Finite State Machine (FSM)

- F.A. with Output
  - Moore Machine
  - Mealy Machine
- F.A. without Output
  - DFA (Deterministic Finite Automata)
  - NFA (Non-deterministic Finite Automata)
  - ϵ-NFA

DFA

DFA are of 5 tuples $(Q, \Sigma, q_0, \delta, F)$

Where Q = Set of all states

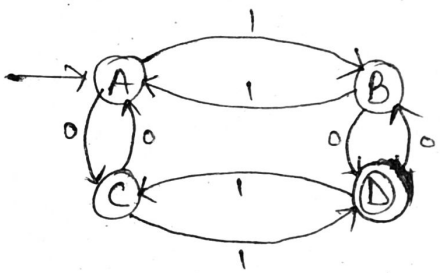
Σ = set of inputs

q_0 = Initial state $(q_0 \in Q)$

F = set of final states $(F \subseteq Q)$

δ = transition function $(Q \times \Sigma \to Q)$

E.g.



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

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

$q_0 = A$

$F = \{D\}$

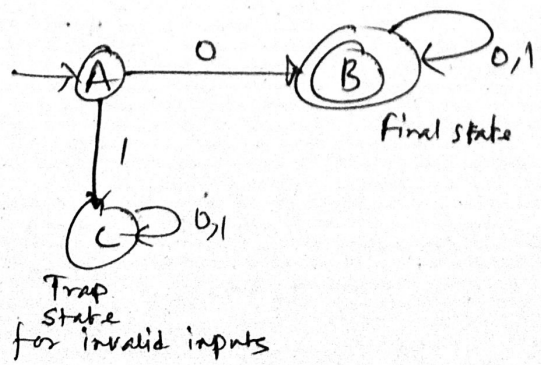
States	Inputs 0	1
A	C	B
B	D	A
C	A	D
D	B	C

Transition function (δ)

Ex-1) L1 = Set of all string starts with '0'. $\Sigma = \{0, 1\}$

$= \{0, 00, 000, \ldots 01, 010, 011, \ldots\}$

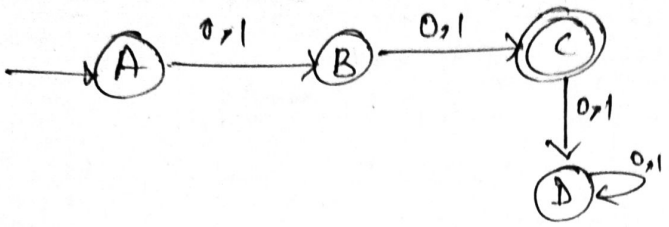
DFA



Ex-2) DFA that accepts sets of all string over {0,1} of length 2.

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

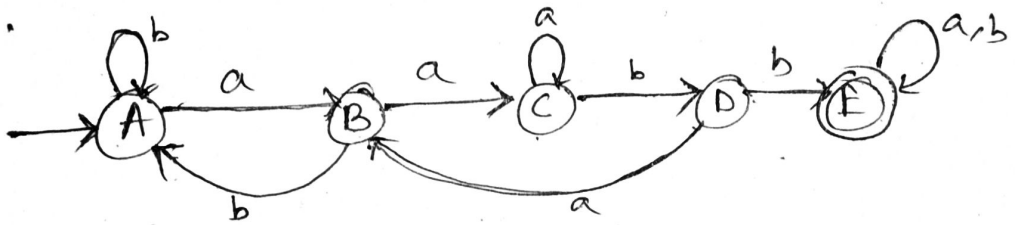
$L = \{00, 01, 10, 11\}$



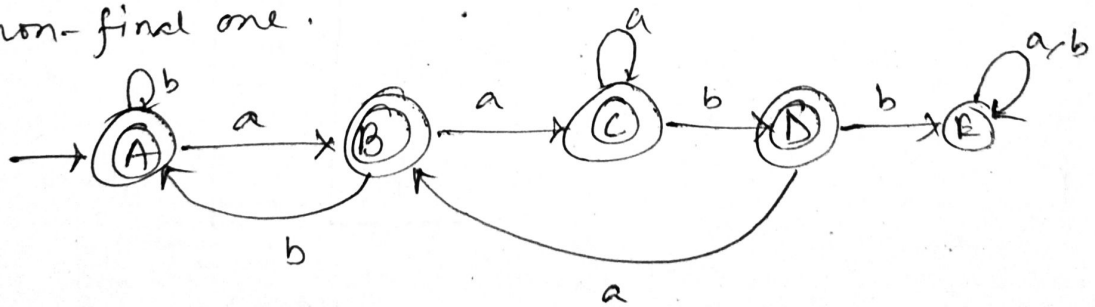
Ex-3) DFA that accepts any strings over (a,b) that does not contain the string "aabb" in it.

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

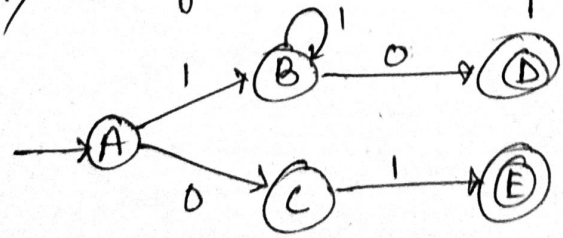
Let us construct a DFA that accepts any string over (a,b) that must contains the string "aabb" in it.



Then make non-finals as final and the final to the non-final one.



Ex-4) Recognise and complete the DFA.



Here, A reaches B if 1 is given and reaches D if given 0 while B can reach itself if 1 is given.

So, In this hand, $L_1 = \{10, 110, 1110, 11110, \ldots\}$

∴ L_1 = {Acceps a string of atleast one '1' followed by a '0'}

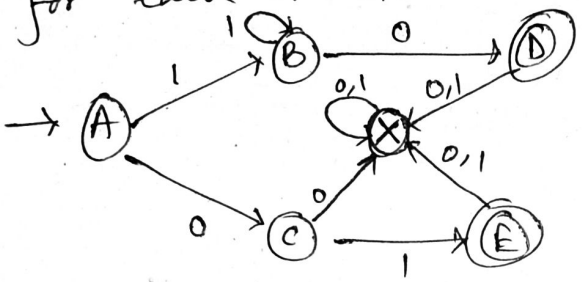
On the other hand, A reaches C is '0' is given and ~~B~~ then E if 1 is given.

∴ $L_2 = \{01\}$

∴ L_2 = { Accepts the string 01 }

∴ $L = L_1 \cup L_2$ = {Accepts the string 01 or a string of atleast one '1' followed by a '0'}

Now, completing the DFA by filling up the missing inputs for each states.



REGULAR LANGUAGES ⟩ A language is said to be a Regular language if and only if some finite State Machines recognizes it.

So, the languages that are not recognized by the FSM and requires memory, are not REGULAR LANGUAGES.

Because FSM has a limited memory and it cannot store or count strings.

Operations on Regular languages

UNION — $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$

CONCATENATION — $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$

STAR — $A^* = \{x_1 x_2 x_3 \cdots x_k \mid k \geq 0 \text{ and each } x \in A\}$

E.g. $A = \{pq, r\}$, $B = \{t, uv\}$

∴ $A \cup B = \{pq, r, t, uv\}$

∴ $A \circ B = \{pqt, pquv, rt, ruv\}$

$A^* = \{\epsilon, pq, r, pqr, rpq, pqpq, rr, pqpqr, rrpq, \ldots\infty\ldots\}$

NFA > What is Determinism >

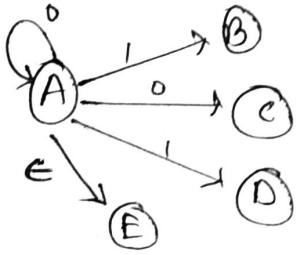
→ In DFA, given the current state, we know what the next state will be.

→ It has only one unique next state.
→ It has no choice or randomness
→ It is simple and easy to design.

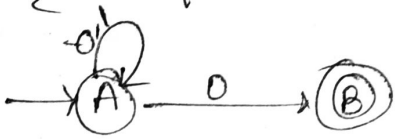
But in NFA >

→ Given the current state there could be multiple next states.
→ The next state may be chosen at random
→ All the nexts states may be chosen in parallel.

E.g.



E.g.1> L = { set of all strings that end with '0' }



Here we can see that A reaches itself as well as B if the input is '0'.

NFA are of 5 tuples $(Q, \Sigma, q_0, F, \delta)$

Where Q = Set of all states. [ϕ = no where]
Σ = Set of inputs
q_0 = initial state
F = final states' set.

In the example, $Q = \{A, B\}$, $\Sigma = \{0, 1\}$, $q_0 = A$, $F = \{B\}$

Here, $A \xrightarrow{0} A, B, AB, \phi$ [as next state can be chosen parallelly]

If we have suppose three states A, B, C then.

$A \xrightarrow{\text{input}} A, B, C, AB, BC, AC, ABC, \phi$

There are 8 possibilities for three states. $(2^3 = 8)$

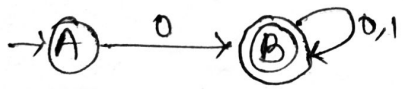
So, the transition function for NFA is

$$\delta \Rightarrow Q \times \Sigma \longrightarrow 2^Q .$$

If there is any way to run the machine that ends in any set of states out of which atleast one state is a final state, then the NFA accepts.

E.g.2) L = {set of all string that starts with '0'}

= {0, 00, 000, 0000, ... , 01, 010, 011, 001, ...

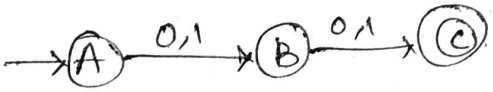


[empty inputs doesn't matter]

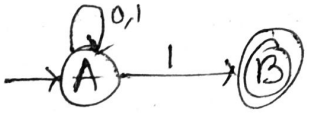
Here, the input seqn "101" is called as dead configuration as 

E.g.3) NFA that accepts all strings over {0,1} of length 2.

