TOC

Introduction

FSM :- Finite State Machine
CFL :- Context Free language
↑
set of strings

Finite State Machine (Pre-requisites)

Symbol :- anything like a, b, c, 0, 1, 2, 3 ...

Alphabet :- Σ — collection of symbols.
E.g :- {a, b}, {d, e, f, g}, {0, 1, 2} ...

String :- A sequence of symbols
E.g :- a, b, 0, 1, aa, bb, ab, 01, ...

language :- set of strings

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

L_1 = set of all strings of length 2.

= {00, 01, 10, 11} (finite set)

L_2 = set of all strings of length 3.

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

(finite set)

L_3 = set of all strings ~~of~~ that begin with 0

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

infinite set

Powers of Σ :-

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

Σ^0 = set of all strings of length 0:

$\Sigma^0 = \{\in\}$

Σ^1 = set of all strings of length 1:

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

Σ^2 = set of all strings of length 2:

$\Sigma^2 = \{00, 01, 10, 11\}$

Σ^3 = set of all strings of length 3.

$\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

Σ^n = set of all strings of length n.

Cardinality :- Number of elements in a set.

$$|\Sigma^n| = 2^n$$

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

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

= set of all possible strings of all lengths over 0 and 1.

(infinite set)

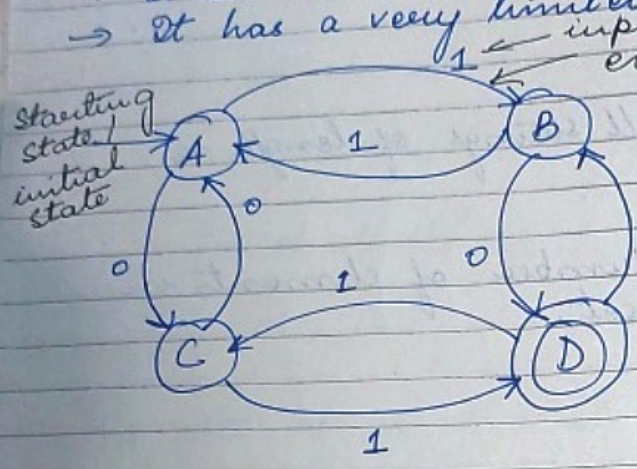
Finite State Machine

Finite automata

- FA with output
  - Moore machine
  - Mealy machine
- FA without output
  - DFA
  - NFA
  - $\in$ NFA

DFA - Deterministic Finite Automata

→ It is the simplest model of computation
→ It has a very limited memory.



These circles A, B, C, D are known as states.

Every DFA can be defined using five tuples $(Q, \Sigma, q_0, F, \delta)$

Q = set of all states
Σ = inputs
q_0 = initial state
F = set of final states
δ = transition function that maps $Q \times \Sigma \rightarrow Q$.

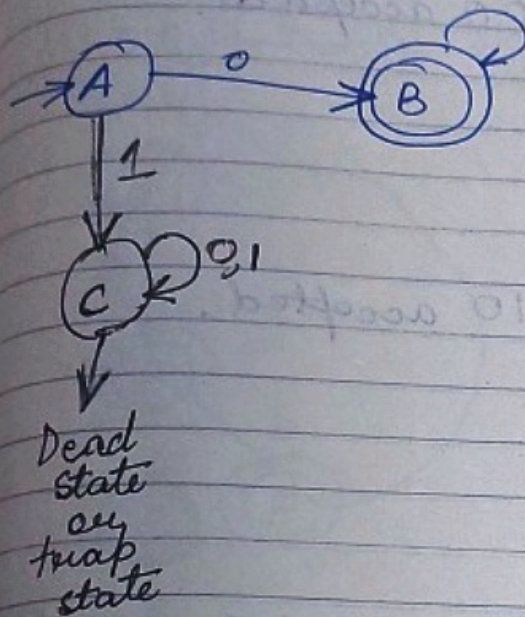
(Σ: sigma)

