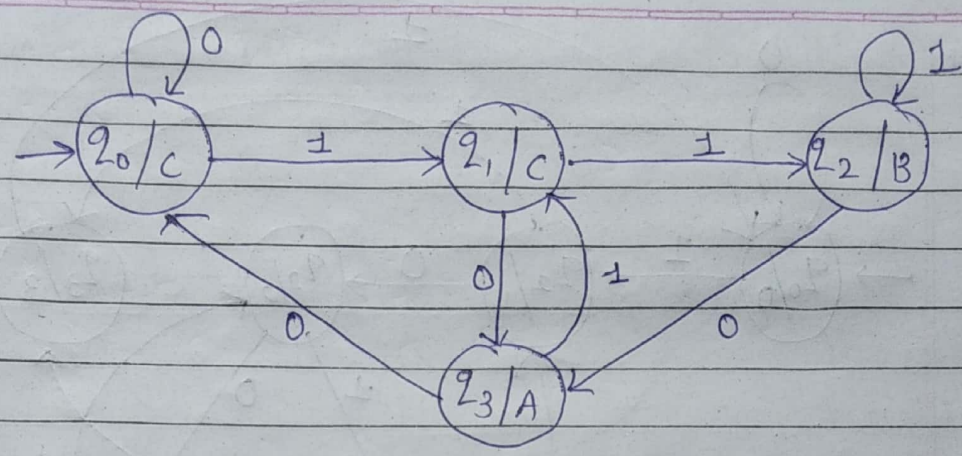


(3) ** Construct a moore machine that takes set of all strings over {0,1} and, produce 'A' as output if input ends with '10', or produce 'B' as output if input ends with '11', otherwise produce 'C'.

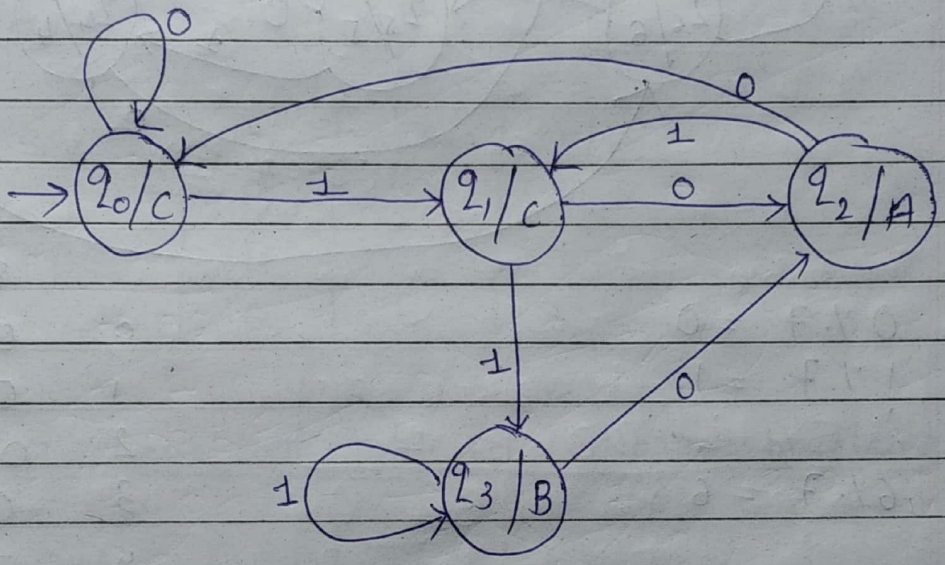
{0, 1}

	I/p	O/p
ends with	... 10 →	A
"	... 11 →	B
	else →	C

Strings: 1, 0, 11, 01, 10, 00, 101, 000, 111, 1010, 1110, 1001, 1000, 101010, 0010010, 1110010, ...



OR



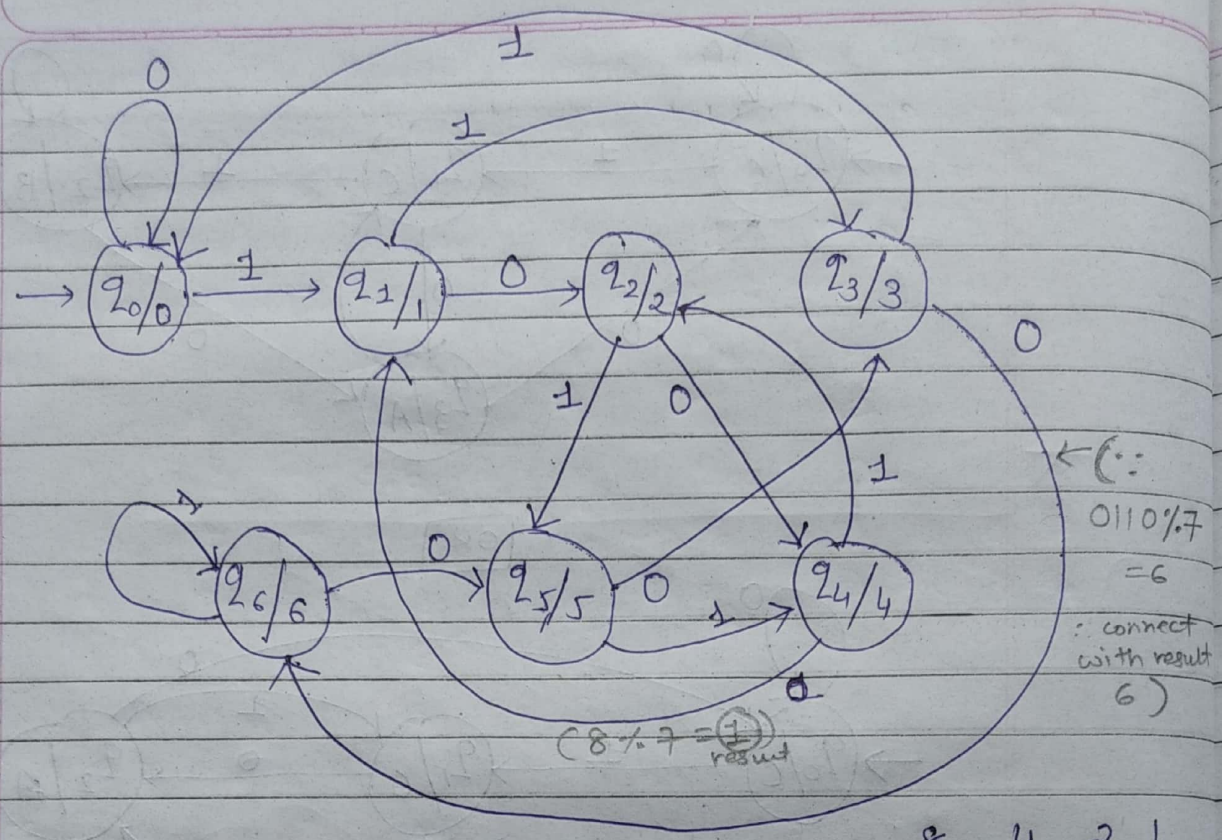
(4)** Take binary number as input and print its residual module of 7.

Input symbols : $\Sigma = \{0, 1\}$

Here, we want to print residual module 7.

∴ Dividing any binary number with 7, the remainder will be 0, 1, 2, 3, 4, 5, 6 only.

Now, for each state we have to point result to construct moore machine.



→ 0 % 7 = 0
1 % 7 = 1
⋮ ⋮
6 % 7 = 6

→ Now, for each binary number, create moore machine by constructing DFA first.

→ For example, for 3 (0011) modulo 7 will be 4.
∴ resultant state for 0011 should be (q3/3)
where 3 is the remainder.

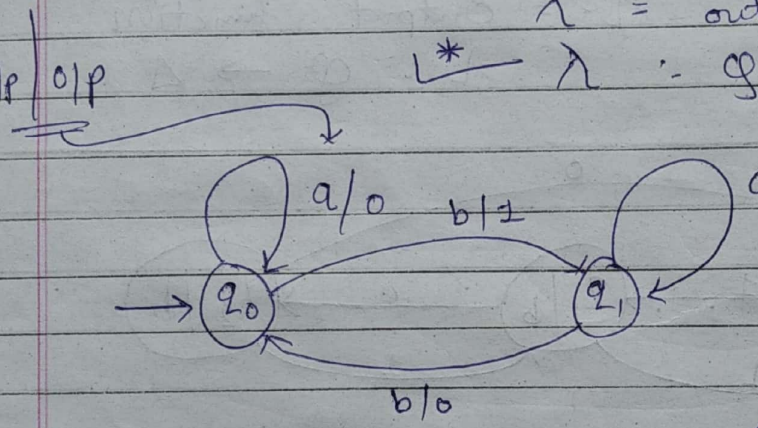
	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0

* Mealy machine :-

→ FSA with output.

→ $M_e = \{ Q, \Sigma, \Delta, \delta, \lambda, q_0 \}$

Q = finite set of states.
Σ = input alphabet
δ = Transition function
$\delta : Q \times \Sigma \to Q$
q_0 = initial state.
Δ = output symbol (0,1)
λ = output function
* $\lambda : Q \times \Sigma \to \Delta$



$\lambda : Q \times \Sigma \to \Delta$

q_0 (a, b)

q_1

→ Input length = n
∴ output length = n.

→ I/p = abba
O/p = 0100

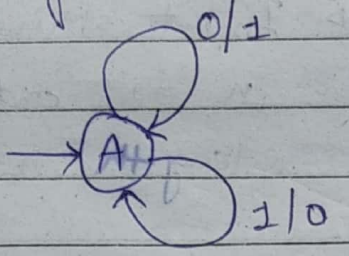
$q_0 a \longrightarrow q_0$
$q_0 b \longrightarrow q_1$
$q_1 a \longrightarrow q_1$
$q_1 b \longrightarrow q_0$

Current	Next state			
	I/p = a		I/p = b	
q_0	q_0	0	q_1	1
q_1	q_1	1	q_0	0
	state	o/p	state	o/p

Transition Table

* Construction of Mealy machine :-

(1) ** Construct a Mealy machine that produces the 1's complement of any binary ~~number~~ input string.

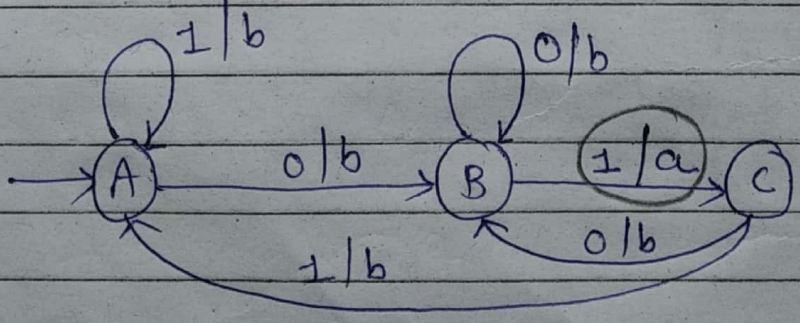


I/p : 10100
O/p : 01011

(2) ** Construct a Mealy machine that prints 'a' whenever the sequence '01' is encountered in any input binary string. $\Sigma = \{0, 1\}$, $\Delta = \{a, b\}$

→ First create DFA for above example and then when we got 01 sequence in the DFA (as an output) then print o/p 'a' otherwise 'b'.

01



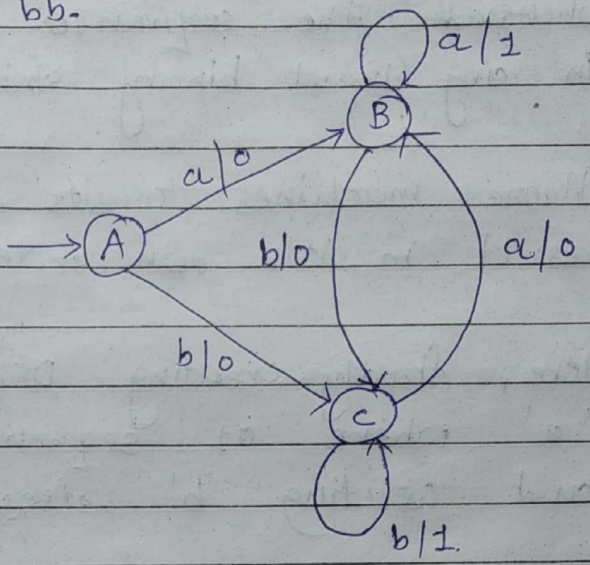
→ There is no final state in Mealy machine.

→ 0110 → 1000
bqbb bbbb

2's comple. = 1's comple. + 1

(3) ** Design a Mealy machine accepting the language consisting of strings from Σ^*, where $\Sigma = \{a, b\}$ and the strings should end with either aa or bb.

→ aa → 1
bb → 1

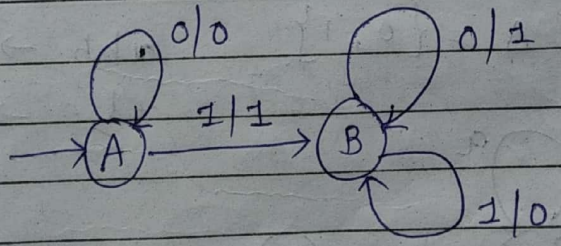


(4) ** Construct a Mealy machine that gives 2's complement of any binary input. (Assume that the last carry bit is neglected.)

MSB ← LSB

→ ① 1 0 1 0 0
2's: 0 1 1 0 0

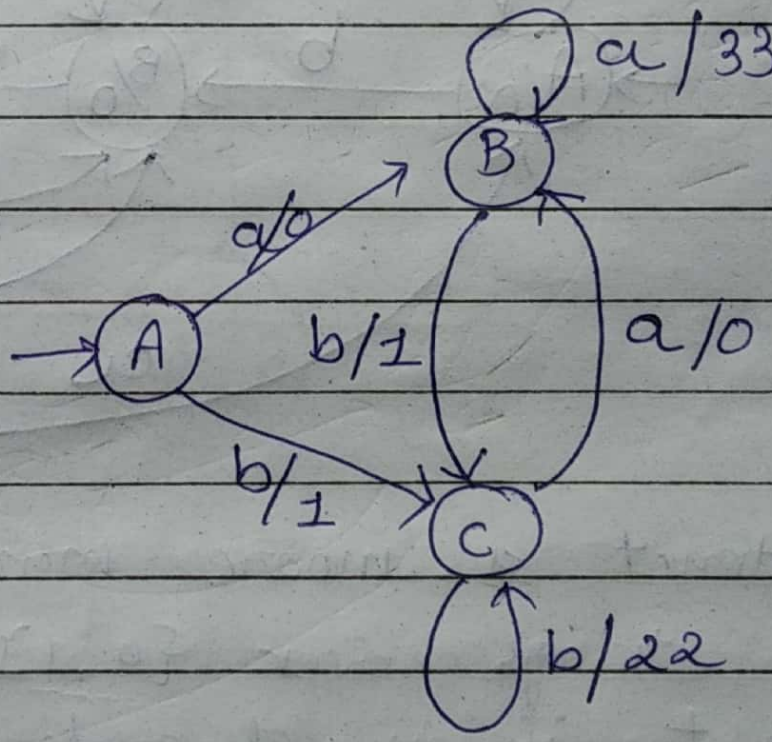
I/p : 1 0 1 0 0
1's : 0 1 0 1 1
+1 : 1
2's compl:- 0 1 1 0 0



* Construct Mealy Machine :-

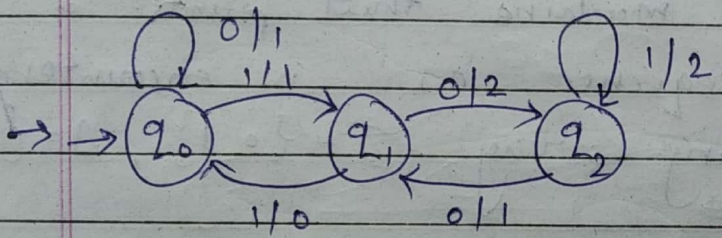
(1) ** Take string over {a, b}. Print 22 when bb is occurred, for rest occurencies of b print 1. Print 33 when aa is occurred, for rest occurrences of a print 0.