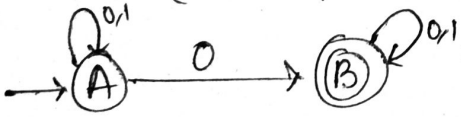
Here, $\Sigma = \{0,1\}$, $L = \{00, 01, 10, 11\}$



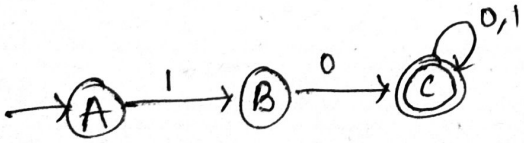
Ex-1) L = {set of all strings that ends with '1'}



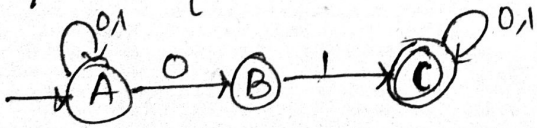
Ex-2) L = {set of all strings that contain '0'}



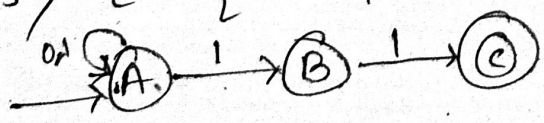
Ex-3) L = {set of all strings that starts with '10'}



Ex-4) L = {Set of all strings that contain '01'}



Ex-5) L = {set of all strings that ends with '11'}



NFA to DFA conversion

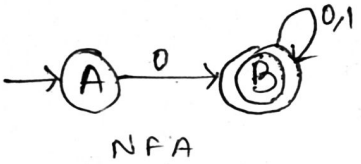
Every DFA is an NFA, but not vice versa. But there is an equivalent DFA for every NFA.

$$\delta_{DFA} = Q \times \Sigma \rightarrow Q$$

$$\delta_{NFA} = Q \times \Sigma \rightarrow 2^Q$$

E.g.) L = { set of all strings over {0,1} that starts with '0'}

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



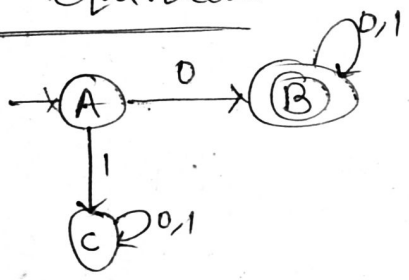
NFA

Transition Diagram

	0	1
A	B	ϕ
B	B	B

⇓

DFA equivalent



Converted T.D. for DFA

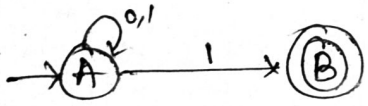
	0	1
A	B	C
B	B	B
C	C	C

[C - is the Dead state]

E.g.2) L = { set of all strings over (0,1) that ends with '1'}

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

NFA

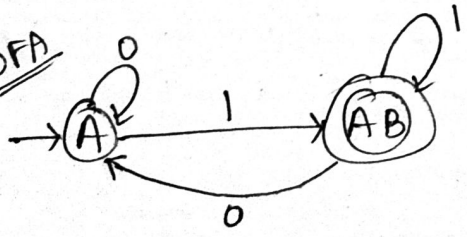


Transition Table

	0	1
A	A	{A, B}
B	ϕ	ϕ

⇓

DFA



Converted T.T. for DFA

	0	1
A	A	AB
AB	A	AB

[AB is a single state]

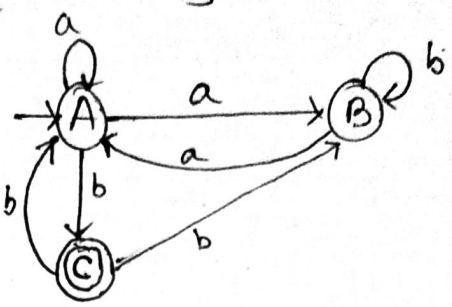
{A, B} ∪ ϕ

(A ∪ B) from Previous table (A ∪ ϕ)

E.g.3) Find the equivalent DFA for the NFA given by $M = \{ \{A,B,C\}, \{a,b\}, \delta, A, \{C\} \}$ where δ is given by →

	a	b
→A	A/B	C
B	A	B
(C)	—	A/B

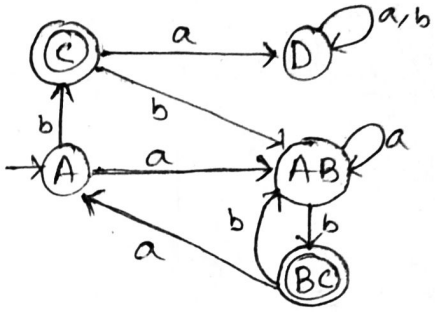
⇒



Converted Transition Table for DFA

	a	b
→A	AB	C
AB	AB	BC
(BC)	A	AB
(C)	D	AB
D	D	D

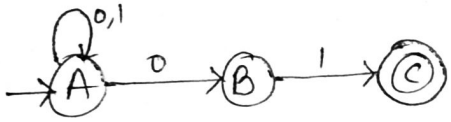
Transition Diagram



E.g.4) L = { Set of all string over (0,1) that ends with '01' }

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

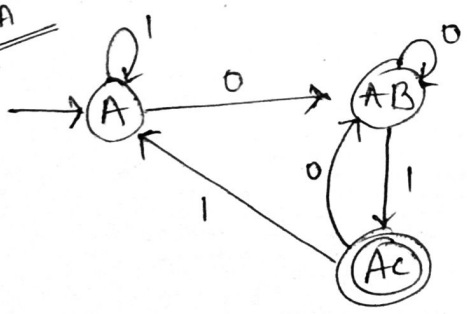
NFA



Transition Table

	0	1
→A	A,B	A
B	ϕ	C
(C)	ϕ	ϕ

DFA



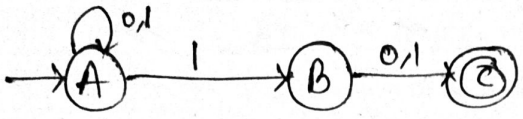
Converted T.T for DFA

	0	1
→A	AB	A
AB	AB	AC
(AC)	AB	A

E.g. 6) Design an NFA for a language that accepts all strings over (0,1), in which the second last is always '1'. Then convert it to its equivalent DFA.

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

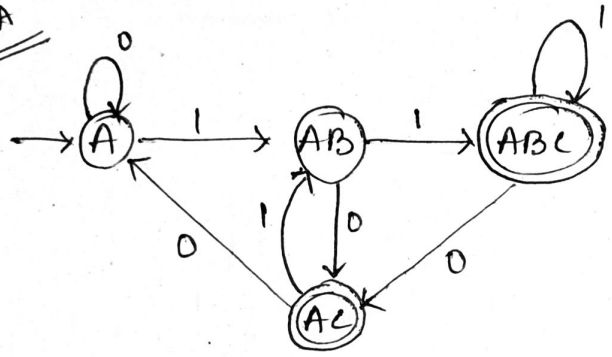
NFA



Transition Table

	0	1
A	A	A,B
B	C	C
(C)	ϕ	ϕ

DFA



converted T.T. for DFA

	0	1
→A	A	AB
AB	AC	ABC
(AC)	A	AB
(ABC)	AC	ABC

Minimization of DFA → Minimization of DFA is required to obtain the minimal version of any DFA which consists of the minimum no. of states possible. Basically we've to merge the equivalent states together.

Two states 'A' and 'B' are said to be equivalent if,

$\delta(A,X) \to F$ or $\delta(A,X) \not\to F$

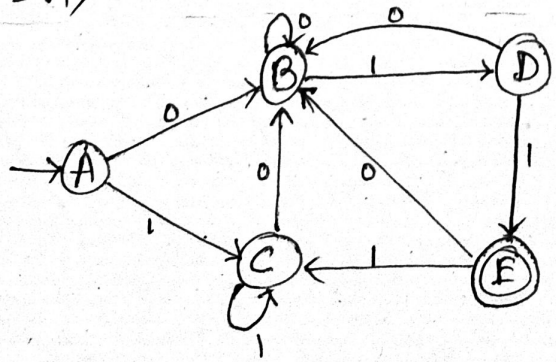
$\delta(B,X) \to F$ $\delta(B,X) \not\to F$

where 'X' is any input string.

If the length of 'X' is 0 here, then A and B are said to be '0' equivalent. $[|X| = 0]$

∴ $|X| = n$, then A and B are said to be n equivalent

E.g. 1)



Transition Table

	0	1
→A	B	C
B	B	D
C	B	C
D	B	E
(E)	B	C

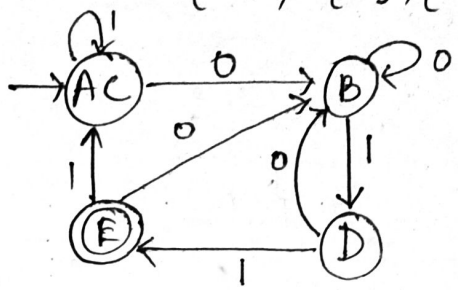
'0' Equivalence = {A, B, C D}, {E}
Seperate final and non-final states.

'1' Equivalence = {E}, {A, B, C}, {D}

'2'-Equivalence = {E}, {D}, {B}, {A, C}

'3'-Equivalence = {E}, {D}, {B}, {A, C}

The minimized DFA is



Ex-1) Construct a minimum DFA equivalent to the DFA described by following Transition Table:—

Given

	0	1
→q_0	q_1	q_5
q_1	q_6	q_2
(q_2)	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

$\pi_0 = \{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}, \{q_2\}$

$\pi_1 = \{q_0, q_4, q_6\}, \{q_1, q_7\}, \{q_3, q_5\}, \{q_2\}$

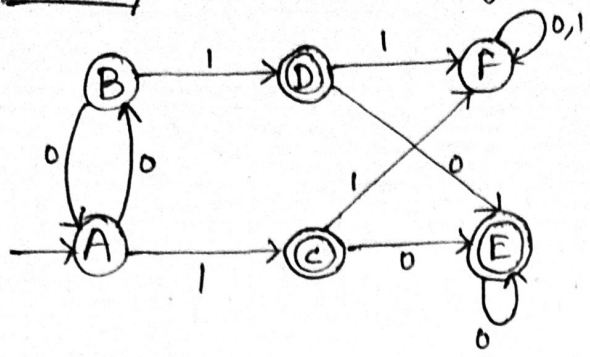
$\pi_2 = \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}, \{q_2\}$

$\pi_3 = \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}, \{q_2\}$

∴ New Transition Table:—

	0	1
→$\{q_0, q_4\}$	$\{q_1, q_7\}$	$\{q_3, q_5\}$
$\{q_6\}$	$\{q_6\}$	$\{q_0, q_4\}$
$\{q_1, q_7\}$	$\{q_6\}$	$\{q_2\}$
$\{q_3, q_5\}$	$\{q_2\}$	$\{q_6\}$
($\{q_2\}$)	$\{q_0, q_4\}$	$\{q_2\}$

Ex–2) Minimize the following DFA.