$Q = \{A, B, C, D\}$
$\Sigma = \{0, 1\}$
$q_0 = A$
$F = \{D\}$

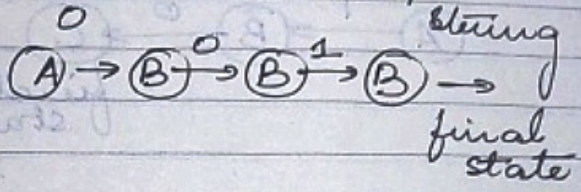
$\delta \rightarrow$

	0	1
A	C	B
B	D	A
C	A	D
D	B	C

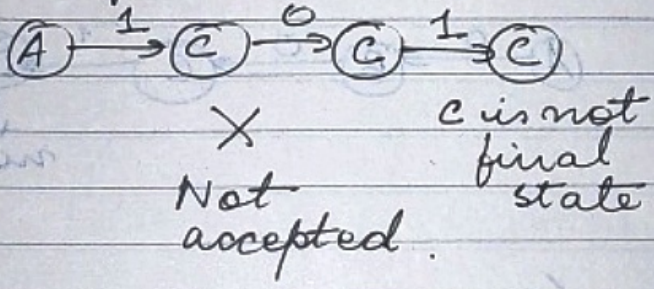
L_1 = set of all strings that start with '0'
= {0, 00, 01, 000, 010, 011, 0000, ... }



E.g :- 001 ✓ Accepted string

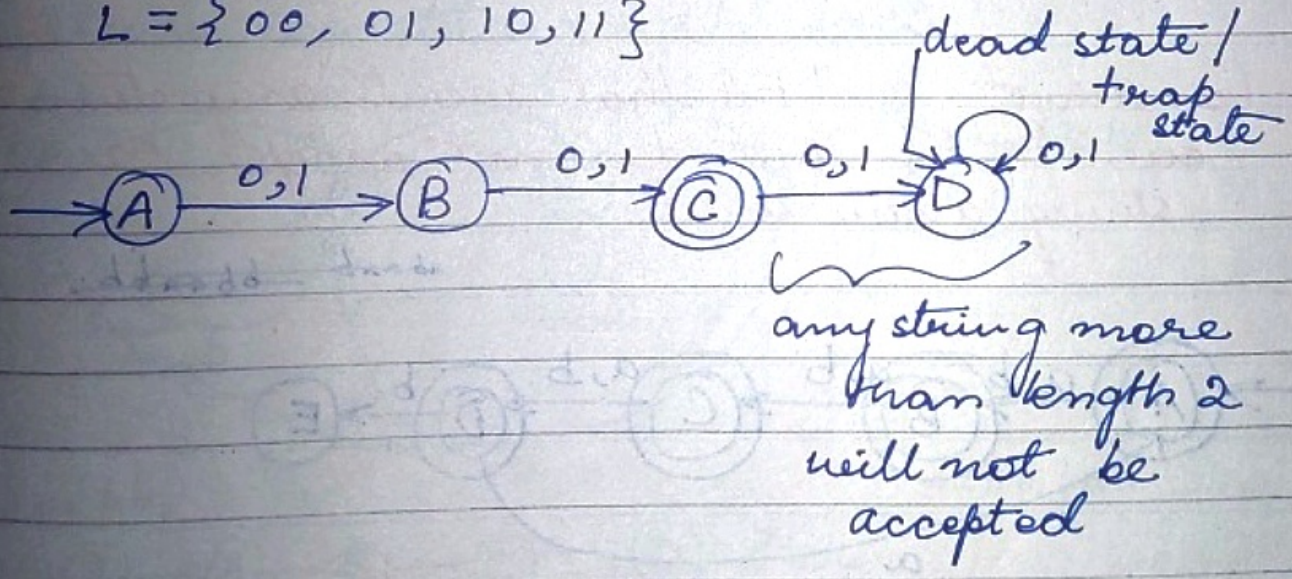


E.g :- 101

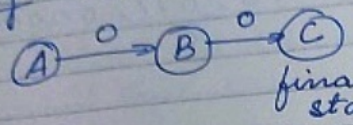


Construct a DFA that accepts all strings over {0, 1} of length 2.