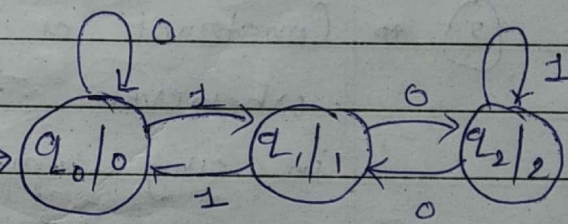
I/p	o/p
→ bb	→ 22
b	→ 1
aa	→ 33
a	→ 0.



Date
Page

* Mealy machine	* Moore machine.
→ output of the mealy machine depends upon present state and present input.	→ output only depends on present state. (Independent on input)
→ Input string of length n ⇒ output string length n.	→ Input string length n ⇒ output string length n+1.
→ $\lambda : Q \times \Sigma \to \Delta$	→ $\lambda : Q \to \Delta$





→ Mealy output is asynchronous.	→ Moore machine output is synchronous with clock

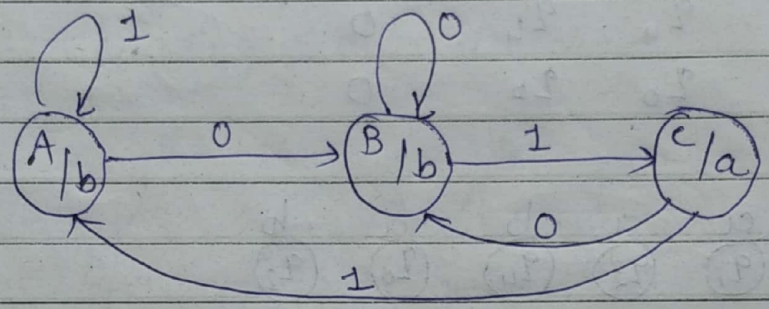
* Conversion of Moore machine to Mealy machine:

(1) Construct a Moore machine that prints 'a' whenever the sequence '01' is encountered in any input binary string and then convert it to its equivalent mealy machine.

→ $\Sigma = \{0, 1\}$, $\Delta = \{a, b\}$.

→ Moore machine :-

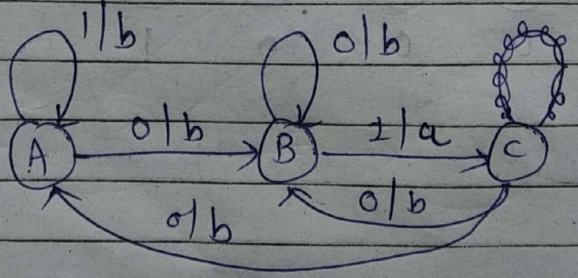
01 → a



→ Transition table :-

State	0	1	Output (Associated with that state)
→ A	B	A	b
B	B	C	b
C	B	A	a

→ Mealy machine:



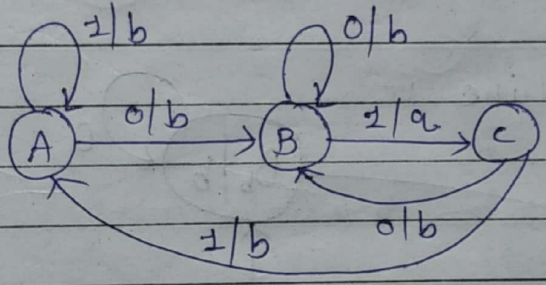
Transition table of Moore machine :

state	0	1	output
→ A	B b	A b	b
B	B b	C a	b
C	B b	A b	a.

Create Mealy machine from the table.

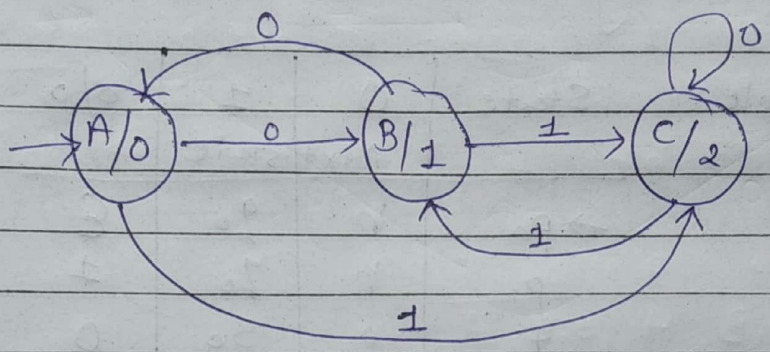
→ transition table for Mealy machine,

State	0	1
A	B b	A b
B	B b	C a
C	B b	A b



* Convert Moore machine to Mealy Machine.

(1) **



Moore machine

Transition table :-

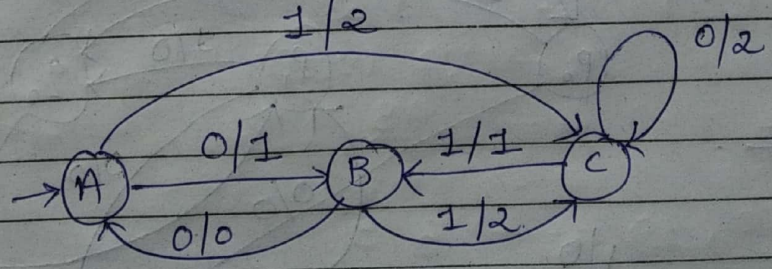
State	0	1	o/p
A	B	C	0
B	A	C	1
C	C	B	2

From transition table of Moore machine :-

Write Associated output from tra. table of Moore Machine.

State	0	1
A	B 1	C 2
B	A 0	C 2
C	C 2	B 1

Mealy machine :-



(2) ** Convert the given Moore machine into Mealy machine using transition table of Moore machine

Tra. table :-

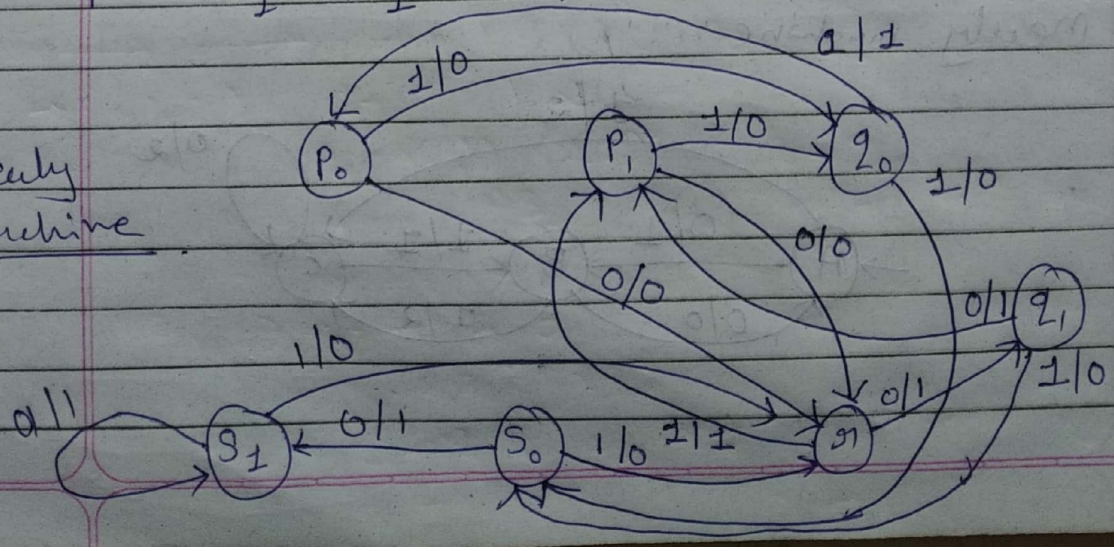
State	0	1	o/p
	Next state		
→ p_0	r	q_0	ε
p_1	r	q_0	1
q_0	p_1	s_0	0
q_1	p_1	s_0	1
r	q_1	p_1	0
s_0	s_1	r	0
s_1	s_1	r	1

Tra. table of Mealy machine :-

State	state 0	o/p 0	state 1	o/p 1
→ p_0	r	0	q_0	0
p_1	r	0	q_0	0
q_0	p_1	1	s_0	0
q_1	p_1	1	s_0	0
r	q_1	1	p_1	1
s_0	s_1	1	r	0
s_1	s_1	1	r	0

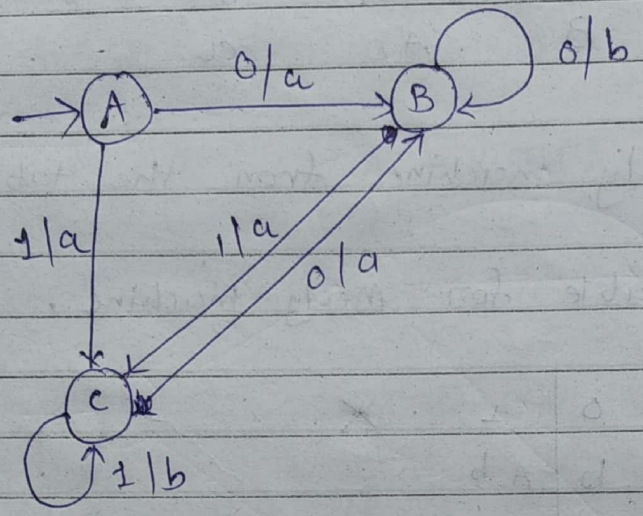
→ Associated o/p from t.t. of moore machine

Mealy machine



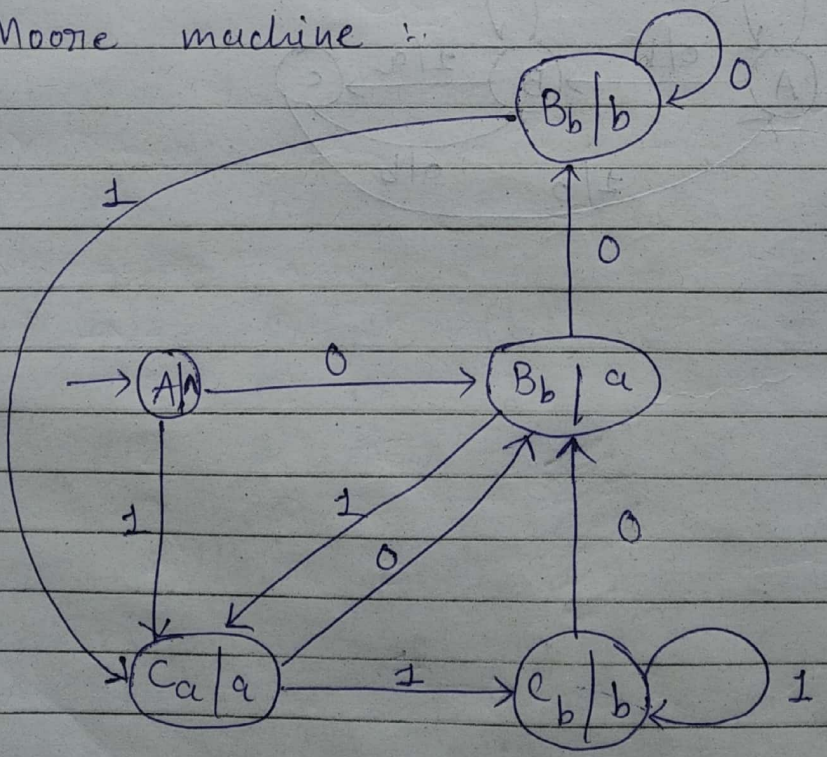
* Conversion of Mealy machine to Moore machine:

(1) ** Convert the following Mealy machine to it's equivalent Moore machine

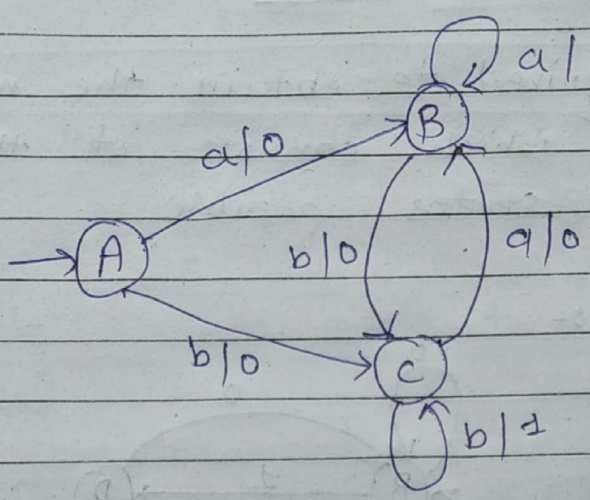


→ $\Sigma = \{0, 1\}$, $\Delta = \{a, b\}$

→ Moore machine :-

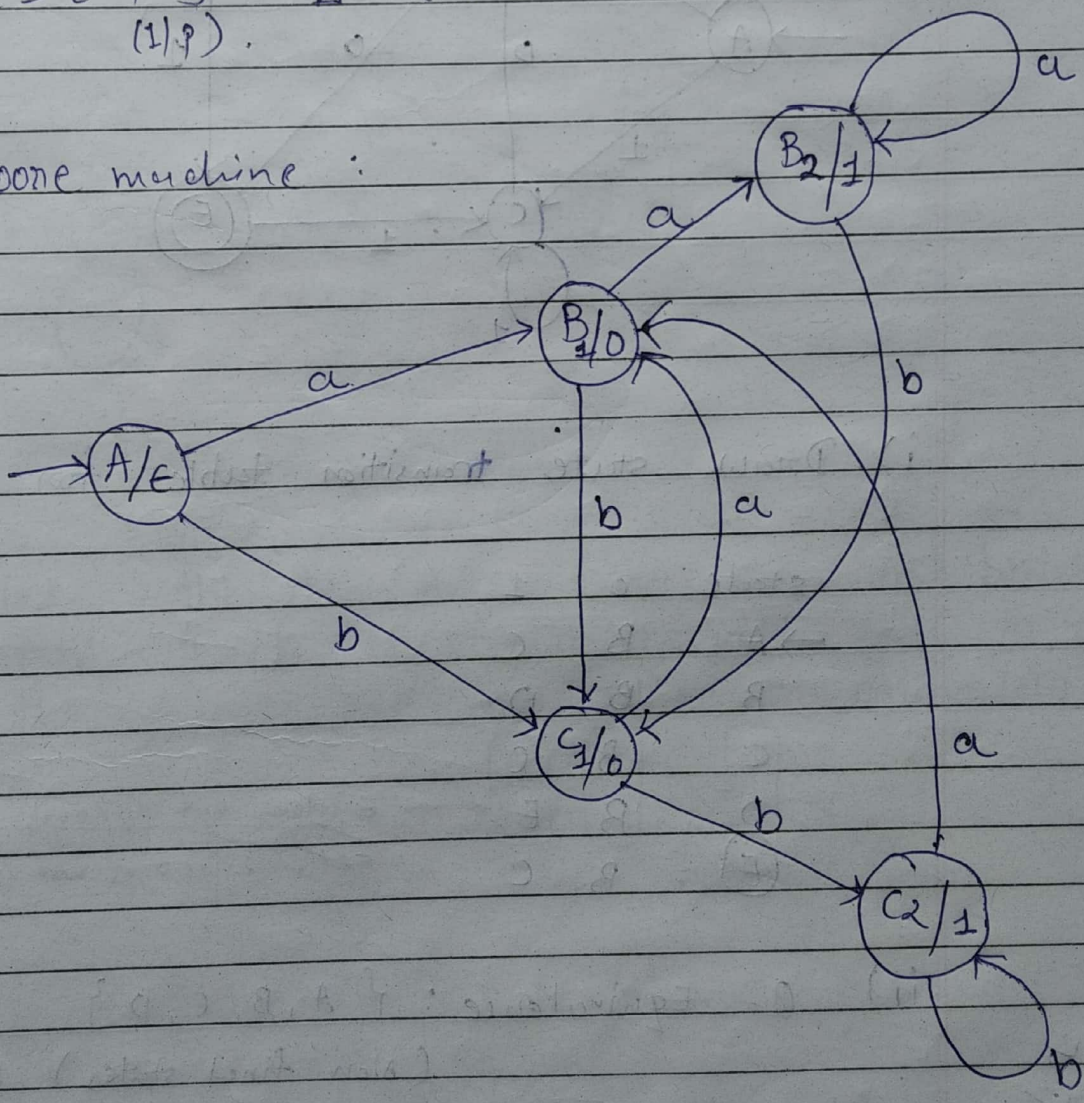


(2) ** Moore machine from Mealy machine :-



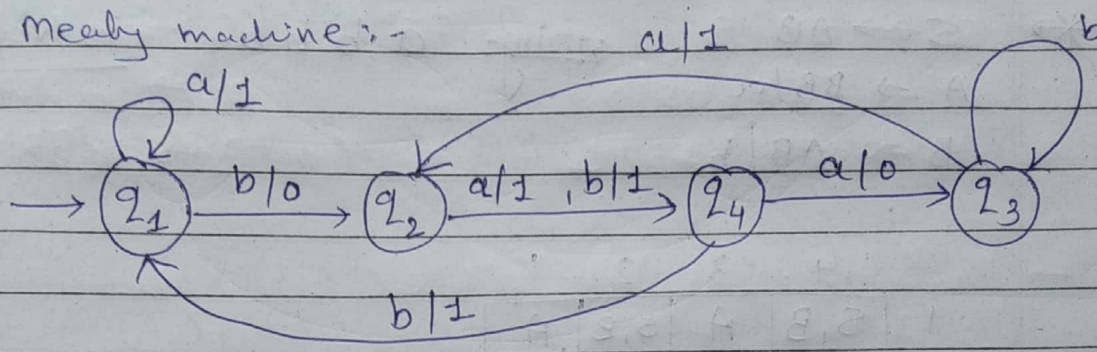
$\Sigma = \{a, b\}$ $\Delta = \{0, 1\}$ (o/p)
(I/P).

Moore machine :

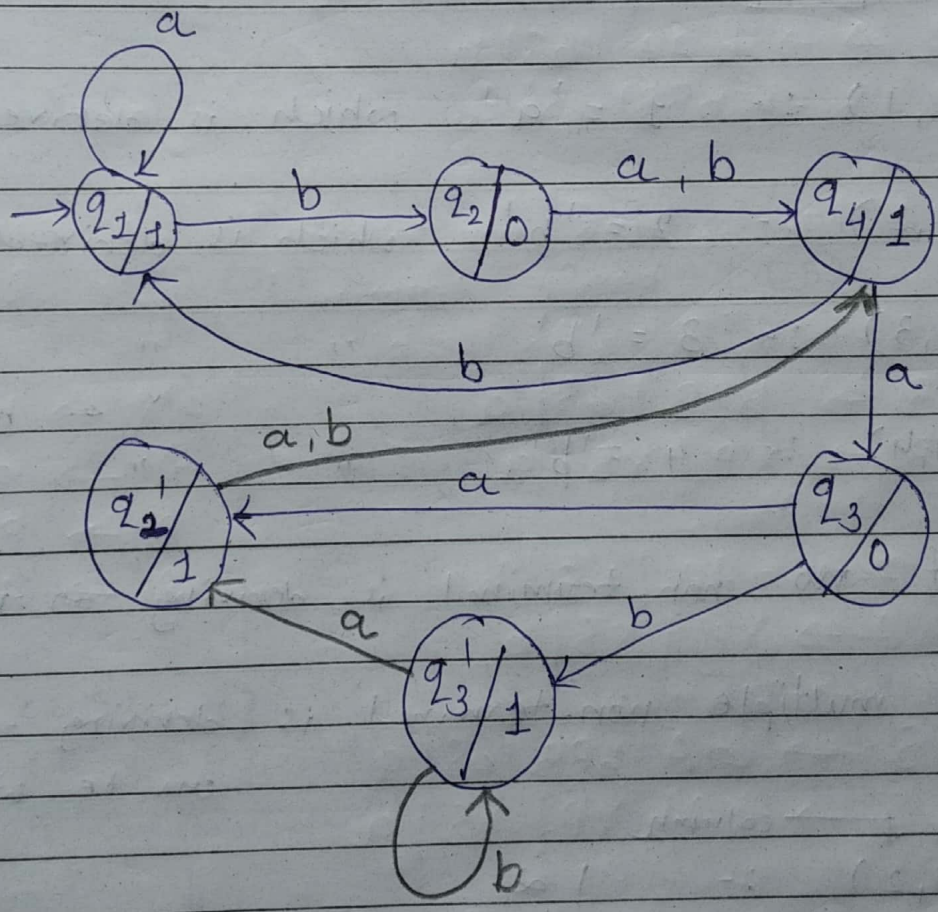


* Convert Mealy machine to Moore machine.

(1) ** Mealy machine :-



Moore machine :-



* Context free grammar :-

→ $G = (V, T, P, S)$

** Examples :-

1) Construct a CFG having any number of a's over the set $\Sigma = \{a\}$.

→ $S \rightarrow aS$ $\quad R.E = a^*$

$S \rightarrow \epsilon$

Refer websites or PPTs of GitHub for more examples.

* Remove Null Production :-

(1) ** $S \to aS \mid A$
$A \to \epsilon$

→ Find nullable non-terminal. [direct / indirect]
{A, S}

→ Put ϵ in place of {A, S} in RHS and append the result.

$S \to aS \mid A$ Remove $A \to \epsilon$
$\Rightarrow S \to aS \mid A \mid a \mid \epsilon$ (∵ it produces null)
$\Rightarrow S \to aS \mid a \mid \epsilon$ (∵ Removed A)

Here ϵ is the part of language. Therefore, ϵ is not removed from language

(2) ** $S \to ABC$
$A \to aA \mid \epsilon$
$B \to bB \mid \epsilon$
$C \to c$

→ {S, A, B} nullable.
↳ S is not nullable because replacing A, B with ϵ will still result in 'C'.

→ $S \to ABC$ (PUT → A=ϵ, B=ϵ → A&B=ϵ)
∴ $S \to ABC \mid BC \mid AC \mid C$

→ $A \to aA \mid \epsilon$
∴ $A \to aA \mid \epsilon \mid a$
∴ $A \to aA \mid a$ (∵ Remove ϵ).

→ B → bB | ∈

∴ B → bB | ∈ | b

∴ B → bB | b (∵ Remove ∈)

∴ Final grammer,

S → ABC | BC | AC | C

A → aA | a

B → bB | b

C → c

(3) ** S → aSb | aAb

A → ∈

→ Nullable : {A}

∴ S → aSb | aAb | ab

∴ S → aSb | ab (∵ A → ∈ removed)

(4) ** S → AbaC

A → BC

B → b | ∈

C → D | ∈

D → d

→ Nullable : {A, B, C}

∴ S → AbaC | baC | Aba | ba

A → Bc | c | B

B → b

C → D

D → d

(5) ** $S \to aSa \mid bSb \mid \epsilon$
$A \to abb \mid bBa$
$B \to aB \mid bB \mid \epsilon$

→ Nullable : {B, S}

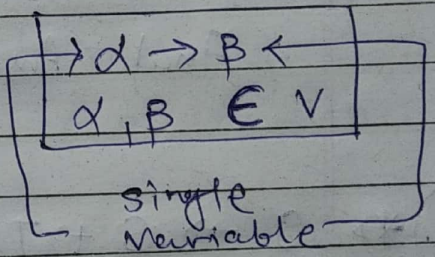
$S \to aSa \mid bSb \mid \epsilon \mid aa \mid bb$
$\therefore S \to aSa \mid bSb \mid aa \mid bb \mid \epsilon$

$A \to abb \mid bBa$
$\therefore A \to abb \mid bBa \mid ba$

$B \to aB \mid bB \mid \epsilon$
$\therefore B \to aB \mid bB \mid \epsilon \mid a \mid b$
$\therefore B \to aB \mid bB \mid a \mid b$ (∵ Remove ϵ)

* Remove Unit Production :-

(1) ** $S \to AB$
$A \to a$
$B \to \epsilon \mid b$
$C \to D$
$D \to E$
$E \to a$.



→ Recognize unit productions,

$B \to C$, $C \to D$, $D \to E$

Here, $E \to a$
$\therefore B \to a$
$C \to a$

$\therefore S \to AB$
$A \to a$
$B \to a \mid b$
~~$C \to a$~~

Removed
$D \to E$, $C \to D$
$E \to a$,
($C \to a$ (∵ $B \to a \mid b$))

(2) →

$S \to aA \mid B$

$A \to ba \mid bb$

$B \to A \mid bba$

→ Recognize unit production,

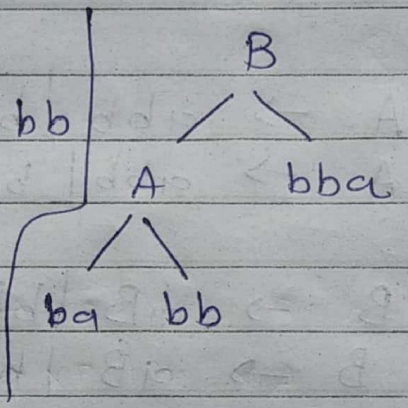
$S \to B$, $B \to A$ $\Rightarrow$ $S \to A$

$S \to aA \mid B$

$\therefore S \to aA \mid bba \mid ba \mid bb$

$A \to ba \mid bb$

$B \to ba \mid bb \mid bba$



Here, B is not reachable from S.

$\therefore$ Removing B.

$\therefore$ Final Answer : $S \to aA \mid bba \mid ba \mid bb$

$A \to ba \mid bb$

CFG to CNF :-

1) Eliminate null production
2) Eliminate unit production
3) Rewrite to fulfill the rule of CNF as given below:

$A \to BC$ (Two non-terminal at RHS)
$A \to a$ (one single terminal at RHS)

(1) $S \to AB$
$A \to aAA \mid \epsilon$
$B \to bBB \mid \epsilon$

→ Nullable : {A, B, S}

$S \to AB \mid B \mid A \mid \epsilon$
$A \to aAA \mid a \mid aA$ ← * replacing single A or B with ϵ.
$B \to bBB \mid b \mid bB$

→ Unit production : $S \to B$, $S \to A$.

$S \to AB \mid bBB \mid b \mid bB \mid aAA \mid a \mid aA \mid \epsilon$

∴ $S \to AB \mid bBB \mid bB \mid b \mid aAA \mid aA \mid a \mid \wedge$
$A \to aAA \mid a \mid aA$
$B \to bBB \mid b \mid bB$

→ Rewrite :- two non-terminal at RHS
single terminal at RHS

$S \to AB \mid bBB \mid bB \mid b \mid aAA \mid aA \mid a \mid \epsilon$

$A \to aAA \mid a \mid aA$

$B \to bBB \mid b \mid bB$

→ $A \to XP \mid a \mid XA$

$B \to Y\varphi \mid b \mid YB$

$S \to AB \mid Y\varphi \mid YB \mid b \mid XP \mid XA \mid a \mid \epsilon$

$X \to a$

$Y \to b$

$P \to AA$

$\varphi \to BB$

which is in CNF: $S \to AB \mid Y\varphi \mid YB \mid b \mid XP \mid XA \mid a \mid \epsilon$

$A \to XP \mid a \mid XA$

$B \to Y\varphi \mid b \mid YB$

$X \to a$

$Y \to b$

$P \to AA$

$\varphi \to BB$

(2) → $S \to AACD$

$A \to aAb \mid \epsilon$

$C \to aC \mid a$

$D \to aDa \mid bDb \mid \epsilon$

→ Nullable :- $\{D, A\}$

$S \to AACD \mid AAC \mid ACD \mid CD \mid C$

$A \to aAb \mid ab$

$C \to aC \mid a$

$D \to aDa \mid bDb \mid aa \mid bb$

→ Eliminating unit production :-

$S \to AACD \mid AAC \mid ACD \mid CD \mid aC \mid a$ | $S \to C$
$A \to aAb \mid ab$ | AC
$C \to aC \mid a$
$D \to aDa \mid bDb \mid aa \mid bb$

→ Rewriting :- two non-terminal at RHS
Single terminal at RHS

$\therefore D \to XDX \mid YDY \mid XX \mid YY$ | ~~$P \to aa$~~
$\therefore D \to XM \mid YN \mid XX \mid YY$ | ~~$Q \to bb$~~
$X \to a$
$Y \to b$
$M \to DX$
$N \to DY$

$\therefore C \to aC \mid a$
$\therefore C \to XC \mid a$

$\therefore A \to aAb \mid ab$
$\therefore A \to XAY \mid XY$
$\therefore A \to XL \mid XY$ | $L \to AY$

$\therefore S \to AACD \mid AAC \mid ACD \mid CD \mid aC \mid a$ | $P \to AA$
$\therefore S \to PQD \mid PC \mid AQ \mid CD \mid XC \mid a \mid AC$ | $Q \to CD$

$\therefore$ CNF :-

$S \to PQ \mid PC \mid AQ \mid CD \mid XC \mid a \mid AC$
$A \to XL \mid XY$
$C \to XC \mid a$
$D \to XM \mid YN \mid XX \mid YY$
$X \to a$
$Y \to b$
$M \to DX$

$N \to DY$
$L \to AY$
$P \to AA$
$Q \to CD$

→ ** Convert the given grammar to CNF :-

$1 \to a$
$0 \to b$

$S \to 1A \mid B$
$A \to 1AA \mid 0S \mid 0$
$B \to 0BB \mid 1S \mid 1$

$S \to aA \mid B$
$A \to aAA \mid bS \mid b$
$B \to bBB \mid aS \mid a$

→ Nullable : Nonnullable

→ Unit production :- $S \to B$

$\therefore S \to aA \mid bBB \mid aS \mid a$
$A \to aAA \mid bS \mid b$
$B \to bBB \mid aS \mid a$

→ Rewrite : two non-terminal at RHS
single terminal at RHS.