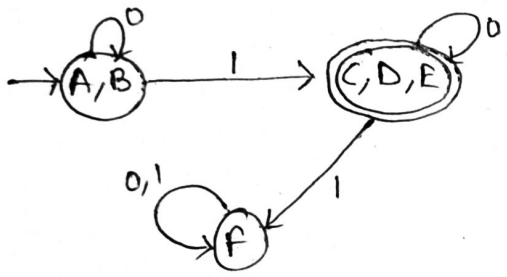
Transition Table

	0	1
A	B	C
B	A	D
C	E	F
D	E	F
E	E	F
F	F	F

$\Pi_0 = \{A, B, F\}, \{C, D, E\}$

$\Pi_1 = \{A, B\}, \{F\}, \{C, D, E\}$

$\Pi_2 = \{A, B\}, \{F\}, \{C, D, E\}$

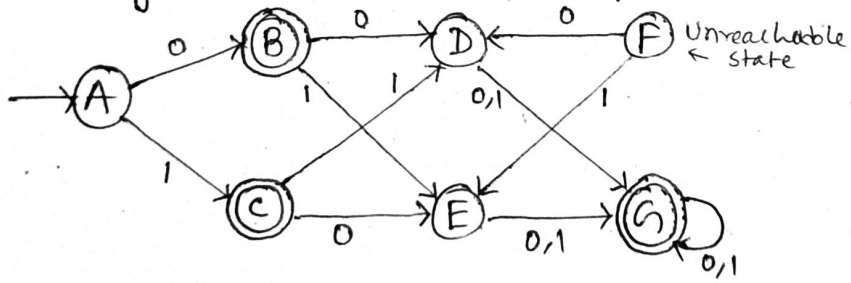


Minimized Transition Table

	0	1
{A, B}	{A, B}	{C, D, E}
{F}	{F}	{F}
{C, D, E}	{C, D, E}	{F}

Ex→(3) for unreachable states involved :-

A state is said to be unreachable ~~state~~ if there is no way it can be reached from the initial states.



Remove the state that is unreachable and design the Transition Table.

	0	1
A	B	C
B	D	E
C	E	D
D	G	G
E	G	G
G	G	G

$\Pi_0 = \{A, D, E\}, \{B, C, G\}$

$\Pi_1 = \{A, D, E\}, \{B, C\}, \{G\}$

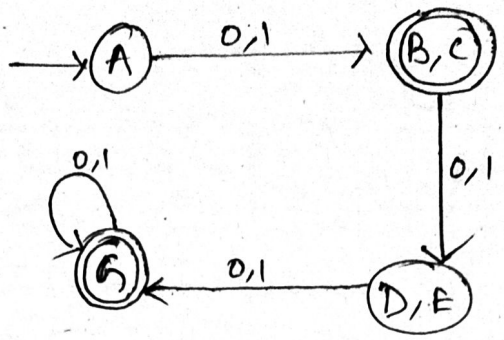
$\Pi_2 = \{A\}, \{D, E\}, \{B, C\}, \{G\}$

$\Pi_3 = \{A\}, \{D, E\}, \{B, C\}, \{G\}$

Minimized Transition Table

	0	1
{A}	{B,C}	{B,C}
{D,E}	{G}	{G}
{B,C}	{D,E}	{D,E}
{G}	{G}	{G}

Transition Diagram



Finite Automata With Outputs

Mealy Machine Mealy Machine is of 6 tuples.

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

Where, Q = Finite set of states.

Σ = Finite non-empty set of input Alphabets

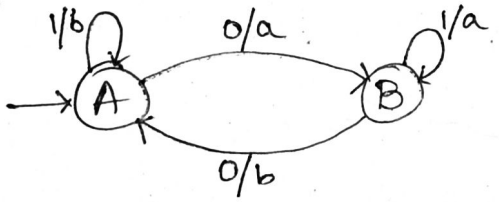
Δ = The set of Output alphabets

δ = Transition function: $Q \times \Sigma \longrightarrow Q$

λ = Output function: $\Sigma \times Q \longrightarrow \Delta$

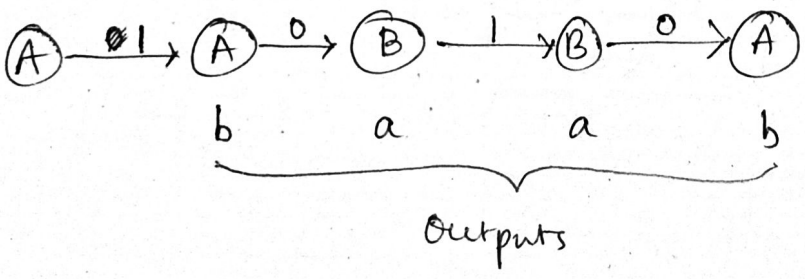
q_0 = Initial states / start states

E.g.



Here, $Q = \{A, B\}$, $\Sigma = \{0,1\}$, $\Delta = \{a,b\}$, $q_0 = A$

If we consider a string 1010 then we can see that,



Here length of input symbols and output symbols are same.

Moore Machine Moore Machine is of 6 tuples also. $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where

Q = Finite set of states

Σ = Finite non-empty set of inputs

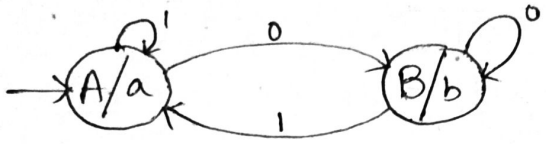
Δ = Set of Output alphabets

δ = Transition function : $Q \times \Sigma \longrightarrow Q$

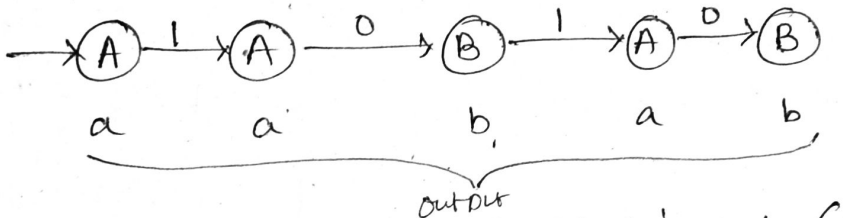
λ = Output Function : $Q \longrightarrow \Delta$

q_0 = Initial state.

E.g>



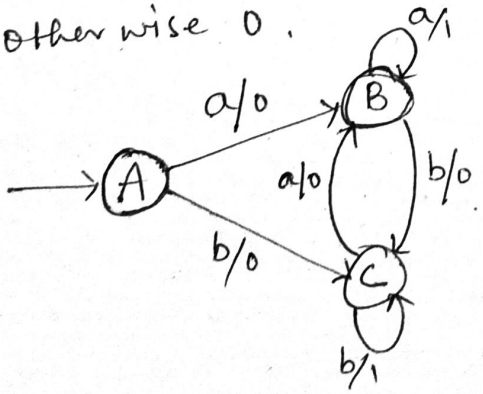
Considering an input string 1010 for the above machine we can see that,



Here the length of the output string is (1+ length of the input string.

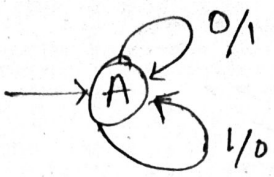
Ex-1> Design a Mealy machine accepting a language consisting of strings from Σ^* where $\Sigma = \{a, b\}$ and the string should end with either "aa" or "bb".

So, we need to assume that whenever the input is ending with 'aa' and 'bb', it will produce output as 1, otherwise 0.

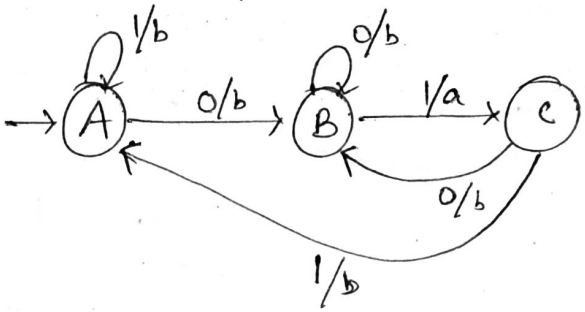


Ex-2) Mealy Machine that produces the 1's complement of any binary input string.

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



Ex-3) Mealy Machine that prints 'a' whenever the sequence '01' is encountered in any input binary string. | $\Sigma = \{0,1\}$, $\Delta = \{a,b\}$

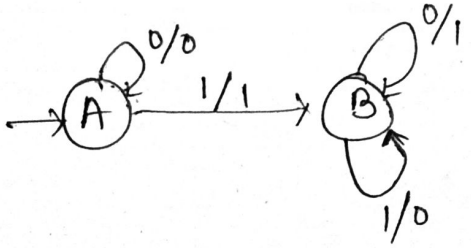


Ex-4) Mealy Machine that gives 2's complement of any binary input string. (Assuming the last carry bit is neglected).

We can see that,

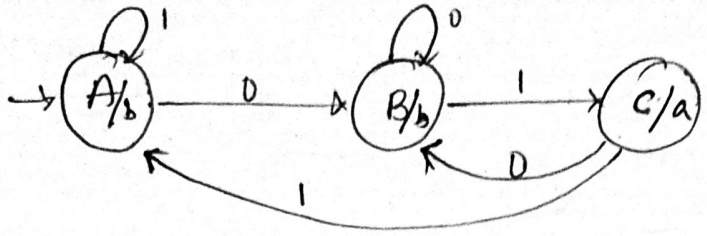
Eg → 101100 , 0111 , 0100
↓ 2's ↓ 2's ↓ 2's
010100 , 0001 , 1100

going through LSB to MSB, when the first '1' is encountered, all the previous bits upto that '1' is same and all the bits after that '1' is inverted.

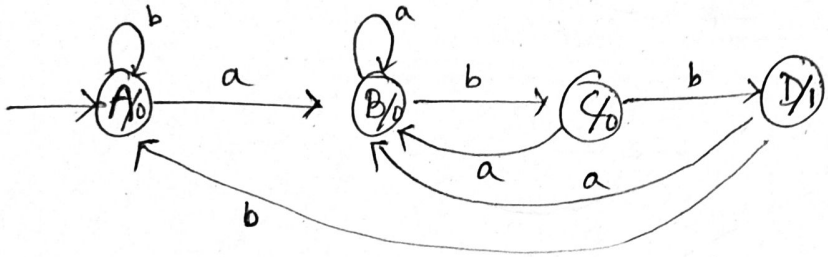


Ex-5) Construct a Moore machine that prints 'a' when ever the sequence '01' is encountered

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



Ex-6) Construct a Moore machine that counts the occurrences of the sequence 'abb' in any input strings over $\{a,b\}$. $\Sigma = \{a,b\}$, $\Delta = \{0,1\}$



Ex-7) For the following Moore Machine the input alphabet is $\Sigma = \{a,b\}$ and the output alphabet is $\Delta = \{0,1\}$. Run the following input sequence and find the respective output. → (i) aabab, (ii) abbb, (iii) ababb

States	Inputs a	Inputs 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 ⇒ Output.