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

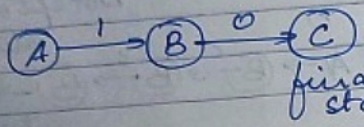


E.g 00



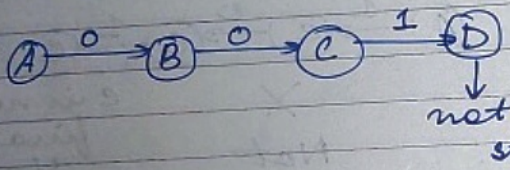
∴ 00 accepted.

E.g 10



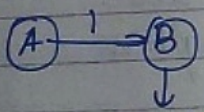
∴ 10 accepted.

E.g 001



∴ string not accepted.

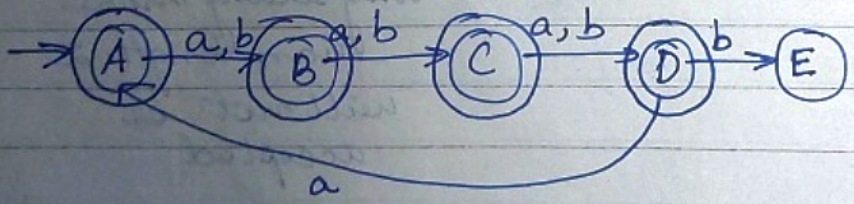
E.g 1



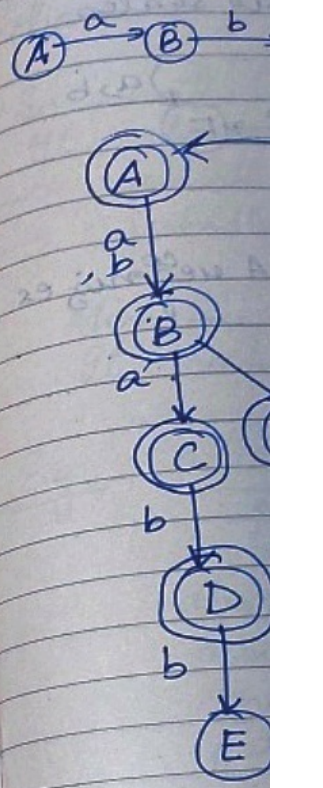
∴ string not accepted.

Construct a DFA that accepts any string over {a, b} that does not accept the string aabb in it.