$B \to bBB \mid aS \mid a$
$\therefore B \to XP \mid YS \mid a$

$A \to aAA \mid bS \mid b$
$\therefore A \to YQ \mid XS \mid b$

$X \to b$
$Y \to a$
$P \to BB$
$Q \to AA$

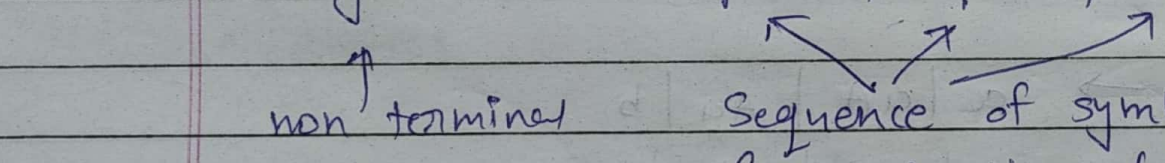
$\therefore S \to aA \mid bBB \mid aS \mid a$
$S \to YA \mid XP \mid YS \mid a$

$\therefore$ CNF : $S \to YA \mid XP \mid YS \mid a$
$A \to YQ \mid XS \mid b$
$B \to XP \mid YS \mid a$
$X \to b$
$Y \to a$
$P \to BB$
$Q \to AA$

* Backus - Naur Form :-

→ It's a notation technique used to represent CFG.
→ Used for specifying syntax of a language.

→ <Symbol> ::= Expr.1 | Expr 2 | Expr 3 ...



(1) → Optional symbols are written in square brackets.
[]

(2) → Repeating the symbol for 0 or more time
* [asterik] is used.

(3) → Repeating the symbol for atleast 1 time
+ is used.

(4) → Alternative rules are seperated by vertical bar |.

(5) → The group of items must be enclosed within brackets.

* Pushdown Automata :-

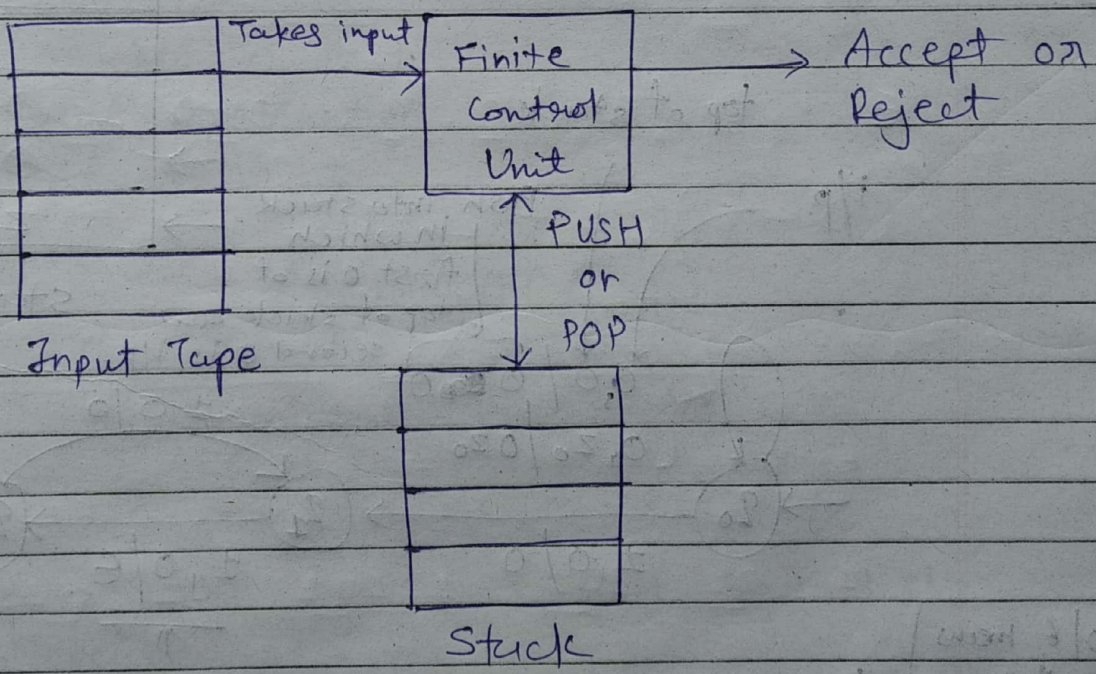
→ It's a way to implement a CFG in a similar way we design finite Automata for regular ~~language~~ grammar.

→ PDA has more memory than FSM.

→ PDA = Finite State Machine + A stack.

→ A Pushdown Automata has 3 components :-

1) An input tape
2) A Finite Control Unit
3) A stack with infinite size



→ $P = (Q, \Sigma, \gamma, \delta, q_0, Z_0, F)$

Q = A finite ~~set~~ set of states

Σ = A finite set of input symbols

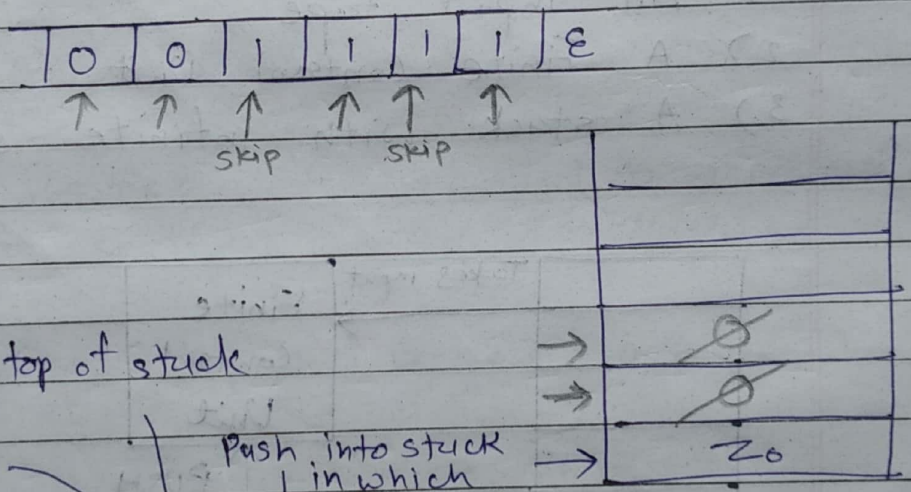
γ = A finite stack Alphabet

δ = The transition function
q_0 = The start state
Z_0 = The start stack symbol.
F = The set of Final / Accepting states

Ex: Design PDA for $L = \{ 0^n 1^{2n}, n>0 \}$

→ Strings accepted : 011, 001111, 000111111, ...

→ logic : for each '0' cut two 1's.



1,0/ε means pop

transitions

state i/p stack	
$\delta(q_0, 0, Z_0) = (q_0, 0Z_0)$	$\delta(q_2, 1, 0) = (q_1, 0)$
$\delta(q_0, 0, 0) = (q_0, 00)$	$\delta(q_2, \epsilon, Z_0) = (q_3, Z_0)$
$\delta(q_0, 1, 0) = (q_1, 0)$	
$\delta(q_1, 1, 0) = (q_2, \epsilon)$	

Convert to PDA :-

(1) $L = \{ a^n b^n \mid n \geq 0 \}$

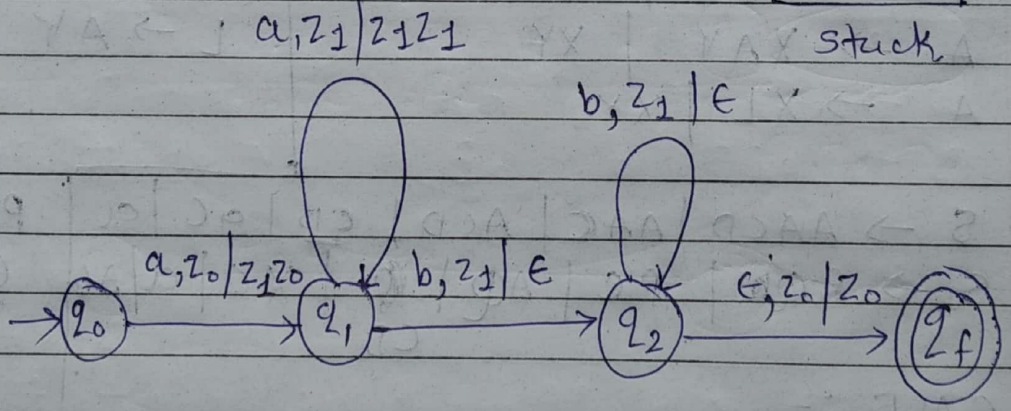
ϵ, ab, aabb, aaabbb, aaaabbbb, ...

a	a	a	a	b	b	b	b	ϵ

logic : for each 'a' pop single 'b'.

Z_1
Z_1
Z_1
Z_1
Z_0

stack



→ Transitions :-

$\delta(q_0, a, Z_0) = (q_1, Z_1 Z_0)$

$\delta(q_1, a, Z_1) = (q_1, Z_1 Z_1)$

POP $\delta(q_1, b, Z_1) = (q_2, \epsilon)$

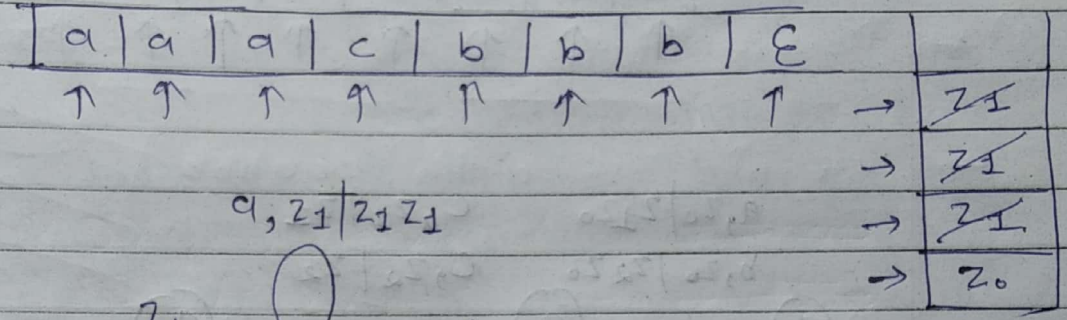
POP $\delta(q_2, b, Z_1) = (q_2, \epsilon)$

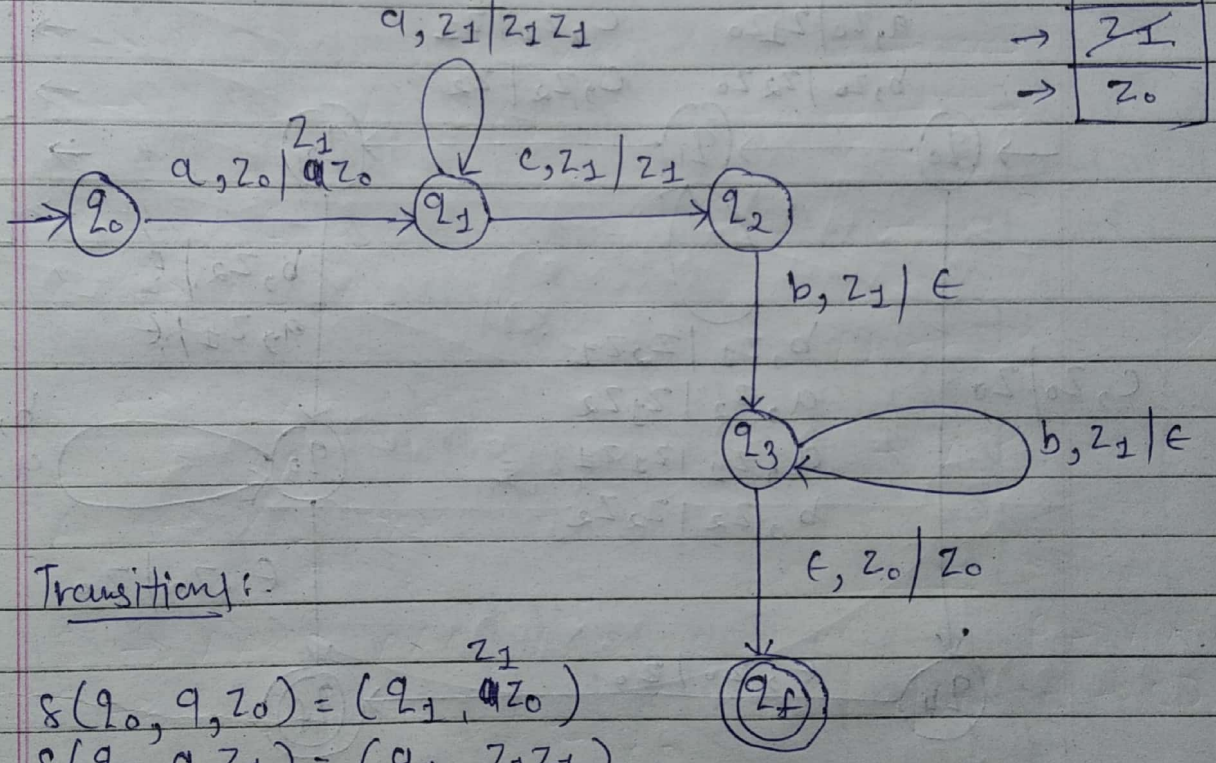
$\delta(q_2, \epsilon, Z_0) = (q_f, Z_0)$

(2) $L = \{ a^n c b^n \mid n \geq 1 \}$

acb, aacbb, aaacbbb, ...

logic: for each 'a' pop 'b' and if 'c' is encountered, skip it.