(ii) a b b b

q_0 q_1 q_3 q_4 q_0

0 0 0 0 0 ⇒ output

(iii) a b a b b

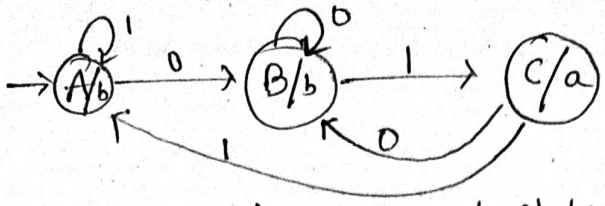
q_0 q_1 q_3 q_4 q_0 q_2

0 0 0 0 0 1 ⇒ output.

Moore to Mealy Conversion

Ex-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.

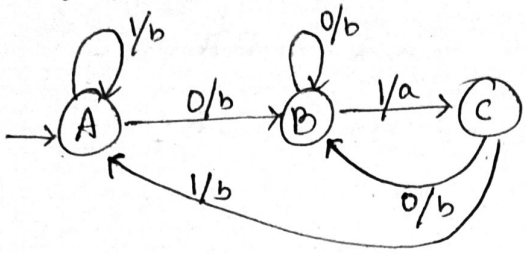
$\Sigma = \{0,1\}$, $\Delta = \{a,b\}$



Transition Table

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

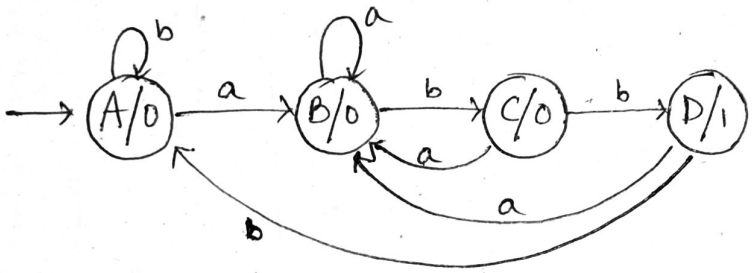
looking for each input of each state and the output of the transition we get,

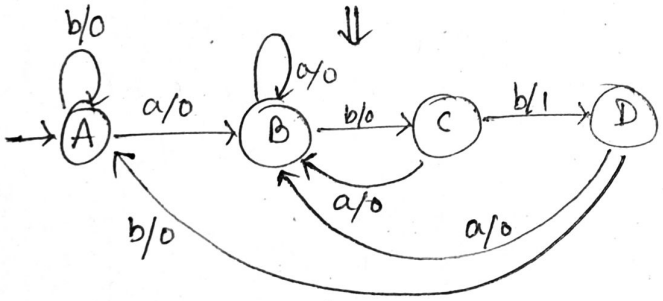


Converted transition table

State	0	1	Output
→A	B,b	A,b	b
B	B,b	C,a	b
C	B,b	A,b	a

Ex-2) Convert the following Moore machine to its Mealy equivalent. $\Sigma = \{a,b\}$, $\Delta = \{0,1\}$





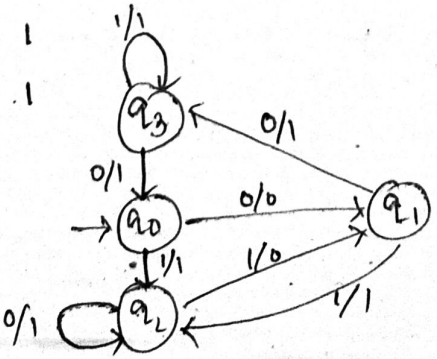
Transition Table for this Mealy Machine

State	a	b
→A	B,0	A,0
B	B,0	C,0
C	B,0	D,1
D	B,0	A,0

Ex-3) Convert the given Moore Machine to its equivalent Mealy Machine. $\Sigma = \{0,1\}$, $\Delta = \{0,1\}$

State	0	1	Outputs
→q_0	q_1	q_2	1
q_1	q_3	q_2	0
q_2	q_2	q_1	1
q_3	q_0	q_3	1

⇒

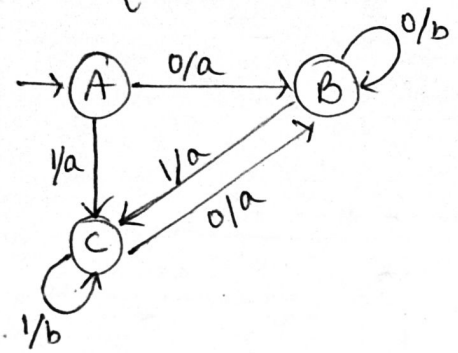


Equivalent Transition Table for Mealy Machine

State	0	1
→q_0	q_1,0	q_2,1
q_1	q_3,1	q_2,1
q_2	q_2,1	q_1,0
q_3	q_0,1	q_3,1

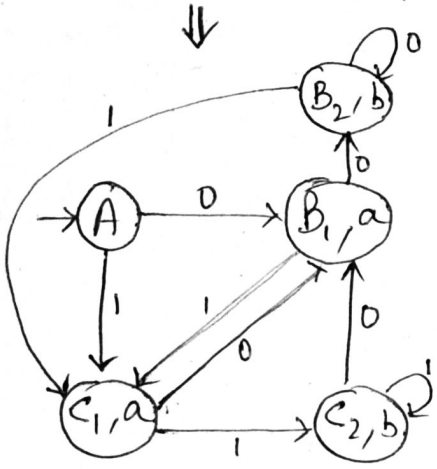
Mealy to Moore Conversion

Ex→ Convert the following Mealy Machine to its Moore equivalent.



Transition Table

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



Converted T.T for Moore

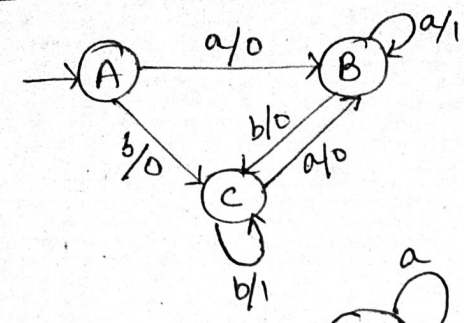
State	0	1	Output
→A	B_1	C_1	^
B_1	B_2	C_1	a
B_2	B_2	C_1	b
C_1	B_1	C_2	a
C_2	B_1	C_2	b

for more than one transition, we have to make new state for that everytime in Mealy to Moore Conversion.

NOTE> In Moore to Mealy Conversion, the no. of states were same. But in Mealy to Moore Conversion, the no. of states will increase as we've seen here. Suppose x and y are the no. of states and output consecutively in the Mealy machine, if we convert it to a Moore equivalent, it will make maximum (x*y) no. of states in worst case scenario.

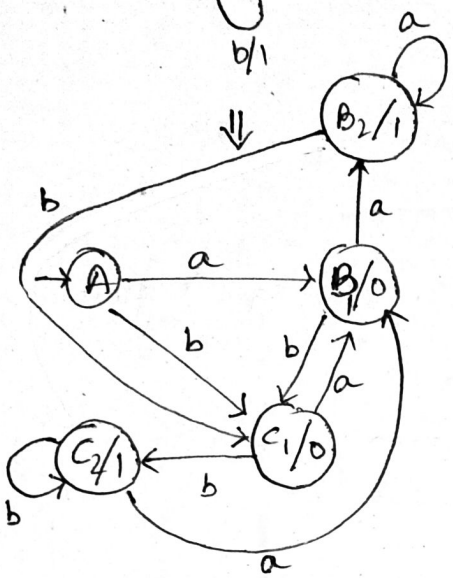
Ex-2> Given a Mealy machine here that prints '1' whenever the sequence 'aa' or 'bb' is encountered in any point in the input binary string from Σ^* where $\Sigma = \{a, b\}$. Design an equivalent Moore machine for it.

$A = \{0, 1\}$



Transition Table

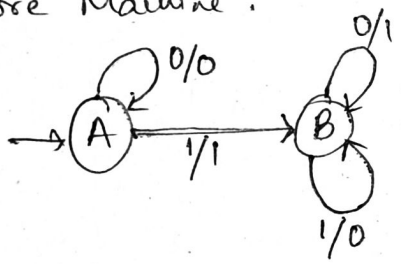
State	a	b
→A	B, 0	C, 0
B	B, 1	C, 0
C	B, 0	C, 1



Converted T.T. for Moore

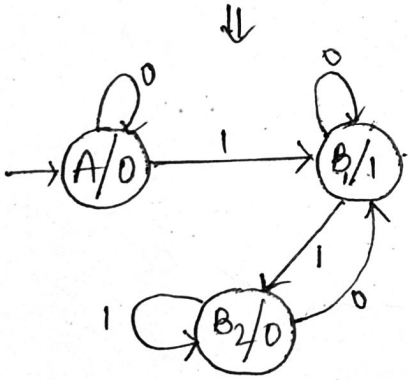
State	a	b	Output
→A	B_1	C_1	Λ
B_1	B_2	C_1	0
B_2	B_2	C_1	1
C_1	B_1	C_2	0
C_2	B_1	C_2	1

Ex-3) Convert the given Mealy Machine that give the 2's complement of any binary input to its equivalent Moore Machine.



T.T.

State	0	1
→A	A, 0	B, 1
B	B, 1	B, 0



Converted T.T. for Moore

State	0	1	Output
→A	A	B_1	0
B_1	B_1	B_2	1
B_2	B_1	B_2	0

Ex-4) Convert the Mealy Machine to Moore equivalent.

State	a	b
→q_0	q_3, 0	q_1, 1
q_1	q_0, 1	q_3, 0
q_2	q_2, 1	q_2, 0
q_3	q_1, 0	q_0, 1

⇒

State	a	b	Output
q_0	q_3	q_{11}	1
q_{10}	q_0	q_3	0
q_{11}	q_0	q_3	1
q_{20}	q_{21}	q_{20}	0
q_{21}	q_{21}	q_{20}	1
q_3	q_{10}	q_0	0

Whenever we see a state is giving different output for the entire transition, we need to split that state w.r.t. its outputs.

REGULAR EXPRESSIONS

Regular Expression are used to representing certain sets of strings in an algebric fashion.

i) Any terminal symbols (symbols $\in \Sigma$) including $\wedge$ and ϕ are regular expression. Eg. a, b, c, $\wedge$, ϕ

ii) The union of two regular expressions is also a regular expression. Eg. $R_1, R_2, (R_1 + R_2)$.

iii) The concatenation of two regular expressions is also a regular expression. Eg. $R_1, R_2, (R_1 . R_2)$

iv) The iteration (or closure) of a regular expression is also a regular expression. (R, R^*)

v) The regular expression over Σ are precisely those obtained recursively by the application of the above rules once or several times.