~~abaab~~ ~~bbaabb~~



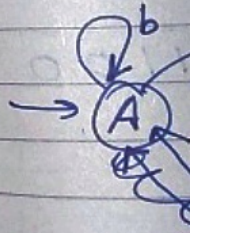
E.g :- a b a



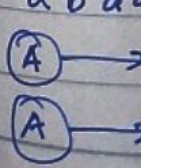
det is d

accept

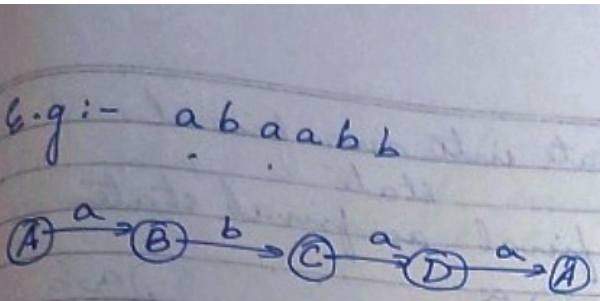
contai



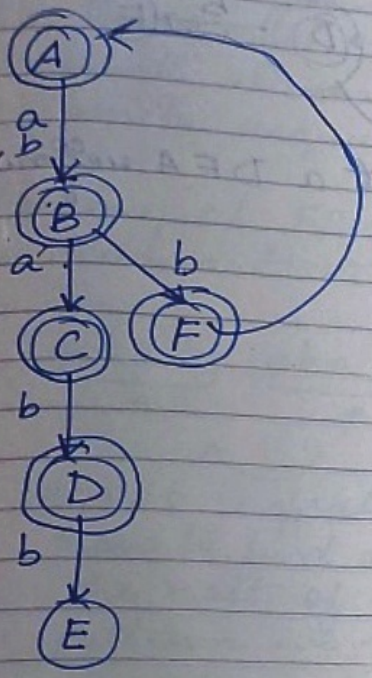
~~abaa~~



E.g:- abaabb



bbaabb



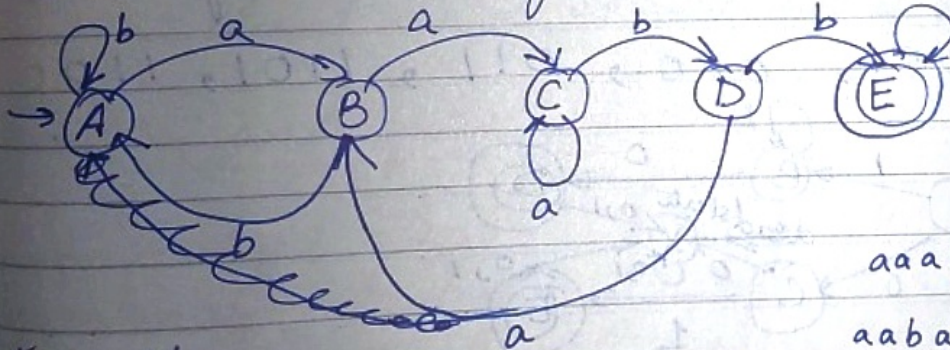
not
epted.

aaab

Let's design a simpler problem.

aaba

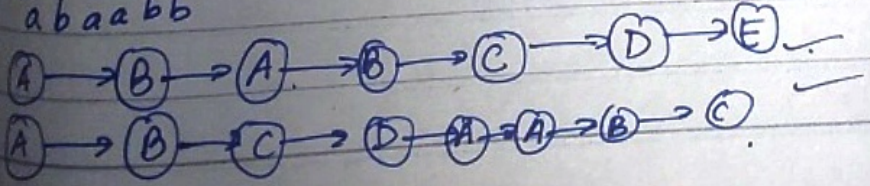
accepts all string over {a,b} that contains the string aabb in it.



aaab

aababaa

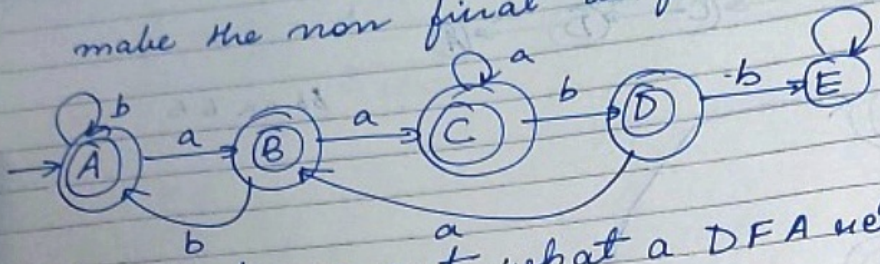
abaabb



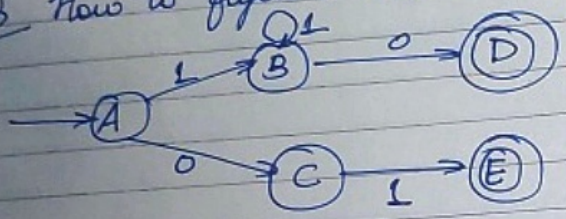
Flip the states

Make the final state into non final state

make the non final as final states.



Q How to figure out what a DFA recognizes.