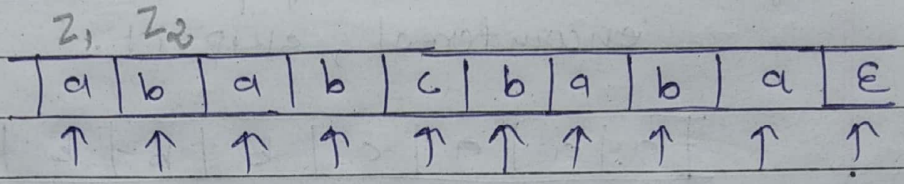
* Transitions:

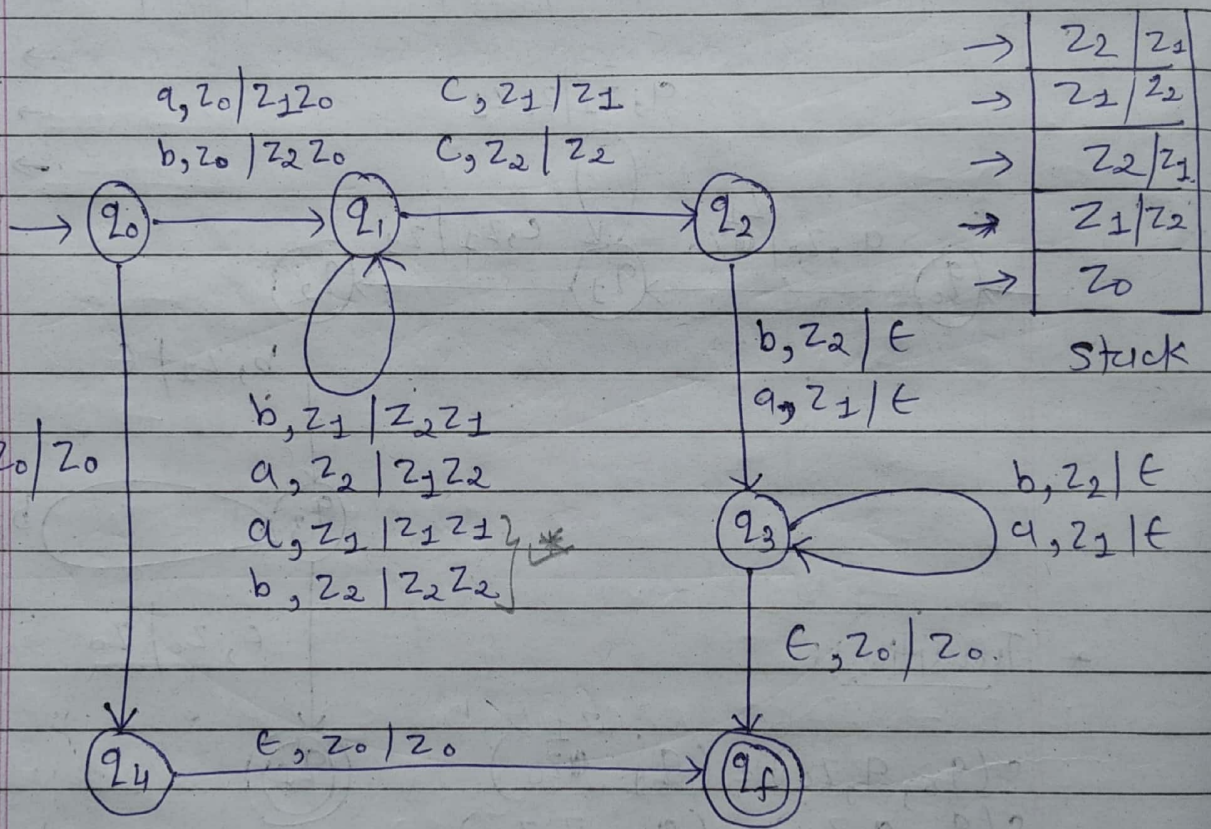
$\delta(q_0, a, Z_0) = (q_1, Z_1 Z_0)$
$\delta(q_1, a, Z_1) = (q_1, Z_1 Z_1)$
$\delta(q_1, c, Z_1) = (q_2, Z_1)$
$\delta(q_2, b, Z_1) = (q_2, \epsilon)$
$\delta(q_2, b, Z_1) = (q_3, \epsilon)$
$\delta(q_3, \epsilon, Z_0) = (q_f, Z_0)$

(3) ** $L = \{ wcw^R \mid w \in \{a,b\}^*, w^R$ is reverse of $w \}$

aacaa , bbcbb , aabbcbbaa, ...

* → Strings :- c, abcba , ababcbaba, ...
bacab , babacabab, ...





(4) ** Construct a PDA for language

$L = \{ 0^n 1^m , n >= 1 , m >= 1 , m > n+2 \}$

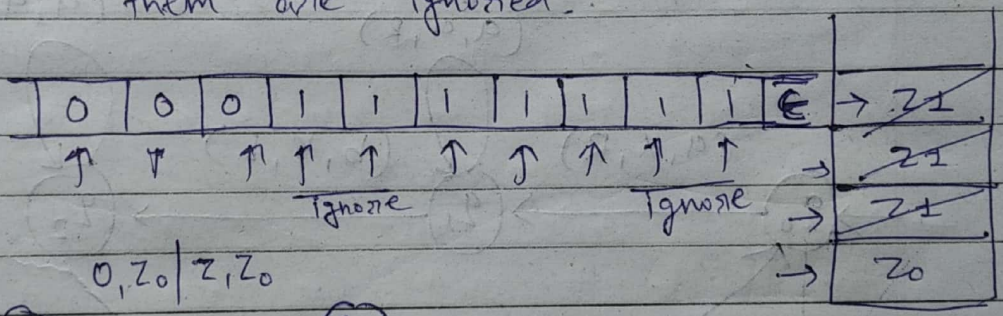
→ Strings : $n = 1, m = 4, 5, \ldots$

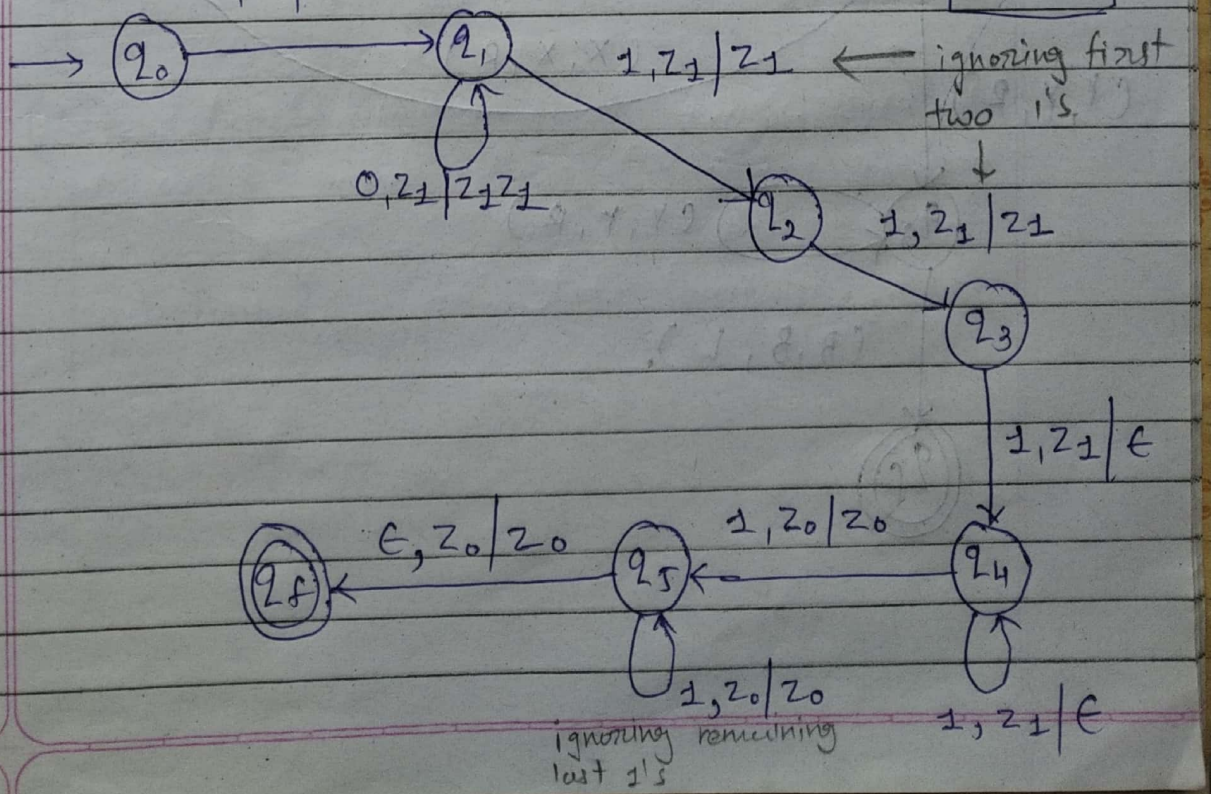
$n = 2, m = 5, 6, \ldots$

$n = 3, m = 6, 7, \ldots$

0 1111 , 00 111111 , 00 1111111 , 000 111 111 , ...

→ Logic :- First 0's are pushed into a stack. When 0's are finished, two 1's are ignored. Thereafter for every '1' as input a '0' is popped out of stack. When stack is empty and still some 1's are left then all of them are ignored.





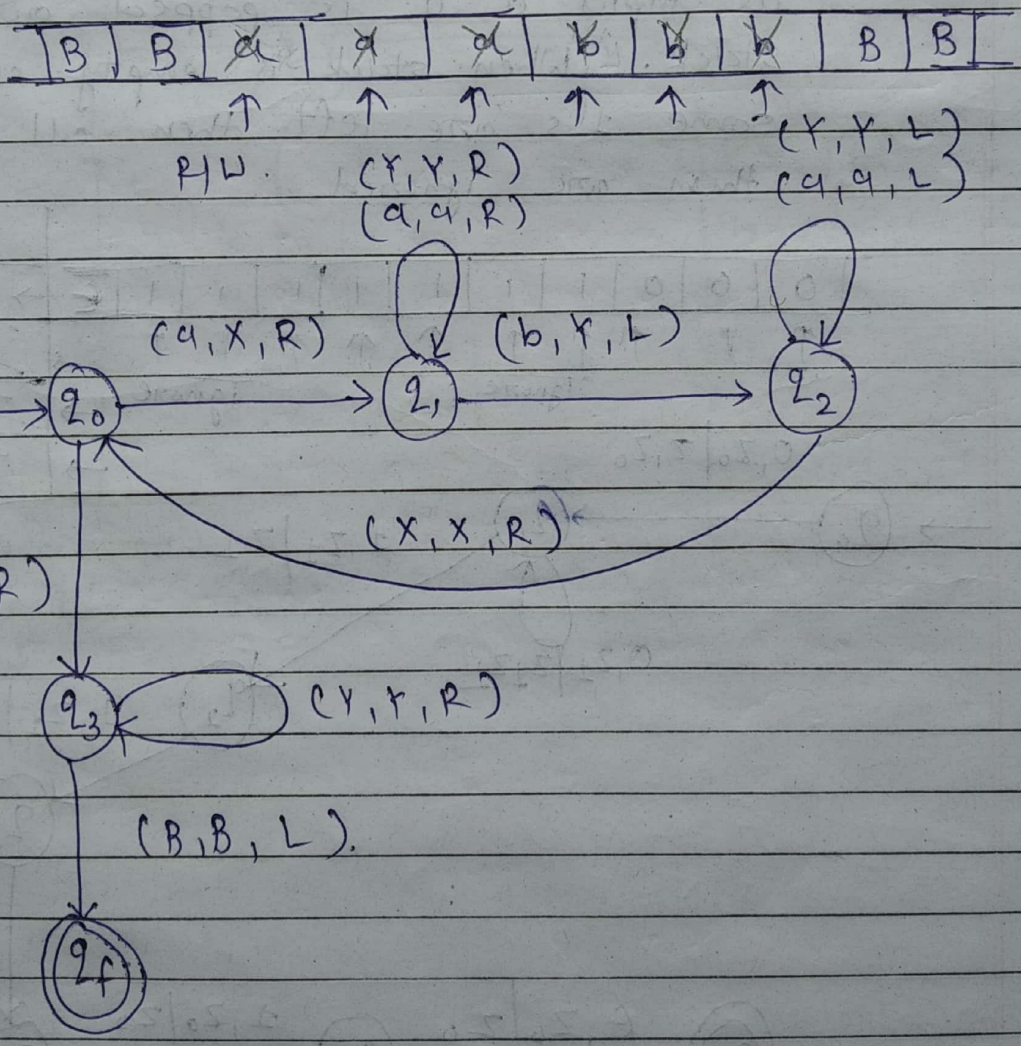
* Turing Machine :-

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

$$\Sigma \subseteq \Gamma$$

(1) ** Design Turing Machine for $L = \{a^n b^n \mid n \geq 1\}$.

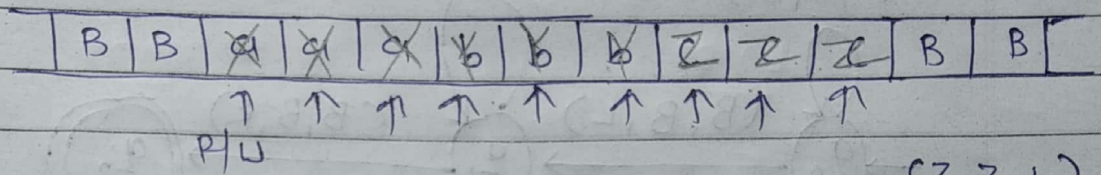
→ String : ab, aabb, aaabbb, ...

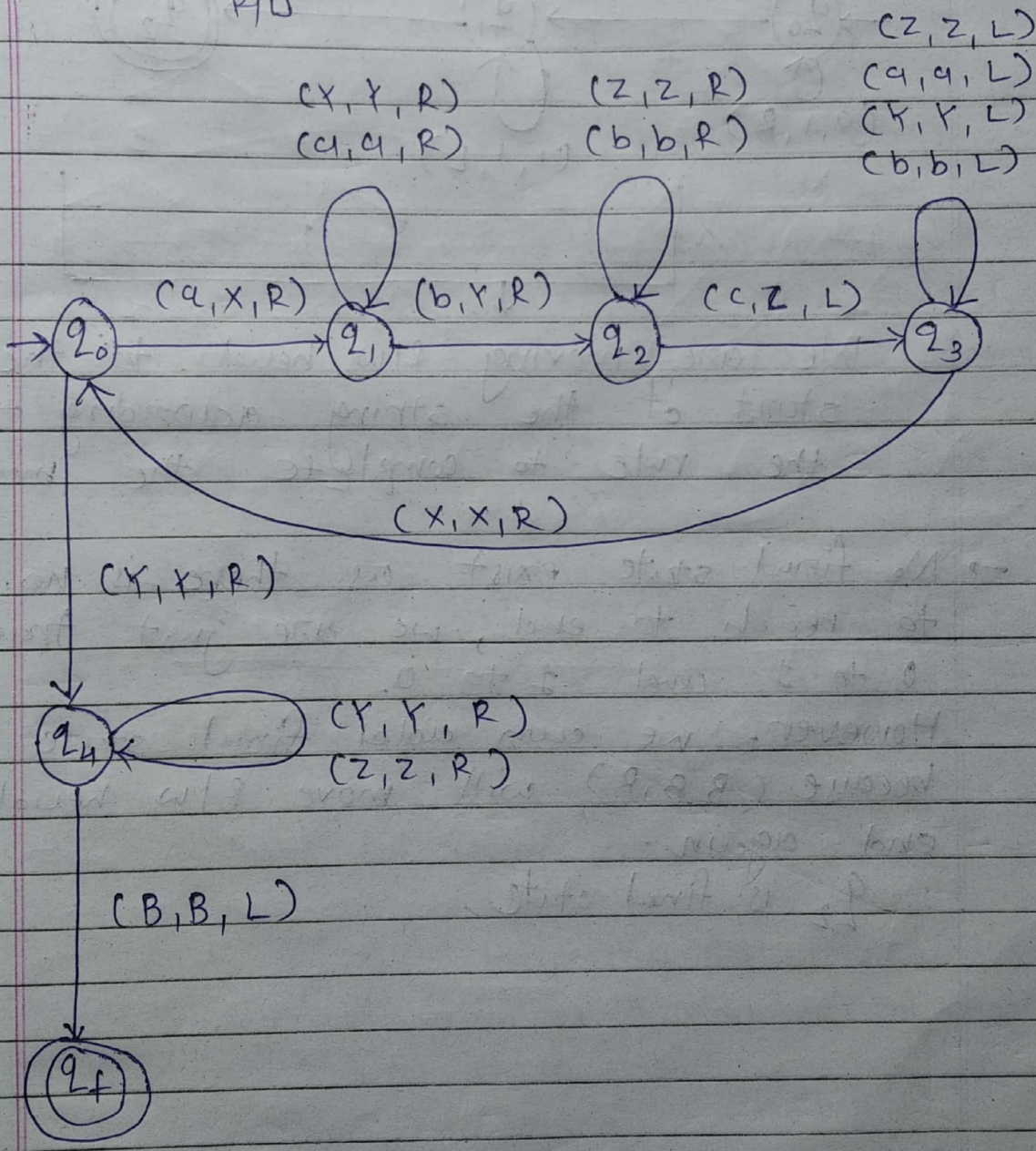


(2) ** Design Turing Machine for language:

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

→ Strings: abc, aabbcc, aaabbbccc, ...