Eg.) 1) $\{0,1,2\}$ it means 0 or 1 or 2

$\therefore R = 0+1+2$

2) $\{\wedge, ab\}$ $\therefore R = \wedge + ab$.

3) $\{abb, a, b, bba\}$ $\therefore R = abb + a + b + bba$

4) $\{\wedge, 0, 00, 000, \ldots\}$ $\therefore R = 0^*$

5) $\{1, 11, 111, 1111, \ldots\}$ $\therefore R = 1^+$

IDENTITIES OF REGULAR EXPRESSION

1) $\phi + R = R$ [ϕ = empty set] here '+' is union.

2) $\phi R + R\phi = \phi$

3) $\epsilon R + R\epsilon = R$

4) $\epsilon^* = \epsilon$ and $\phi^* = \epsilon$

5) $R + R = R$

6) $R^* R^* = R^*$

7) $RR^* = R^* R$ $[RR^* = R^+]$

8) $(R^*)^* = R^*$

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

10) $(PQ)^* P = P(QP)^*$

11) $(P+Q)^* = (P^* Q^*)^* = (P^* + Q^*)^*$

12) $(P+Q)R = (PR + QR)$ and $R(P+Q) = RP + RQ$

ARDEN'S THEOREM

If P and Q are two regular expressions over Σ, and if P does not contain ϵ, then the following equation in R given by $R = Q + RP$ has a unique solution i.e. $R = QP^*$.

Proof

$$R = Q + RP \quad \text{——— (i)}$$

$$= Q + QP^* P \qquad [R = QP^*]$$

$$= Q(\epsilon + PP^*)$$

$$= QP^* \qquad [\epsilon + RR^* = R^*]$$

$\therefore$ $R = QP^*$ is a solution.

Now $R = Q + RP$

$$= Q + (Q+RP)P \qquad [R = Q+RP]$$

$$= Q + QP + RP^2$$

$$= Q + QP + (Q+RP)P^2 \qquad [R = Q+RP]$$

$$= Q + QP + QP^2 + RP^3$$

$$\vdots$$

$$= Q + QP + QP^2 \text{---} QP^n + RP^{n+1}$$

$$= Q + QP + QP^2 + \cdots\cdots + QP^n + QP^*(P^{n+1})$$

$$= Q(\epsilon + P + P^2 + \cdots\cdots + P^n + P^*P^{n+1})$$

$$= QP^* \qquad [P^* = (\epsilon + P + P^2 + \cdots P^n + P^*P^{n+1})]$$

$\therefore$ $R = QP^*$ is the unique solution.

Ex-1) Prove that $(1+00^*1) + (1+00^*1)(0+10^*1)^*(0+10^*1)$ is equal to $0^*1(0+10^*1)^*$

$$L.H.S = (1+00^*1) + (1+00^*1)(0+10^*1)^*(0+10^*1)$$

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

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

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

$$= 0^*1(0+10^*1)^* = R.H.S$$

Ex-2) Design Regular Expression for the following languages over $\{a,b\}$.

1) Language accepting strings of length exactly 2.
2) Language accepting strings of length atleast 2.
3) Language accepting strings of length atmost 2.

---

1) $L_1 = \{aa, ab, ba, bb\}$

$\therefore R = aa + ab + ba + bb$

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

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

2) $L_2 = \{aa, ab, ba, bb, aaa, \cdots\cdots\}$

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

3) $L_3 = \{\epsilon, a, b, aa, bb, ab, ba\}$

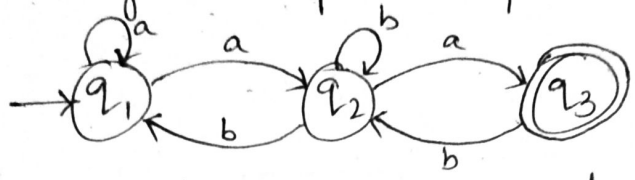
$\therefore R = \epsilon + a + b + aa + bb + ab + ba$

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

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

NFA to Regular Expression

Find the Regular Expression of the given NFA.



Incoming transitions should be checked for every step.

$q_3 = q_2 a$ ---- (1)

$q_2 = q_1 a + q_2 b + q_3 b$ ---- (2)

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

(1) $q_3 = q_2 a$

$= (q_1 a + q_2 b + q_3 b) a$ [Substituting q_2]

$= q_1 aa + q_2 ba + q_3 ba$ ---- (4)

(2) $q_2 = q_1 a + q_2 b + q_3 b$

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

$= q_1 a + q_2 (b + ab)$ [$R = Q + RP$]

$= Q P^*$

$= q_1 a (b + ab)^*$ ---- (5)

(3) $q_1 = \epsilon + q_1 a + q_2 b$ ---- (6)

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

$= \epsilon + q_1 (a + a(b+ab)^*) b$ [$R = Q + RP$]

$\therefore q_1 = \in.\left((a+a(b+ab)^*)b\right)^*$ [$\in R = R$]

$\therefore q_1 = \left((a+a(b+ab)^*)b\right)^*$ ..... (6)

∴ Final state

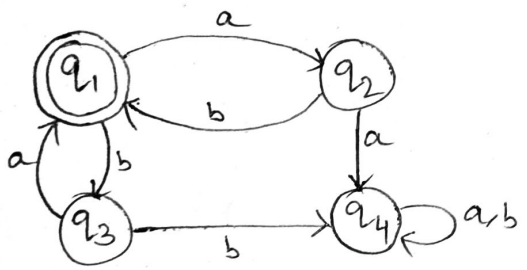
$q_3 = q_2 a$

$= \left(q_1 a(b+ab)^*\right)a$ from (6)

$$\boxed{q_3 = \left(a+a(b+ab)^*b\right)^* a(b+ab)^*a}$$ Ans

DFA to Regular Expression Incoming transitions are to be considered —

Ex-1)



$q_1 = \in + q_2 b + q_3 a$ — (i)

$q_2 = q_1 a$ — (ii)

$q_3 = q_1 b$ — (iii)

$q_4 = q_2 a + q_3 b + q_4 a + q_4 b$ — (iv)

(i) $q_1 = \in + q_2 b + q_3 a$

$= \in + q_1 ab + q_1 ba$

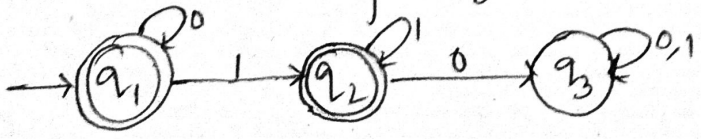
$= \in + q_1(ab + ba)$ [$R = Q + RP$]

$= \in (ab+ba)^*$ [$\in R = R$]

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

Here q_1 is the final state, that means we've already got our final state. So, the required RE is already obtained.

Ex-2) Find the RE for given DFA



$q_1 = \in + q_1 0$ — (i)

$q_2 = q_1 1 + q_2 1$ — (ii)

$q_3 = q_2 0 + q_3 0 + q_3 1$ — (iii)

(1) $q_1 = \epsilon + q_1 0$ $[R = Q + RP = QP^*]$

$= \epsilon \cdot 0^*$ $[\epsilon R = R]$

$q_1 = 0^*$ ——— (iv)

(2) $q_2 = q_1 1 + q_2 1$ $[q_1 = 0^*]$

$= 0^* 1 + q_2 1$ $[R = Q + RP]$

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

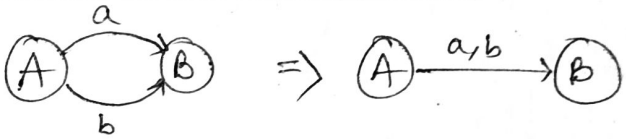
∴ RE = Union of all the final states.

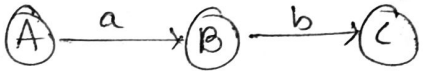
$= 0^* + 0^* 1 (1^*)$

$= 0^* (\epsilon + 1 (1^*))$ $[\epsilon + RR^* = R^*]$

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

REGULAR EXPRESSION TO FINITE AUTOMATA

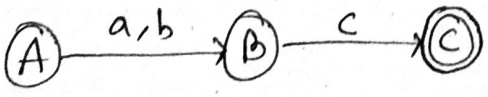
RULES (i) $(a+b)$ ⟹ 

(ii) (ab) ⟹ 

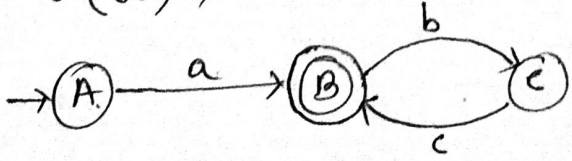
(iii) a^* ⟹ 

Ex-1) RE = ba^*b, Convert it to an F.A.

Ex-2) RE = $(a+b)c$, convert to an f.A.



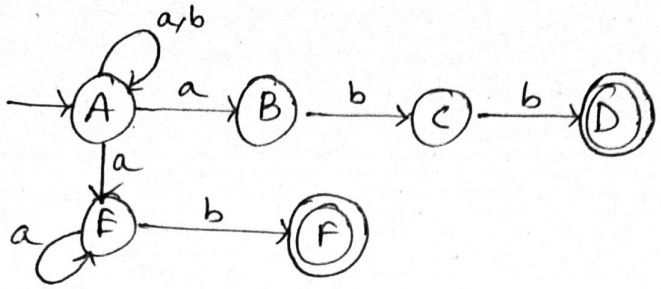
Ex-3) RE = $a(bc)^*$, convert it to an F.A.