10 ✓

01 ✓

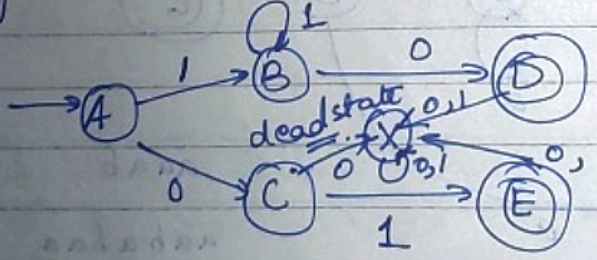
1→1→0 110 ✓

1111...0 ✓

A B D

L = { Accepts the string 01 or a string of at least one '1' followed by a '0' }

E.g 001, 010, 011, 1101, 1100



A language
if and
machine

So what
The lan
→ which
→ which

Memo
It ca

E.g a b

ab
n
H
P
F

E.g

Operati

Union
Conca
STA

Regular languages

A language is said to be a regular lang. if and only if some finite state machine recognizes it.

So what languages are not regular?
The languages
→ which are not recognized by FSM
→ which require memory.

Memory of FSM is very limited.
It cannot store or count strings.

E.g. a b a b b a b a b b

ababb repeats itself
now to find out whether
the rest of the string follows the same pattern we need to store "ababb" which is not possible in FSM ∴ not regular language.

E.g. $a^N b^N$
no of a's should be equal to no of bs.
language not recognized by FSM
count no of as and no of bs.

Equations 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 :- $A^* = \{x_1 x_2 x_3 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$

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

$A \cup B = \{pq, r, t, uv\}$

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

$A^* = \{\epsilon, pq, r, pqr, rpq, pqpq, rrr, pqpqpq, \ldots\}$ ∞ no of elements

Theorem 1:- The class of regular languages is closed under UNION.

Theorem 2:- The class of regular language is closed under CONCATENATION.

NFA — Non Deterministic Finite automata

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.

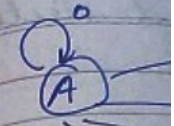
→ It is simple and easy to design.

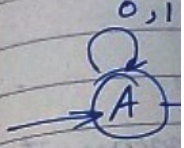
In NFA:-

→ given the current state, there could be multiple next states.

→ the next state may be chosen at random.

→ all the next states may be chosen in parallel. (simultaneously choose all next states)





→ mul

→ we

wh

to

Q, Σ

$Q =$

$\Sigma =$

$q_0 =$

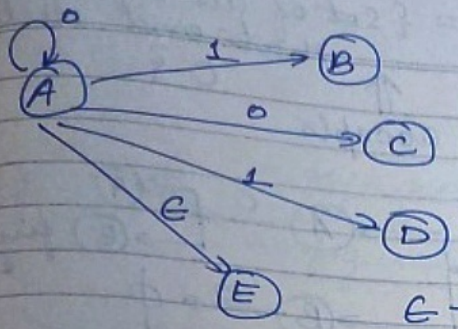
$F =$

$\delta =$

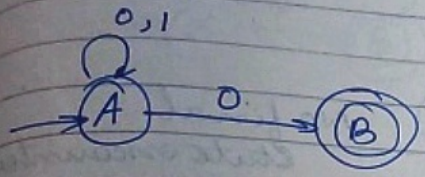
A

A

A



ϵ → Empty input



L = { Set of all strings that end with 0 }

→ multiple next states

→ we didnot mention any input to B which is legal in NFA, i.e any input to it doesnot lead it anywhere.

Q, Σ, q_0, F, δ

Q = Set of all states ⟶ {A, B}
Σ = set of inputs ⟶ {0, 1}
q_0 = initial state ⟶ A
F = final set of states ⟶ {B}
δ = transition function | $Q \times \Sigma \rightarrow 2^Q$