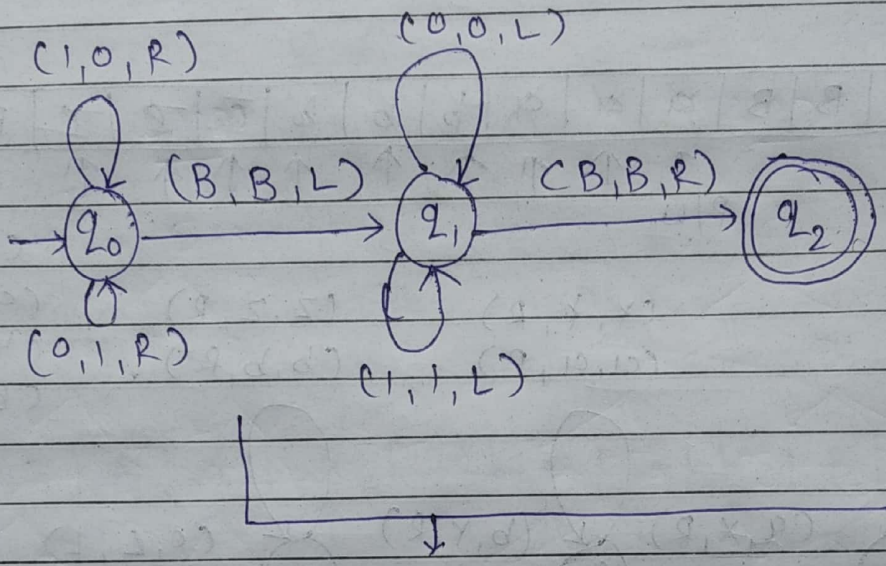




(3) ** Design a Turing Machine for 1's complement.

→ String:- 1 0 1 0 , 1 1 0 0 1 0

1's : 0 1 0 1 1's : 0 0 1 1 0 1



We are moving R/W head to the start of the string according to the rule to complete the machine.

→ No final state exist as there is no need to reach to end, we are just transfering 0 to 1 and 1 to 0.
However, we can add final state here because (B,B,R) will move R/W head to the end again.
∴ q_2 is final state.

(4) ** Design a Turing Machine for 2's complement.

→

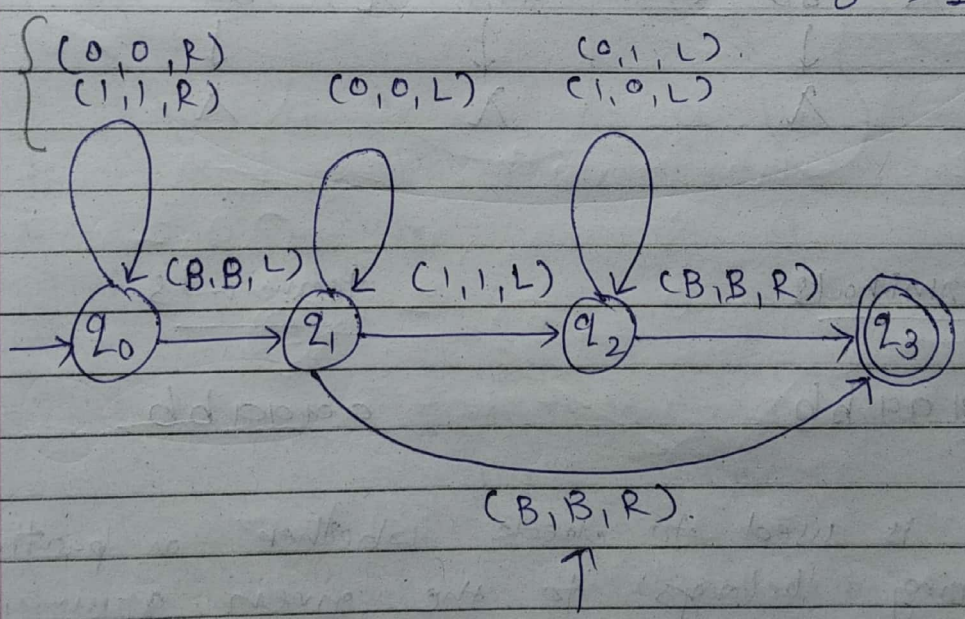
	1	0	1	1	1	1	1	0
1's:	0	1	0	0	0	0	0	1
+1								1
2's:	0	1	0	0	0	0	1	0

→

I/p:	1	0	1	1	1	1	1	0		
o/p:	0	1	0	0	0	0	1	0	B	B
	R/W					↑	↑	↑	↑	

→ logic: Scan from right to left.
Pass all consecutive 0's
Pass first '1', do nothing.
After that, convert 1 → 0 and
0 → 1 for all.

Moving R/W to left

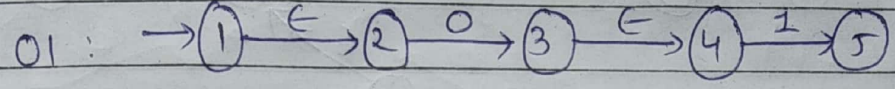


What if only 0's are there as an input? ⇒ Blank will be encountered then move right.

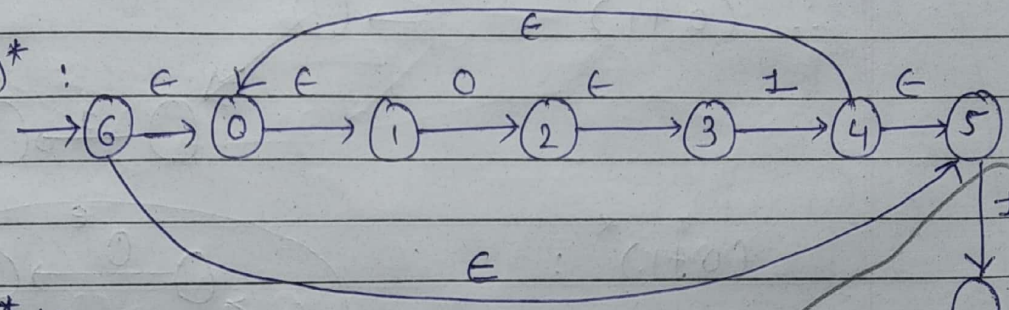
* Construct ε-NFA | Kleen's Theorem :-

(1) ** $((01)^*10 + 00)^*$

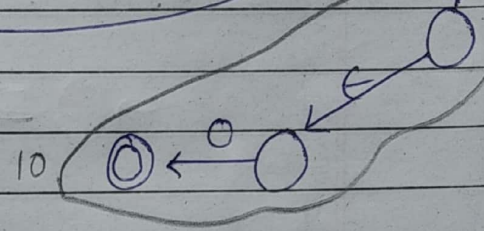
01 :



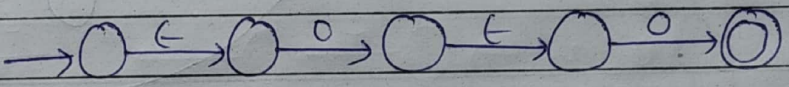
$(01)^*$:



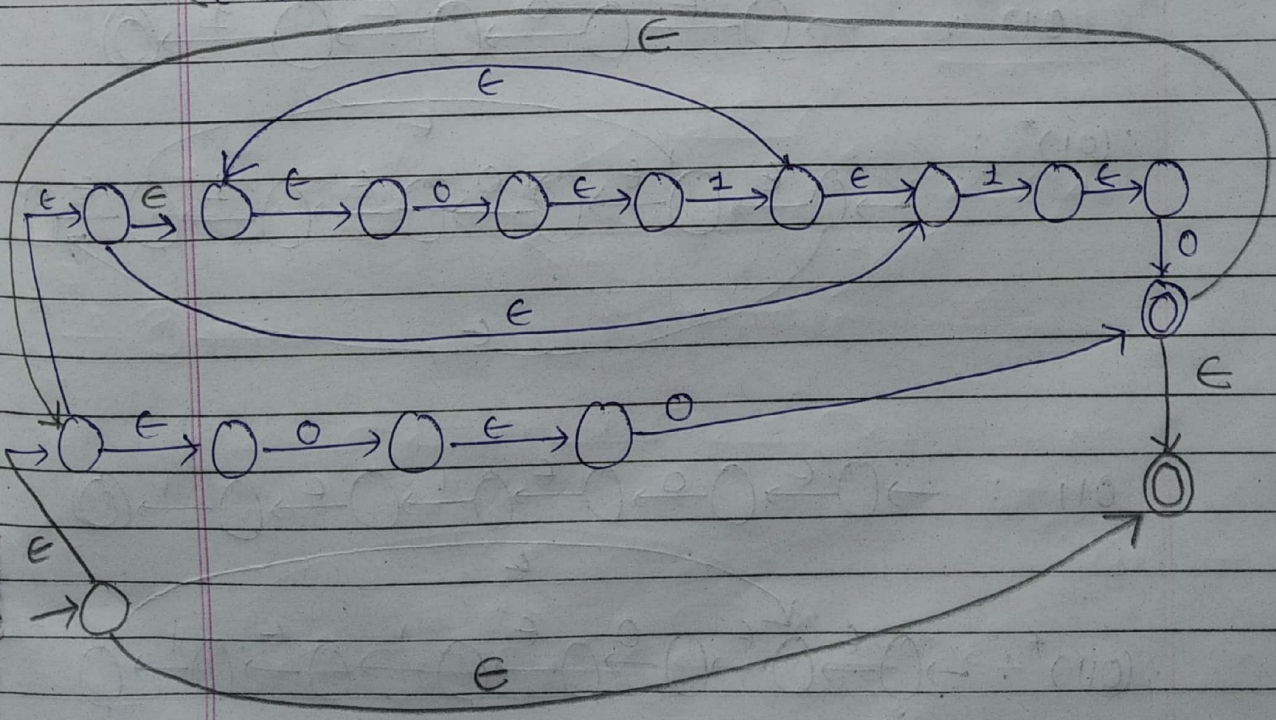
$(01)^*10$



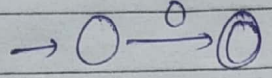
00 :

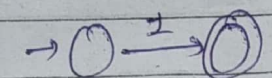


$(01)^*10 + 00)^*$

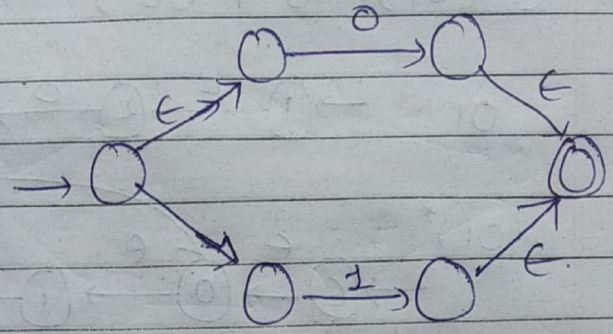


(2) ⇒ $(0+1)^* (01)^* (011)^*$

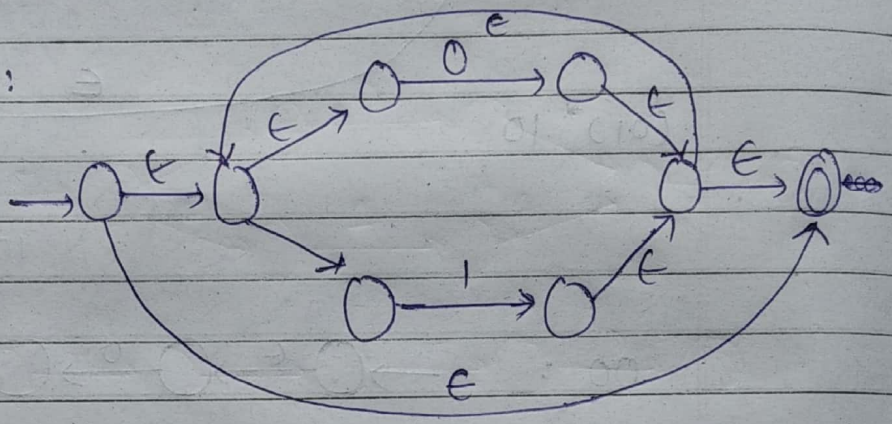




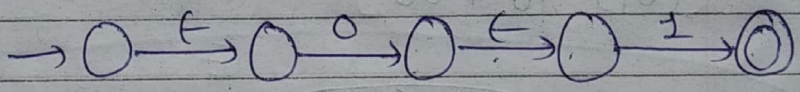
$\therefore (0+1)$:



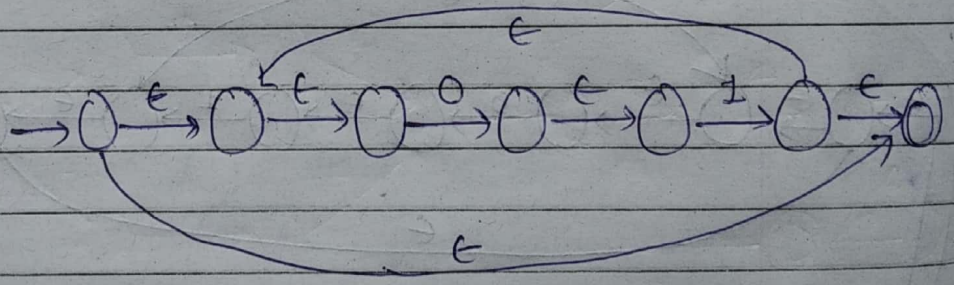
$(0+1)^*$:



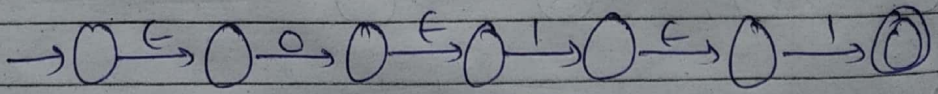
01 :