Ex-4) RE = $(a|b)^*(abb|a^+b)$

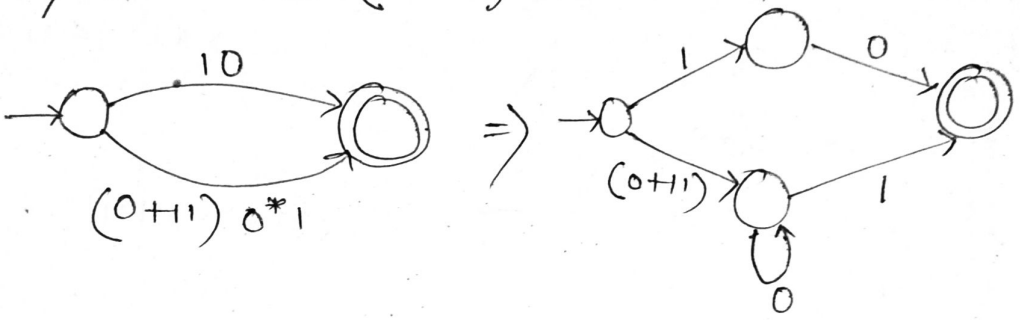
[| → or symbol same as '+']

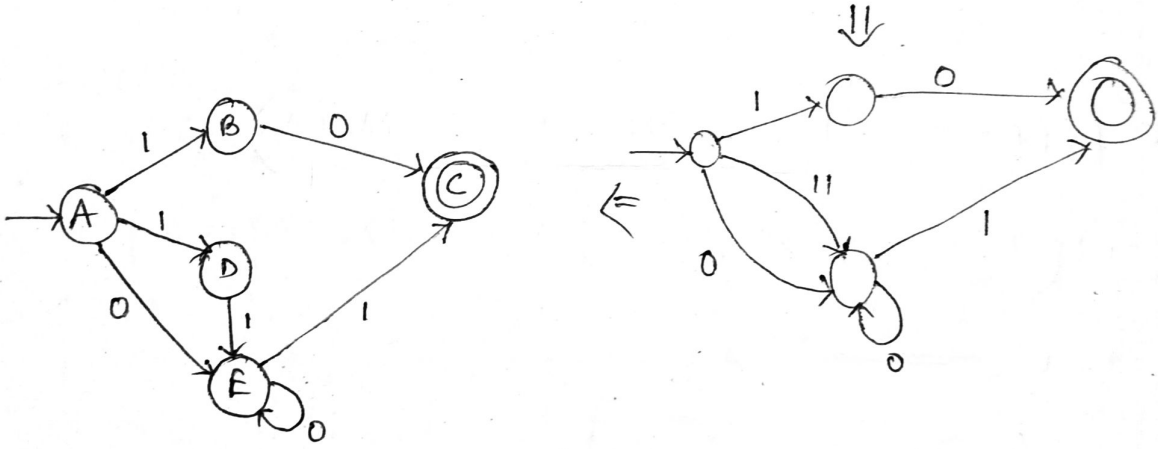
[$a^+ = \{a, aa, aaa, \ldots\}$]



This is an NFA.

Ex-5) RE = $10+(0+11)\,0^*1$





EQUIVALENCE OF TWO FINITE AUTOMATA

Step 1) For any pair of states $\{q_i, q_j\}$ the transition for input $a \in \Sigma$ is defined by $\{q_a, q_b\}$

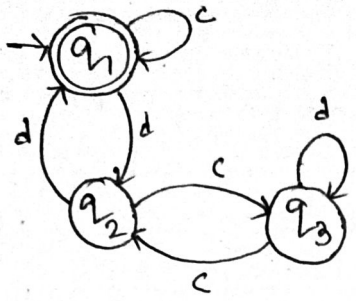
where $\delta(q_i, a) = q_a$ and $\delta(q_j, a) = q_b$

The two automata are not equivalent if for a pair $\{q_a, q_b\}$, one is Intermediate state and the other is final state.

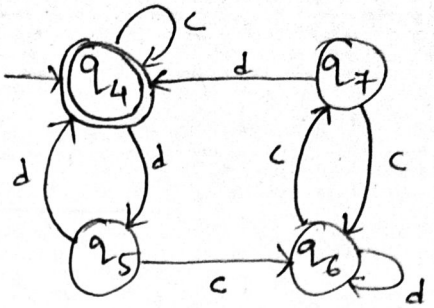
Step 2) If the initial state is final state of one automaton then in second automaton also Initial state must be

Final state for them to be equivalent.

E.g.)



A



B

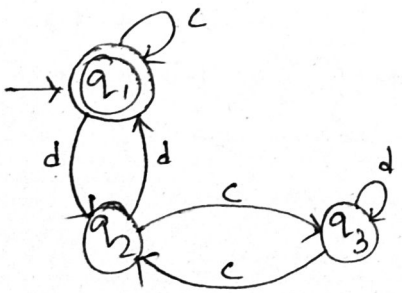
Here $\Sigma = \{c, d\}$, Now,

States	c	d
(q_1, q_4)	(q_1, q_4)	(q_2, q_5)
$\rightarrow (q_2, q_5)$	(q_3, q_6)	(q_1, q_4)
(q_3, q_6)	(q_2, q_7)	(q_3, q_6)
$\rightarrow (q_2, q_7)$	(q_3, q_6)	(q_1, q_4)

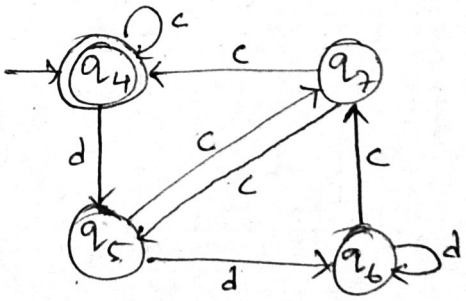
Here, we can see that all the pairs of states contains both intermediate or both final states, so the first condition is true. We can also see that both the automata have their final state as the initial state.

∴ A and B are equivalent.

E.g. 2)



A



B

We can see that both the automata have their final states as the initial one. So the second condition is true.

States	c	d
$\{q_1, q_4\}$	$\{q_1, q_4\}$	$\{q_2, q_5\}$
$\{q_2, q_5\}$	$\{q_3, q_7\}$	$\{q_7, q_6\}$ → This pair is breaking the first condition as q_1 is final and q_6 is a non-final state.

∴ A and B are not equivalent.

PUMPING LEMMA

» Pumping Lemma is used to prove that a language is Not Regular.

» It cannot be used to prove that a language is Regular.

LEMMA If A is a Regular language, then A has a pumping length 'p' such that any string 's' where $|s| \geq p$ may be divided into three parts $s = xyz$ such that the following conditions must be true:

(1) $xy^iz \in A$ for every $i \geq 0$

(2) $|y| > 0$

(3) $|xy| \leq p$

To prove that a language is not Regular using Pumping Lemma, follow the below steps (Using Contradiction)

→ Assume that A is Regular

→ It has to have a Pumping Length (Say 'p')

→ All strings longer than p can be pumped

$|s| \geq p$

→ Now find a string 'S' in A such that $|S| \geq p$.

→ Divide 'S' into $x\,y\,z$.

→ Show that $x\,y^i\,z \notin A$ for some i.

→ Then consider all ways that S can be divided into $x\,y\,z$.

→ Show that none of these can satisfy all the 3 pumping conditions at the same time.

→ S cannot be PUMPED == CONTRADICTION

Example 1) Using Pumping Lemma prove that the Language $A = \{a^n b^n \mid n \geq 0\}$ is Not Regular.

★ **Proof:** Assume that A is Regular.

Pumping length = p.

$S = a^p b^p$

Let us assume $p = 7$, We have to divide S into xyz form

$\therefore S = aaaaaaabbbbbbb$

Now, Case no. I) y is in the 'a' part.

aaaaaaabbbbbbb
x y z

if $i = 2$,
$xy^2z = aaaaaaaaaabbbbbbb$
Here, 11 no. of 'a's and 7 no. of 'b's are there and $|xy| = 6$

case-II) y is in the 'b' part.

aaaaaaabbbbbbb
x y z

Here,
$xy^2z = aaaaaaabbbbbbbbbbbbb$
Here 7 no of 'a's and 11 no of 'b's and $|xy| = 13$

Case-III) y is in the 'a' as well as b part.

aaaaaaabbbbbbb
x y z

Here,
$xy^2z = aaaaaaaabbbaabbbbbbb$
pattern is mismatching.
and $|xy| = 9$

$\therefore xy^iz \notin A$ for $i = 2$ here.
$\therefore |xy| \not\leq p$ for case-II and III

$\therefore$ S can't be pumped
$\therefore$ A is not Regular.

Example-2> Using pumping lemma, prove that A = $\{yy \mid y \in \{0,1\}^*\}$ is not Regular.

Assume that A is Regular, then it must have a pumping length p.

Now, $S = 0^p 1 0^p 1$

We need to divide 's' in x, y, z.

Assuming P = 7.

Take-1> S = 00000001 0000000 1
x y z

if, i = 2,
xy^2z = 00000000 0100000001
first half of the string is not equal to the second half.
∴ so for i=2, $xy^iz \notin A$

Here, $|y| > 0$ and $|xy| = 6 \not\leq P$

∴ ~~A is not~~ ~~S is~~ S cannot be pumped

∴ A is not Regular.

REGULAR GRAMMAR

Noam Chomsky gave a Mathematical model of Grammar which is effective for writing computer languages.

The four types of Grammar according to Noam Chomsky are:-

Grammar type	Grammar Accepted	Language Accepted	Automation
TYPE - 0	Unrestricted	Recursively Enumerable	Turing Machine
TYPE - 1	Context Sensitive	Context Sensitive	Linear Bounded
TYPE - 2	Context free	Context free	Pushdown
TYPE - 3	Regular	Regular	FSM

GRAMMAR: A Grammar 'G' can be formally described using 4 tuples as $G = (V, T, S, P)$ where,

V = Set of variables or Non-terminal symbols.

T = Set of Terminal Symbols.

S = Start Symbol.

P = Production Rules for Terminals and Non-Terminals.

A production rules has the form $\alpha \rightarrow \beta$ where α and β are strings on $V \cup T$ and atleast one symbol $\alpha \in V$.

E.g.) $G = (\{S, A, B\}, \{a, b\}, S, \{S \rightarrow AB, A \rightarrow a, B \rightarrow b\})$

Here, $V = \{S, A, B\}$

$T = \{a, b\}$

$S = S$

$P = S \rightarrow AB,\ A \rightarrow a,\ B \rightarrow a.$

Now, Let,

$$\begin{aligned} S &\rightarrow AB \\ &\rightarrow aB \quad [A \rightarrow a] \\ &\rightarrow ab \quad [B \rightarrow b] \end{aligned}$$

This can find which kind of strings are accepted by this G.

REGULAR GRAMMAR) Regular grammar can be divided into two types.

Left-Linear Grammar

A Grammar is said to be left linear if all productions are of the form

$A \rightarrow Bx$

$A \rightarrow x$

where $A, B \in V$ and $x \in T$

Right-Linear Grammar

A Grammar is said to be Right Linear if all productions are of the form,

$A \rightarrow xB$

$A \rightarrow x$

where $A, B \in V$ and $x \in T$.

Eg. $S \rightarrow abS \mid b$ — as the Non terminal 'S' lies at the right part of the Production Rule, Its a right linear Grammar.

$S \rightarrow Sbb \mid a$ — Non-terminal 's' lies at the left part of the production rule, its a left linear grammar.

DERIVATION FROM A GRAMMAR

The set of all strings that can be derived from a Grammer is said to be the LANGUAGE generated from that Grammar.