$A \times 0 \rightarrow A$
$A \times 0 \rightarrow B$
$A \times 1 \rightarrow A$
$B \times 1 \rightarrow \phi$
$B \times 0 \rightarrow \phi$

input

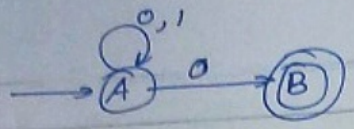
$A \rightarrow A, B, AB, \phi$

3 states A, B, C

$A \xrightarrow{1} A, B, C, AB, AC, BC, ABC, \phi$

3 states → 8 possibilities

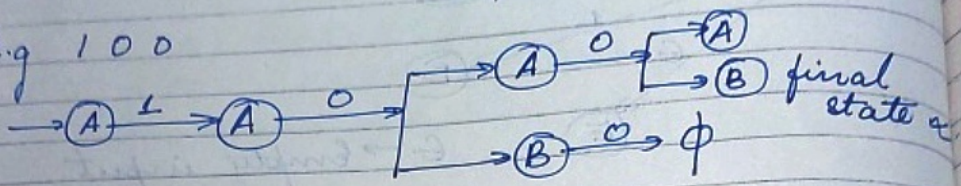
NFA → Example 1



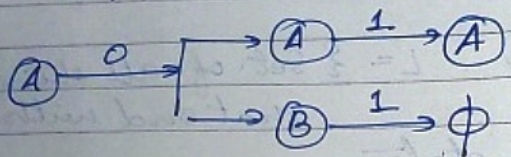
L = { set of all strings that end with 0 }

↑ accepts

E.g 100



E.g:- 01

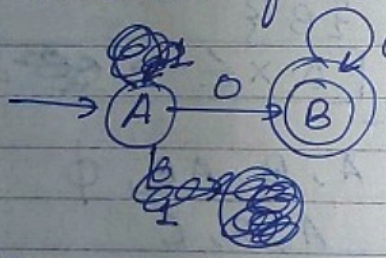


No final state encountered ∴ string not accepted

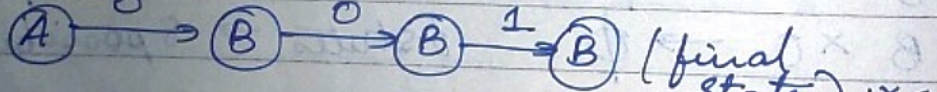
If there is any way to run the (set of states) machine that ends in a way such that at least one state is a final state then the NFA accepts a given input.

NFA - Example 2

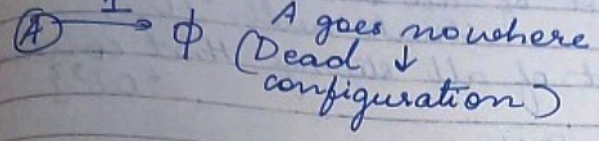
L = { Set of all strings that start with 0 }



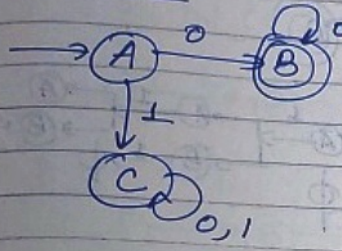
E.g:- 001



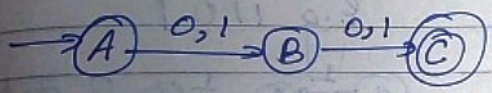
Eg 101



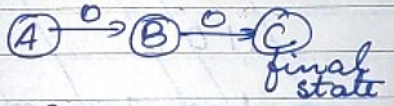
In DFA:-



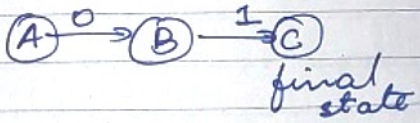
8. Construct a NFA that accepts sets of all strings over {0, 1} of length 2.