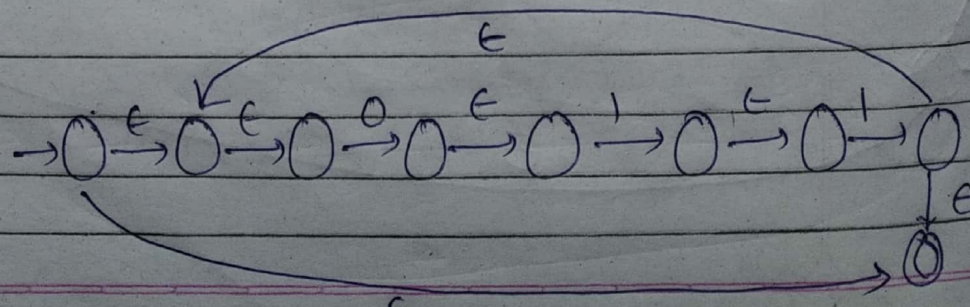
$\therefore (01)^*$:



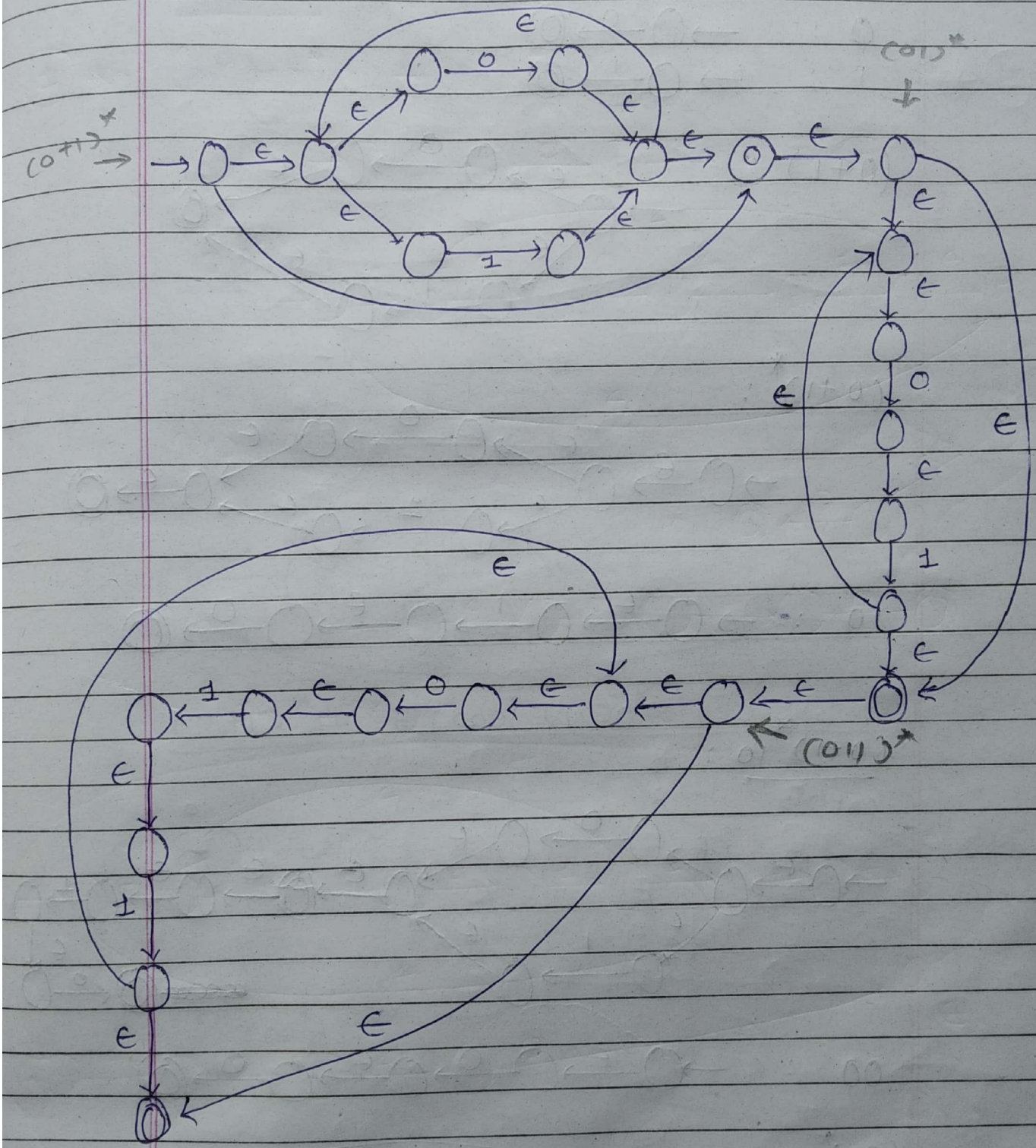
011 :



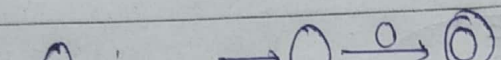
$\therefore (011)^*$:

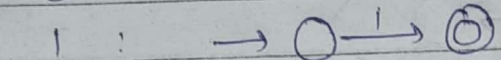


$(0+1)^*(01)^*(011)^*$

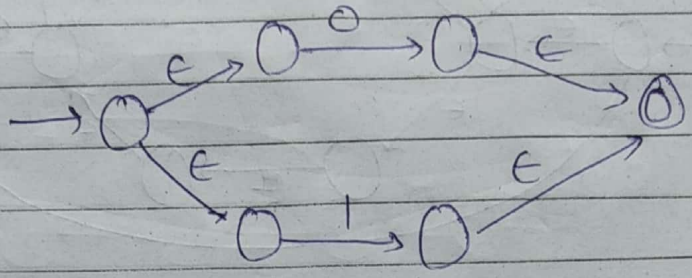


(3) * ((0+1)* 10 + (00)* (11)*)*

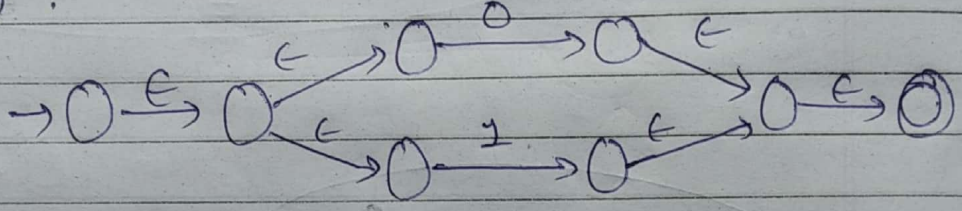
0 : 

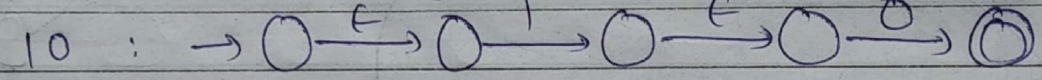
1 : 

(0+1) :



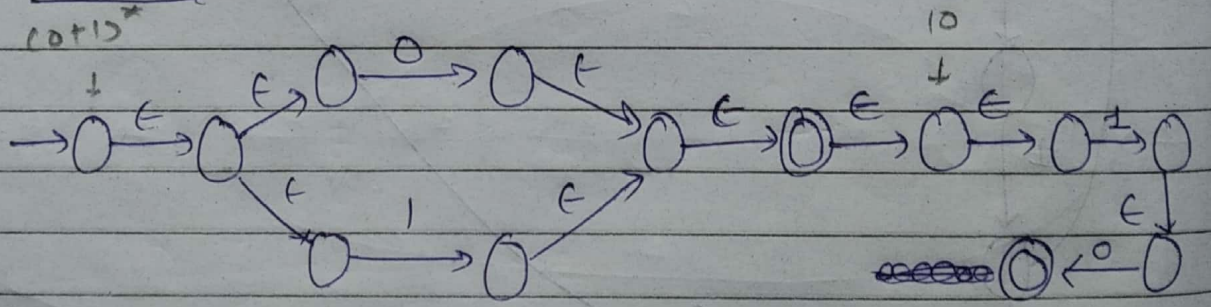
(0+1)* :

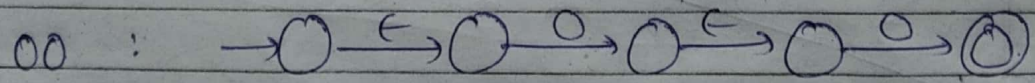


10 : 

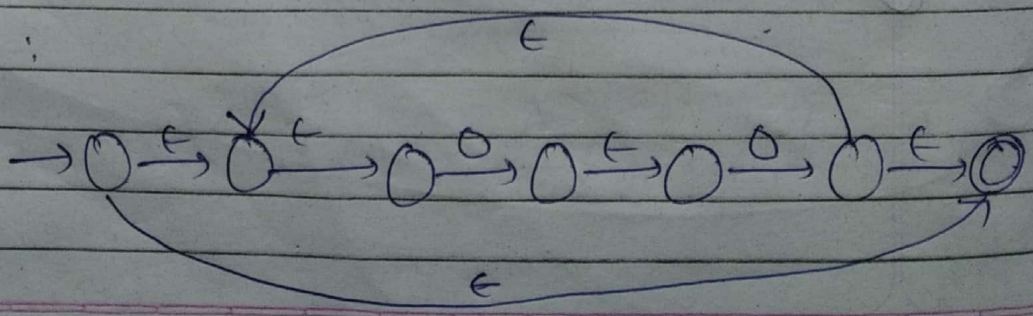
(0+1)* 10 :

(0+1)*

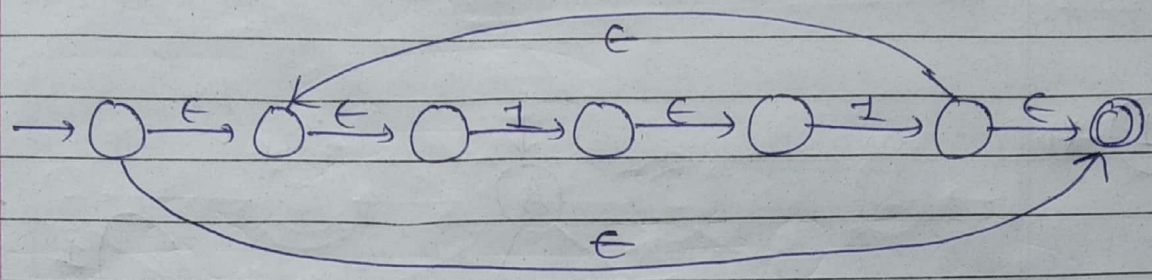


00 : 

(00)* :

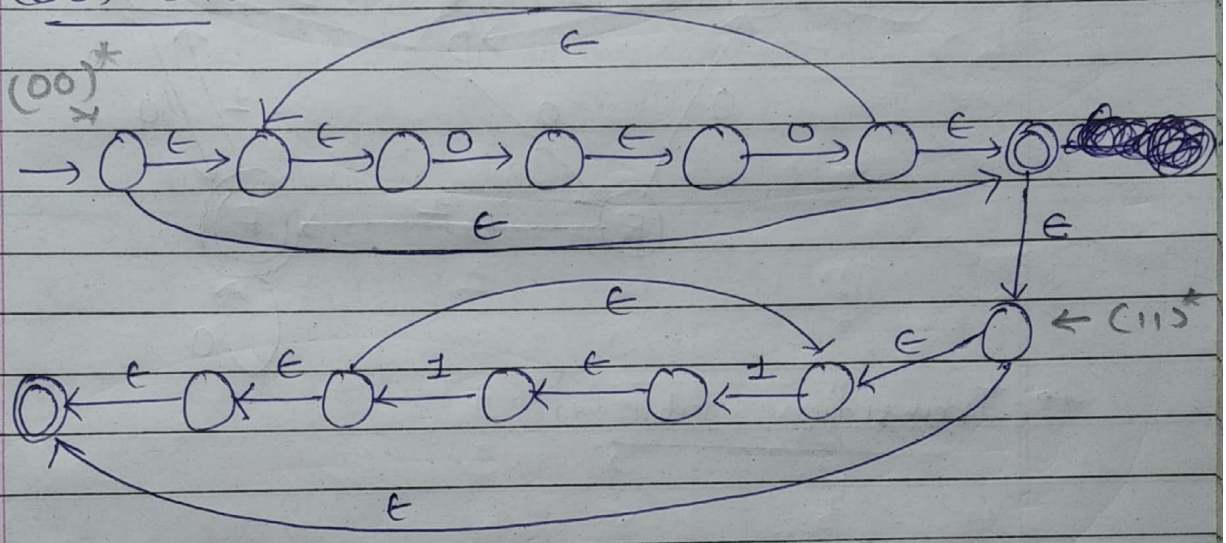


Similarly $(11)^*$;

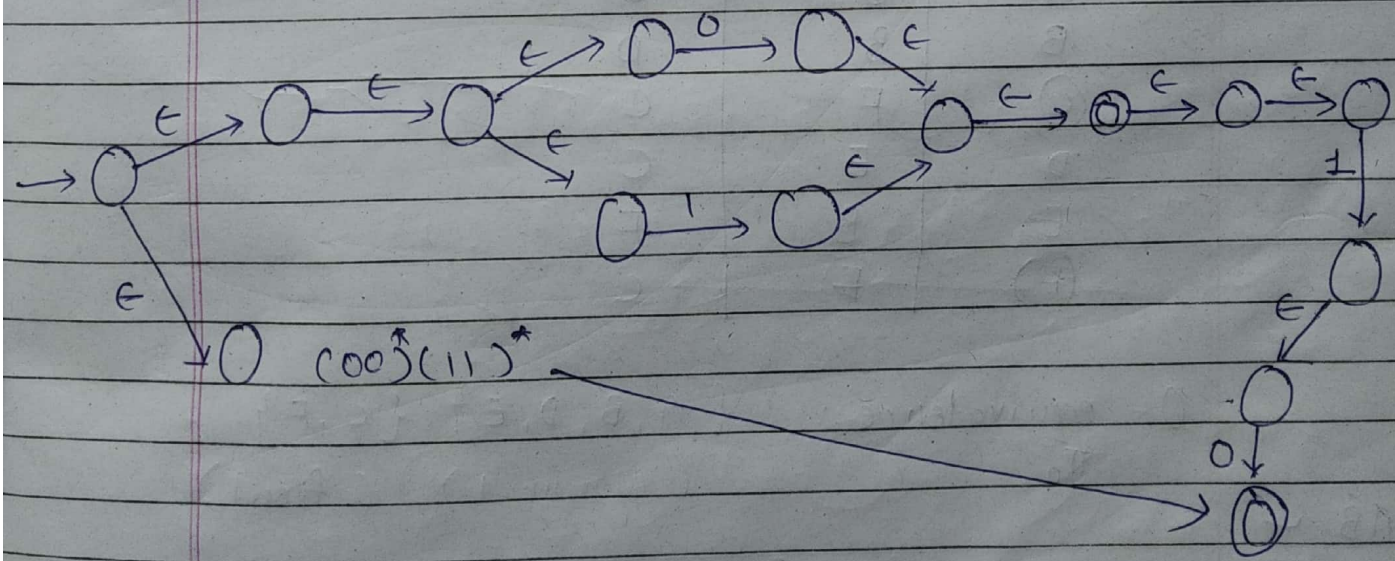


$(00)^*(11)^*$

$(00)^*$

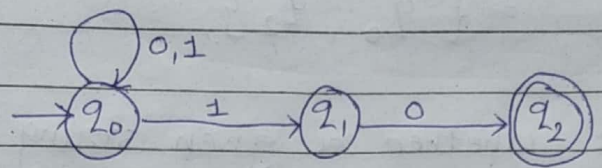


$(0+1)^*10 + (00)^*(11)^*$



* Extended Transition Function for NFA:

(1) ** NFA accepting strings end with 10.
Find $\delta^*(q_0, 1010)$.