Example—1) Consider the grammar,

$G1 = (\{S, A\}, \{a, b\}, S, \{S \rightarrow aAb, aA \rightarrow aaAb, A \rightarrow \epsilon\})$

Now,

$S \rightarrow aAb$ [by $S \rightarrow aAb$]

$\therefore S \rightarrow aaAbb$ [by $aA \rightarrow aaAb$]

$\therefore S \rightarrow aaaAbbb$ [by, $aA \rightarrow aaAb$]

$\therefore S \rightarrow aaabbb$ [by $A \rightarrow \epsilon$]

this is how we can produce the language. by deriving all the string like this.

Example) $G2 = (\{S, A, B\}, \{a, b\}, S, \{S \rightarrow AB, A \rightarrow a, B \rightarrow b\})$

Now, $S \rightarrow AB$

$\therefore S \rightarrow ab$

As, this is the only string that can be generated by the grammar.

$\therefore \quad L(G2) = \{(a,b)\}$

Example-3) $G3 = \left(\{S,A,B\}, \{a,b\}, S, \{S \to AB,\ A \to aA \mid a,\ B \to bB \mid b\}\right)$

Now,

$S \to AB$	$S \to AB$	$S \to AB$	$S \to AB$
$\to ab$	$\to aAbB$	$\to aAb$	$\to abB$
	$\to aabb$	$\to aab$	$\to abb$

$L(G3) = \{ab, a^2b^2, a^2b, ab^2, \ldots\}$

$= \{a^m b^n \mid m \geq 0 \text{ and } n \geq 0\}$

CONTEXT FREE GRAMMAR AND LANGUAGE

In formal language theory, a context free language is a language generated by some Context free Grammar. The set of all CFL is identical to the set of languages accepted by pushdown automata.

Context Free Grammar is defined by 4 tuples a $G = \{V, \Sigma, S, P\}$ where,

V = Set of variables or Non-terminals

Σ = Set of terminals.

S = Start Symbol.

P = Production Rule.

Context Free Grammar has Production Rule of the form, $A \longrightarrow \alpha$, where $\alpha = \{V \cup \Sigma\}^*$ and $A \in V$.

Example: For generating a language that generates equal no. of a's and b's in the form $a^n b^n$, the context free grammer will be defined as

$$G = (\{S, A\}, \{a, b\}, \{S \longrightarrow aAb,\ A \longrightarrow aAb \mid \epsilon\})$$

Now, $S \longrightarrow aAb$

$\longrightarrow aaAbb$

$\longrightarrow aaaAbbb$

$\longrightarrow aaa\epsilon bbb$

$\therefore L(G) = \{ab, a^2b^2, a^3b^3, \dots\}$

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

Method to find whether a string belong to a Grammer or not

Step 1: Start with the start symbol and choose the closest production that matches to the given string

Step 2: Replace the variables with its most appropriate production. Repeat the process until the string is generated or ~~until~~ until no other production are left.

Example 1) Verify the grammer $S \rightarrow 0B \mid 1A$

$A \rightarrow 0 \mid 0S \mid 1AA \mid \wedge$

$B \rightarrow 1 \mid 1S \mid 0BB$

generates the string 00110101.

Now,

$S \rightarrow 0B$ [$S \rightarrow 0B$]

$\rightarrow 00BB$ [$B \rightarrow 0BB$]

$\rightarrow 001B$ [$B \rightarrow 1$]

$\rightarrow 0011S$ [$B \rightarrow 1S$]

$\rightarrow 00110B$ [$S \rightarrow 0B$]

$\rightarrow 0011010S$ [$B \rightarrow 1S$]

$\rightarrow 00110101$ [$B \rightarrow 1$]

∴ So the given grammar can generate the string 00110101.

Example-2) Verify whether the Grammar $S \rightarrow aAb$, $A \rightarrow aAb \mid \wedge$ generates the string aabbb.

Now,

$S \rightarrow aAb$ $[S \rightarrow aAb]$

$S \rightarrow aaAbb$ $[A \rightarrow aAb]$

$S \rightarrow aabb$ $[A \rightarrow \Lambda]$ ✗

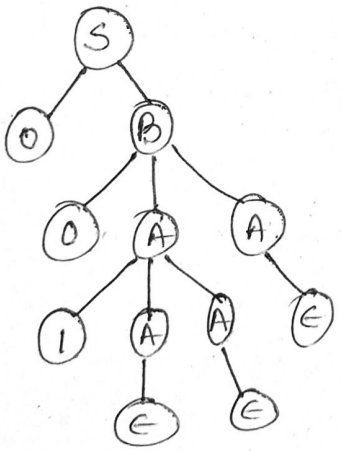
$S \rightarrow aaaAbbb$ $[A \rightarrow aAb]$

$S \rightarrow aaabbb$ $[A \rightarrow \Lambda]$ ✗

∴ So the given grammar can't generate the string aabbb.

DERIVATION TREE A derivation tree or parse tree is an ordered rooted tree that graphically represents the semantic information of strings derived from a context free Grammar.

Example) For the grammar $G = \{V, T, P, S\}$ where $S \rightarrow 0B$, $A \rightarrow 1AA \mid \in$, $B \rightarrow 0AA$.



Root Vertex: Must be labelled by the Start Symbol.

Vertex: Labelled by Non-terminal Symbols

Leaves: Labelled by terminal symbols or $\in$.

Derivation for → 001

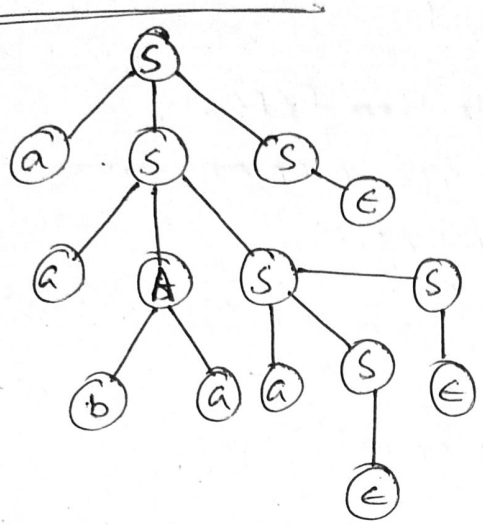
i) **Left Derivation Tree** A left derivation tree is obtained by applying production to the leftmost variable in each step.

ii) **Right Derivation Tree** A right derivation tree is obtained by applying production to the right-most variable in each step.

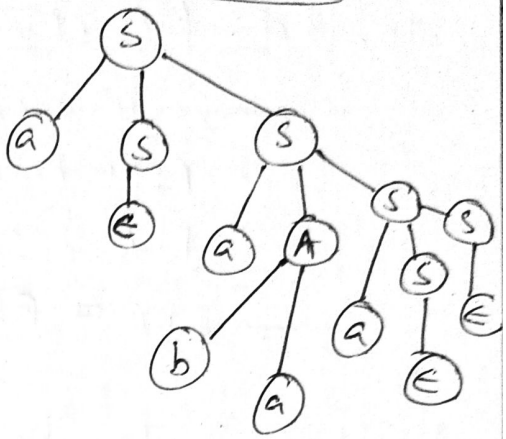
E.g. Generate the string aabaa from Grammar

$S \rightarrow aAS \mid aSS \mid \epsilon$, $A \rightarrow SbS \mid ba$

Left Derivation



Right Derivation



Ambiguous Grammar

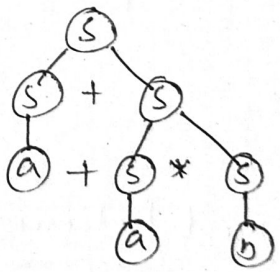
A Grammar is said to be ambiguous if there exists two or more LDT or RDT (Left Derivation Tree or Right Derivation Tree) for a string ω.

Example) $G = \{\{S\}, \{a+b, +, *\}, P, S\}$ where P is consisting of $S \rightarrow S+S \mid S*S \mid a \mid b$.

Let, we are willing to generate $a+a*b$.

LDT-1)

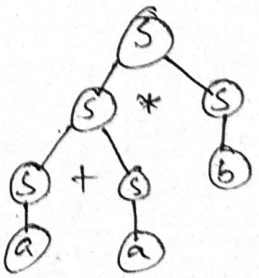
$S \rightarrow S+S$
$\rightarrow a+S \quad [S \rightarrow a]$
$\rightarrow a+S*S \quad [S \rightarrow S*S]$
$\rightarrow a+a*S \quad [S \rightarrow a]$
$\rightarrow a+a*b \quad [S \rightarrow b]$



LDT-2)

$S \rightarrow S*S \quad [S \rightarrow S*S]$
$\rightarrow S+S*S \quad [S \rightarrow S+S]$
$\rightarrow a+S*S \quad [S \rightarrow a]$
$\rightarrow a+a*S \quad [S \rightarrow a]$
$\rightarrow a+a*b \quad [S \rightarrow b]$

∴ Thus this grammar is Ambiguous.



PUSHDOWN AUTOMATA

A push-down automata (PDA) is a way to implement a context-free G in a similar way we design finite automata for regular grammer.

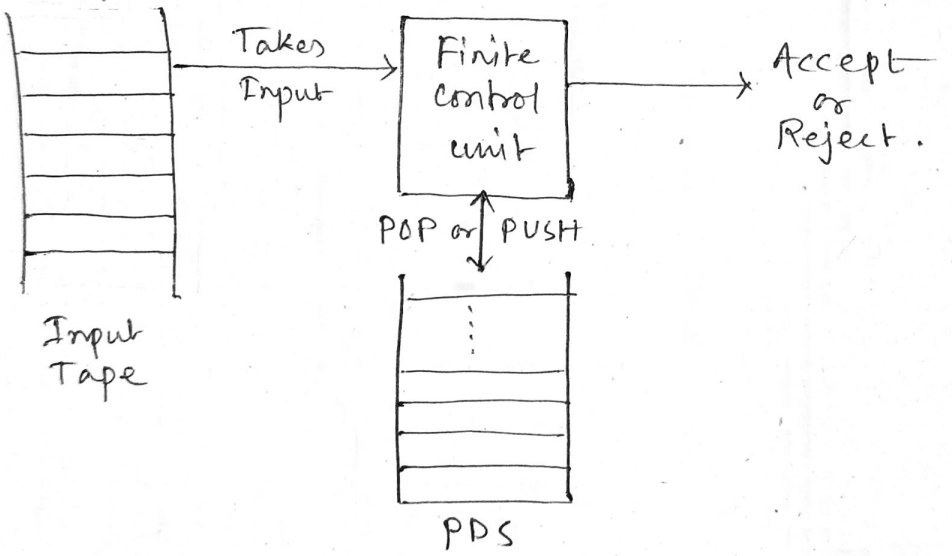
→ It is more powerful than FSM.