Eg 00



Eg 01



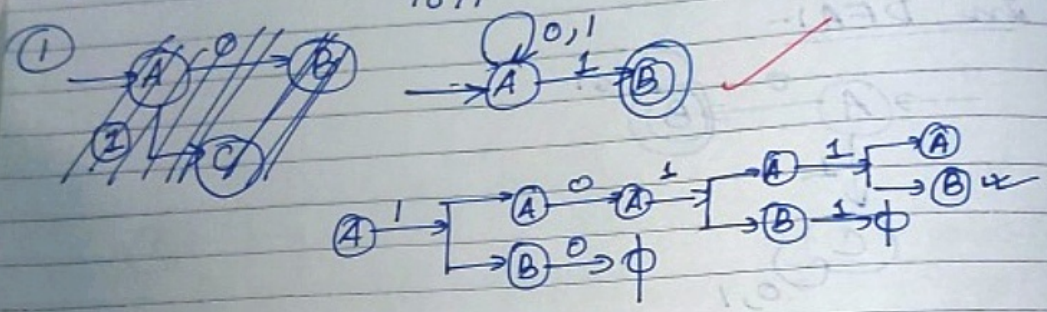
Eg 1:- L1 = { Set of all strings that ends with '1' }

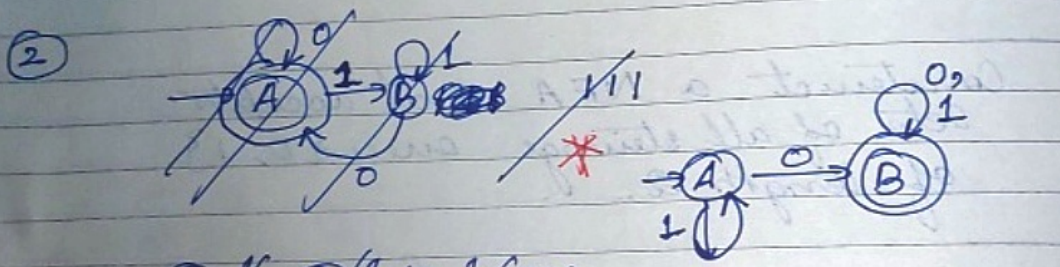
Eg 2:- L2 = { Set of all strings that contain '0' }

Eg 3:- L3 = { Set of all strings that starts with '10' }

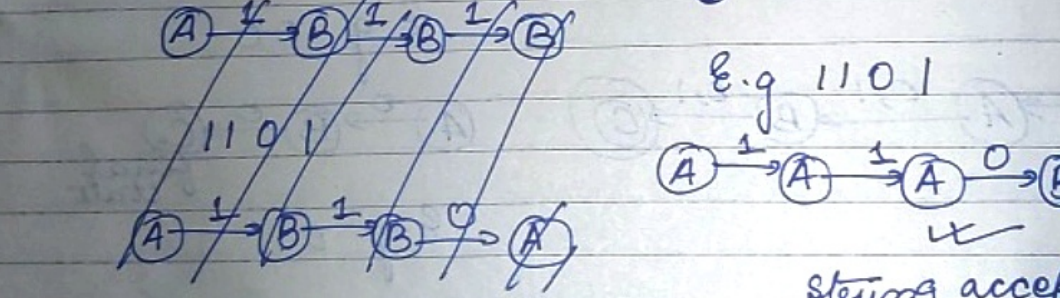
Eg 4:- L4 = { Set of all strings that contains '01' }