δ	0	1
$\rightarrow q_0$	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_2\}$	$\emptyset$
(q_2)	$\emptyset$	$\emptyset$

* Base : $\delta^*(q_0, \varepsilon) = \{q_0\}$
Induction : $w = xa$
$\delta^*(q_0, w) = \delta(\delta^*(q_0, x), a)$

$w = 1010$
$x = 101$, $a = 0$

start →

Soln: $w = 1010$

$\delta^*(q_0, \varepsilon) = \{q_0\}$

1010 $\delta^*(q_0, 1) = \delta(\delta^*(q_0, \varepsilon), 1)$
$= \delta(\{q_0\}, 1)$
1 ε $= \{q_0, q_1\}$ (∵ from transition table.)

1010 $\delta^*(q_0, 10) = \delta(\delta^*(q_0, 1), 0)$
$= \delta(\{q_0, q_1\}, 0)$
$= \delta(q_0, 0) \cup \delta(q_1, 0)$
$= \{q_0\} \cup \{q_2\}$
$= \{q_0, q_2\}$

1010 $\delta^*(q_0, 101) = \delta(\delta^*(q_0, 10), 1)$
$= \delta(\{q_0, q_2\}, 1)$
$= \delta(\{q_0\}, 1) \cup \delta(\{q_2\}, 1)$
$= \{q_0, q_1\} \cup \{\}$
$= \{q_0, q_1\}$

1010 $\delta^*(\{q_0, 1010) = \delta(\delta^*(q_0, 101), 0)$

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

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

$= \{q_0\} \cup \{q_2\}$

$= \{q_0, q_2\}$

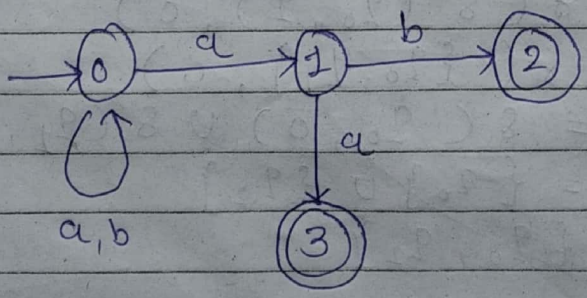
→ Now check whether a given string is accepted by the NFA or not.

If result $\{q_0, q_2\}$ has atleast one final state ⇒ string is accepted.
otherwise not acceptable.

$\therefore \underbrace{\delta^*(q_0, 1010)}_{\text{result } \delta^*} \cap \underbrace{\{q_2\}}_{\text{final}} = \{q_0, q_2\} \cap \{q_2\} = \{q_2\}.$

Here, we are getting q_2 which is a final state. ∴ Given string is accepted.

(2) ** Compute $\delta^*(0, abab)$ for given NFA.



δ	a	b
0	{0,1}	{0}
1	{3}	{2}
(2)	∅	∅
(3)	∅	∅

ε~~abab~~ → Base: $\delta^*(0, \varepsilon) = \{0\}$.

abab → $\delta^*(0, a) = \delta(\delta^*(0, \varepsilon), a)$

$\varepsilon \wedge a = \delta(\{0\}, a) = \{0, 1\}$

abab $\delta^*(0, ab) = \delta(\delta^*(0,a), b)$

$= \delta(\{0,1\}, b)$

$= \delta(\{0\}, b) \cup \delta(\{1\}, b)$

$= \{0\} \cup \{2\}$

$= \{0,2\}$.

abab $\delta^*(0, aba) = \delta(\delta^*(0, ab), a)$.

$= \delta(\{0,2\}, a)$

$= \delta(\{0\}, a) \cup \delta(\{2\}, a)$

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

$= \{0,1\}$.

abab $\delta^*(0, abab) = \delta(\delta^*(0, aba), b)$

$= \delta(\{0,1\}, b)$

$= \delta\{\{0\}, b) \cup \delta\{\{1\}, b\}$

$= \{0\} \cup \{2\}$

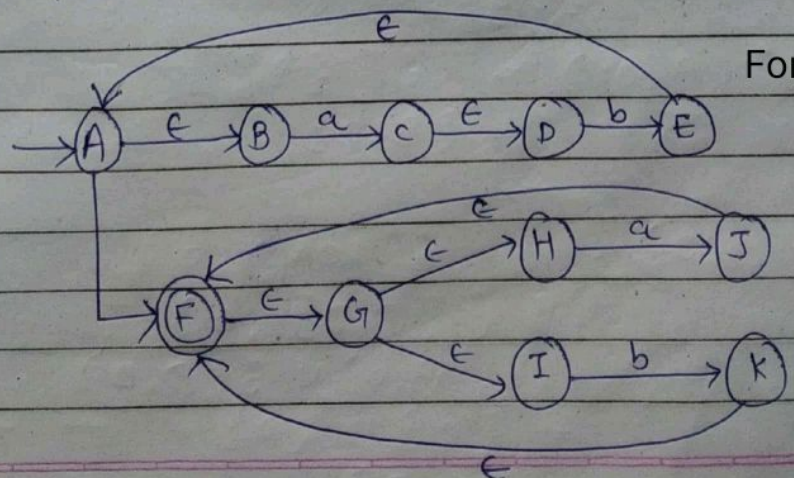
$= \{0,2\}$.

$\therefore \delta^*(0, abab) = \{0,2\}$.

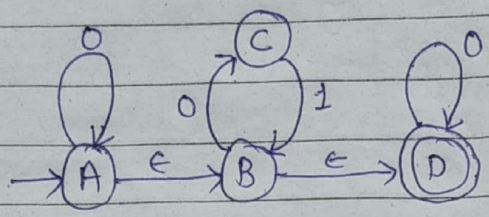
abab is accepted as $\{2\}$ is final state which exists in the $\delta^*(0, abab)$.

(3) ** Compute $\delta^*(A, aba)$ for given ϵ-NFA.

For Practice Purpose



(4) * Find $\delta^*(A, 010)$ for given ϵ-NFA.



δ	ϵ	0	1
A	A, B, D	A	$\emptyset$
B	B, D	C	$\emptyset$
C	C	$\emptyset$	B
D	D	D	$\emptyset$

Soln:

$\delta^*(A, \epsilon) = \epsilon(\{A\})$ *

$= \{A, B, D\}$

010 $\delta^*(A, 0) = \epsilon(\delta(\delta^*(A, \epsilon), 0))$

$\epsilon \wedge 0 = \epsilon[\delta(\{A, B, D\}, 0)]$

$= \epsilon[\delta(A, 0) \cup \delta(B, 0) \cup \delta(D, 0)]$

$= \epsilon[\{A, C, D\}]$

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

010 $\delta^*(A, 01) = \delta(\delta^*(A, 0), 1)$

$= \delta(\{A, B, C, D\}, 1)$

$= \delta(A, 1) \cup \delta(B, 1) \cup \delta(C, 1) \cup \delta(D, 1)$

$= \emptyset \cup \emptyset \cup \{B\} \cup \emptyset$

$= \{B\}$

$\therefore \delta^*(A, 01) = \epsilon[\{B\}]$

$= \{B, D\}$

010 $\delta^*(A, 010) = \delta(\delta^*(A, 01), 0)$

$= \delta(\{B, D\}, 0)$

$= \delta(B, 0) \cup \delta(D, 0)$

$= \{C\} \cup \{D\}$

$= \{C, D\}$

Date
Page

[redacted]

$\therefore \delta^*(A, 010) = \in[\{C, D\}]$

$= \{C, D\}$

$\therefore$ Answer : $\{C, D\}$

* Derivation Trees :-

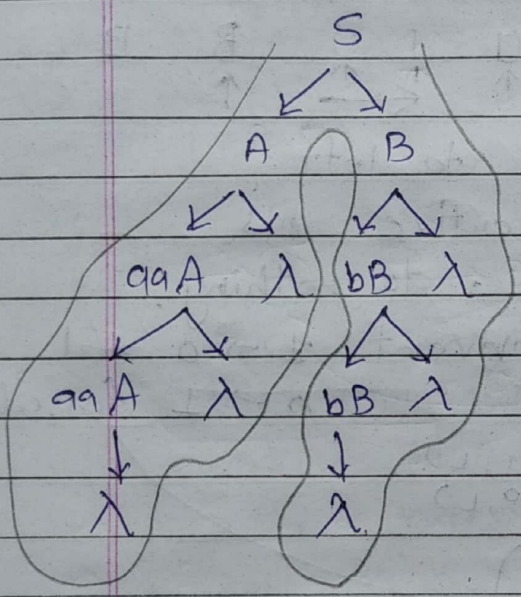
(1) $S \rightarrow AB$

$A \rightarrow aaA \mid \lambda$

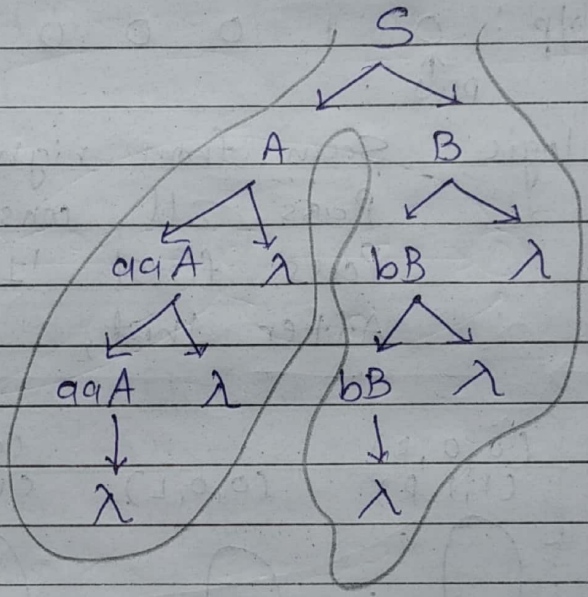
$B \rightarrow bB \mid \lambda$

λ = null or $\in$

LMDT



RMDT



terminals

aa aa bb

terminals

aaaa bb

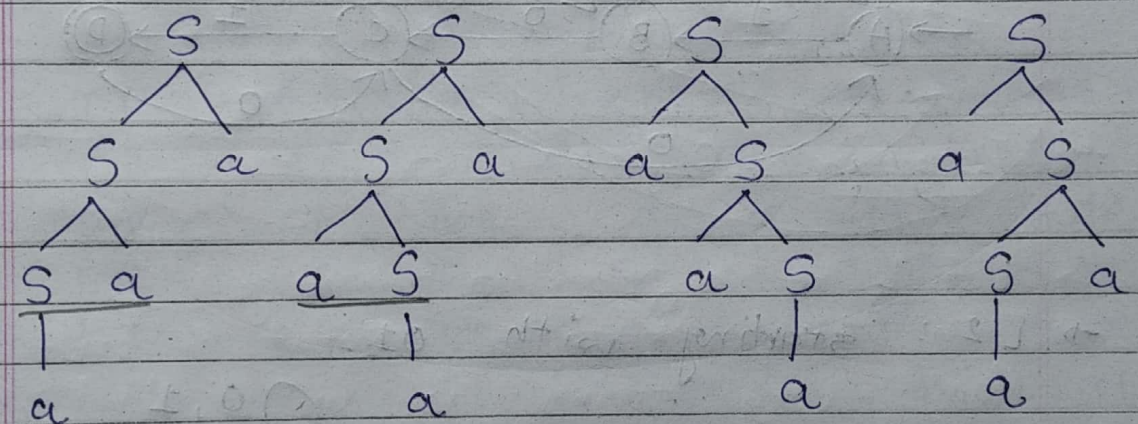
→ It is used to check whether a particular string belongs to the given grammar or not.

* Ambiguous & Unambiguous Grammar :-

→ More than one derivation tree for any word ⇒ Ambiguous grammar

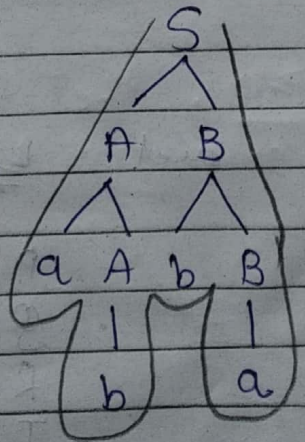
Exactly one derivation tree for any word ⇒ Unambiguous grammar

(1) →* $S \to Sa \mid aS \mid a$



→ More than one derivation tree exists.
∴ Given grammar is ambiguous.

(2) →* $S \to AB$
$A \to aA \mid b$
$B \to bB \mid a$



⇒ Here only one derivation tree is possible for any word (abab).

→ We can not derive abab by another derivation tree.
∴ It is unambiguous grammar.

* Pumping lemma for Regular language :-

→ If language satisfies the property of lemma
⇓
May or may not be regular

→ If language fails to satisfy the property of lemma
⇓
Not a regular language

** Theorem :-

→ Let L be a regular language
→ Then there exists a constant n such that for every string w in L, $|w| \geq n$. (length)
→ We can break w into 3 strings, $w = xyz$ such that

(i) $y \neq \varepsilon$ or $|y| \neq 0$
(ii) $|xy| \leq n$
(iii) For all $k \geq 0$, the string xy^kz is also in L.

* Pumping lemma for Context Free languages:-

→ Let L be a CFL.

→ Let n be any constant

→ Any string z in L, $|z| \geq n$

Split $z = uvwxy$ such that

(i) $|vwx| \leq n$

(ii) $vx \neq \epsilon$ or $|vx| \geq 1$

(iii) For all $i \geq 0$ $uv^iwx^iy \in L$.

⇓ if not then language is not context free language.

① ** $L = \{a^n b^n c^n \mid n \geq 1\}$.

→ $z = a^n b^n c^n$, $n \geq 1$

→ Split $z = uvwxy$

where $|vwx| \leq n$.

$\therefore u = a^n$, $vwx = b^n$ ($|vwx| \leq n$), $y = c^n$

length of vx →

→ $|vx| \geq 1$ $\therefore vx = b^{n-m}$, $m < n$

→ $uv^iwx^iy = uv^{i-1}.vwx.x^{i-1}.y$ ($\because$ to put the value of $v x^{i-1}$)

$= uv^{i-1}wx^{i-1}y(vx)$

$= uvwxy(vx)^{i-1}$

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

$= a^n b^n c^n (b^{ni-n-mi+m})$

$= a^n b^{n+ni-n-mi+m} c^n$

$= a^n b^{ni-mi+m} c^n$

$\therefore uv^i wx^i y = a^n c^n b^{ni-mi+m}$

→ For $i=0$,

$uv^i wx^i y = a^n c^n b^m \notin L$

which does not belongs to the given language because for L, $n \geq 1$ but $m < n$.

∴ Given language is not a context free language.

— By Rushik Rathod.