→ FSM has a very limited memory but PDA has more memory.

→ PDA = FSM + PDS (Pushdown Store) stack

A pushdown automata has 3 components:

1) An input tape
2) A finite control unit
3) A stack with infinite size.



Formal Definition > A pushdown automata is formally defined by 7 tuples.

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

Where,

Q = A finite set of states

Σ = A finite set of input symbols.

Γ = A finite stack Alphabet.

δ = The transition function

q_0 = The start symbol.

Z_0 = The start stack symbol.

F = A set of all Final / Accepting states $(F \subseteq Q)$

Unlike FSM, here the δ takes an argument a triple (q, a, X) where:

i) 'q' is a state in Q $[q \in Q]$

ii) 'a' is either an Input symbol in Σ or ϵ
$[a \in \Sigma$ or $a = \epsilon]$

iii) X is a stack symbol that is the member of Γ $[X \in \Gamma]$

The output of δ is finite set of pair (P, γ)
where, P = a new state $[P \in Q]$

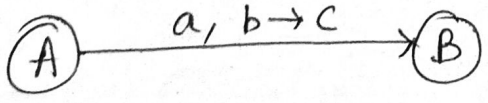
γ = a string of stack symbols that replaces X at the top of the PDS stack

E.g. If $\gamma = \epsilon$, then stack is popped.

If $\gamma = X$, then stack is unchanged.

If $\gamma = YZ$, then X is replaced by Z and Y is pushed onto the stack.

GRAPHICAL REPRESENTATION



a = input symbol (maybe ϵ)

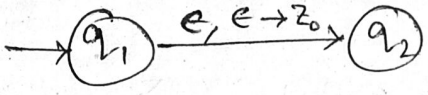
b = Symbol on the top of the stack (This symbol is popped) (Here ϵ means the stack is neither read nor popped)

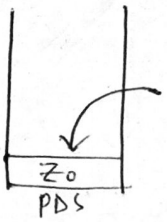
c = This symbol is pushed onto the stack ('ϵ' means nothing is pushed)

Example> Construct a PDA that accepts

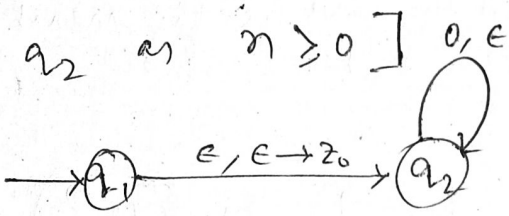
$$L = \{0^n 1^n \mid n \geq 0\}$$

We need to place z_0 first in the PDS. For this, we need a transition with input ϵ and also pop nothing from the stack as the stack is empty before this transition.

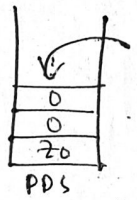




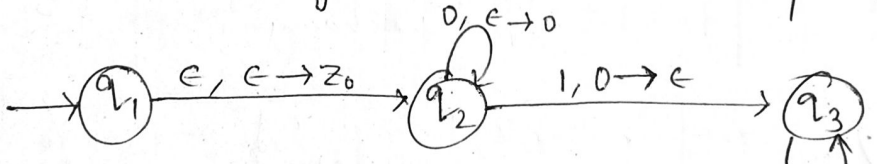
[Now from the question, we need to place n - no. of 0's first and to do that we need here a self loop at q_2 as $n \geq 0$]

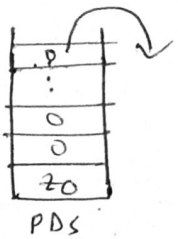


[In q_2, we gave self loop of input 0 and ~~on~~ place 0 onto the stack. For n-no. of 0's, n-no. of 0's are pused onto the stack.]

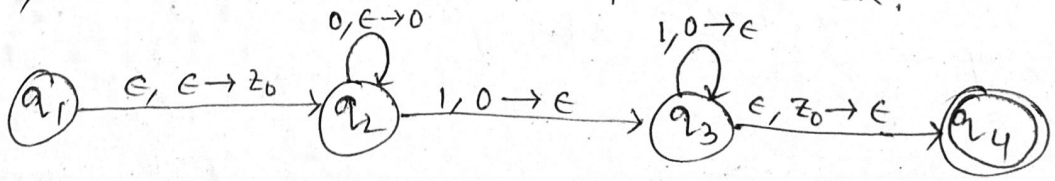


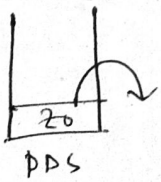
Now, for input 1, we need a state change and pop elements from stack top one by one so that we can generate same no. of 1's as the 0's.





Atlast, we need to pop the last element z_0, by which, we've checked the end of the stack.





[Note: Z_0 - is denoted by the end of stack.]

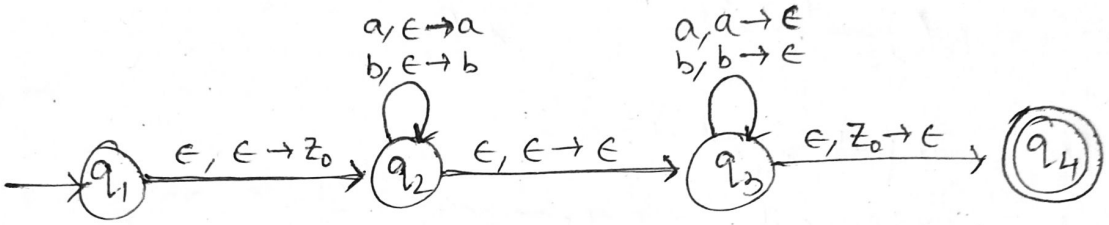
Example P-1) Construct a PDA that accepts Even palindromes of the form

$$L = \{ W W^R \mid W = (a+b)^+ \}$$

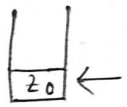
Now,

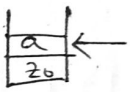
Here, W = first half of palindrome

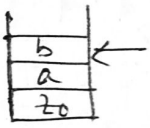
W^R = Reverse of W / second half of palindrome.

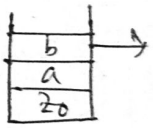


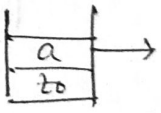
Checking) let us take 'abba'.

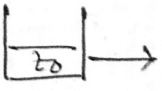
In q_1 → Z_0 pushed onto the stack 

In q_2 → a " " " " 

" " → b " " " " 

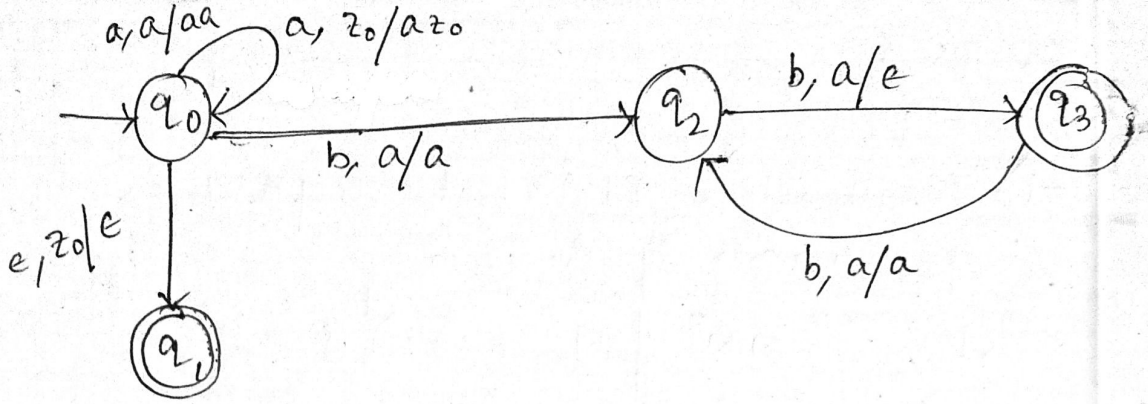
" q_3 → b is popped from the top 

" q_3 → a " " " " " 

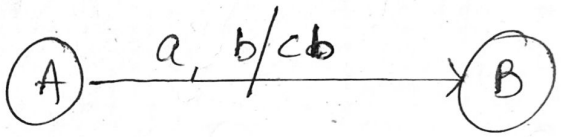
" q_4 → Z_0 " " " " " 

So, the PDA is functional.

Example P-2) $L = \{ a^n b^{2n} \mid n \geq 0 \}$



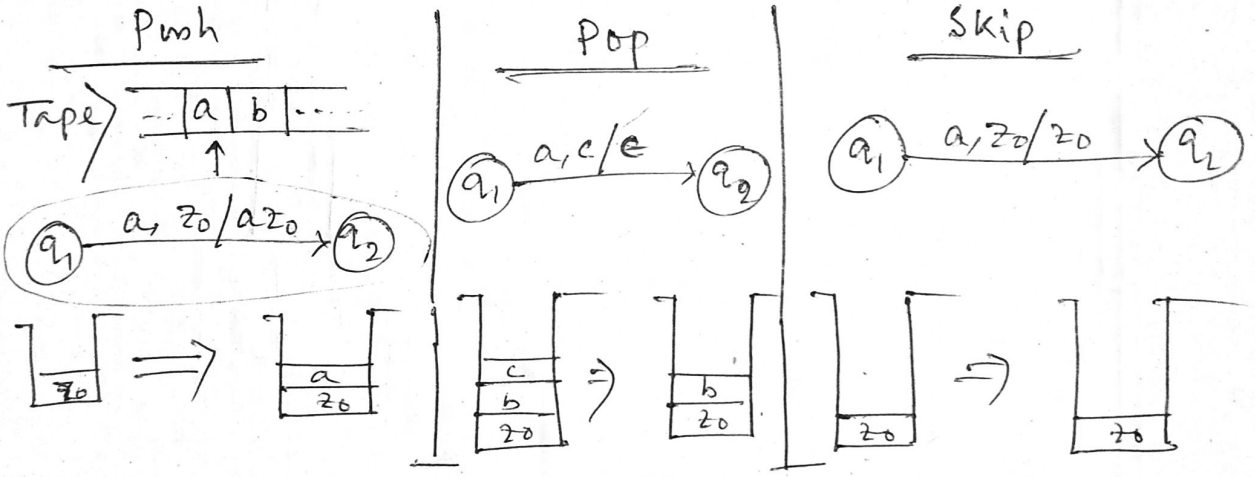
Graphical Representation – II



a = input symbol.

b = element to popped out / Stack top (Current)

c = element to be pushed in. stack top will now updated to c.



Example P–3) $L = \{ wcw^R \mid w \in (a+b)^* \}$ odd palindrome.