1011





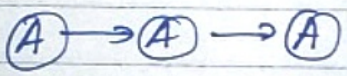
111

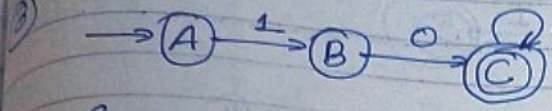


Eg 1101

String accepted

Eg 111

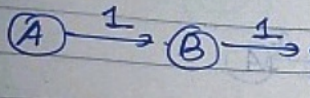




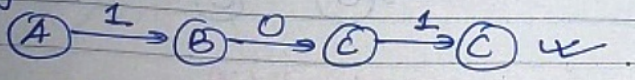
Eg 010

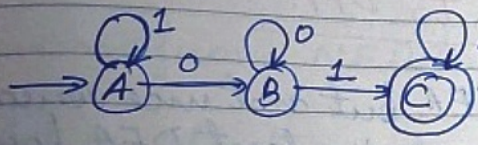
$A \xrightarrow{0} \phi$

Eg 110

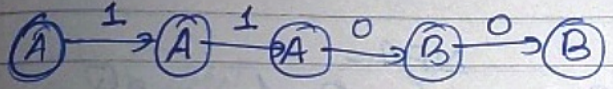


Eg 101

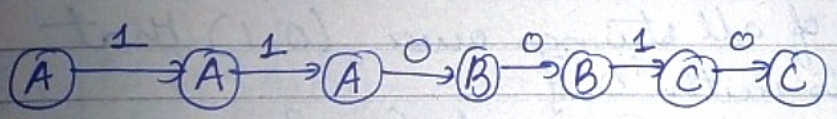




11100

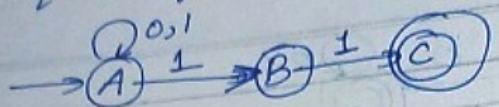


110010

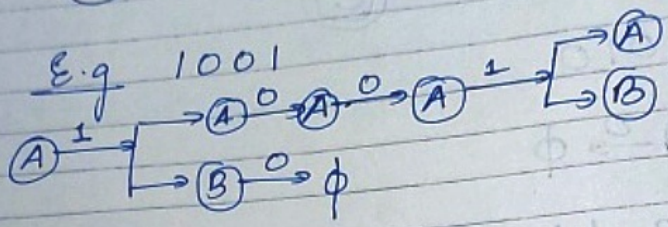


AI ML

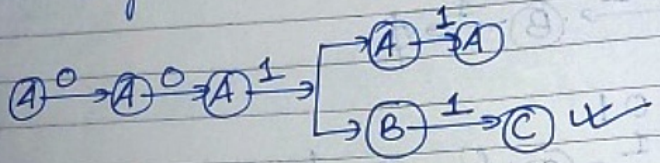
(5) L5 = { Set of all strings that ends with '1' }



E.g. 1001



E.g. 0011



Conversion of NFA to DFA

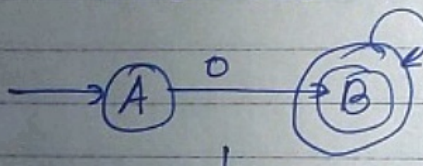
Every DFA is an NFA but not vice versa but there is an equivalent DFA for every NFA.

DFA $\delta \quad Q \times \Sigma \rightarrow Q$

NFA $\delta \quad Q \times \Sigma \rightarrow 2^Q$

L = { Set of all strings over (0,1) that starts with '0' }

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



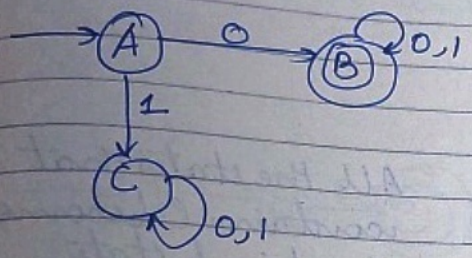
← NFA

	0	1
A	B	φ
B	B	B

DFA

	0	1
A	B	C
B	B	C
C	C	C

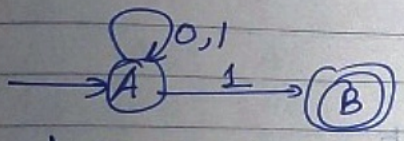
C ← dead state / trap state



Conversion from NFA to DFA

L = {Set of all strings over (0,1) that ends with '1'}

NFA



	0	1
A	{A}	{A, B}
B	ϕ	ϕ

→ we will combine these two states into a single state called AB.

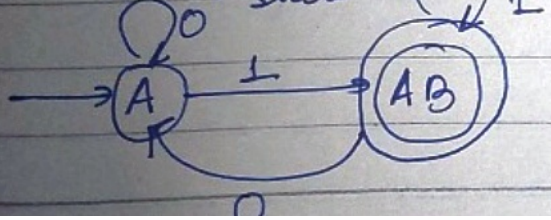
↓ final state

DFA

	0	1
A	{A}	{AB}
AB	{A}	{AB}

we will perform operation

↓ final state



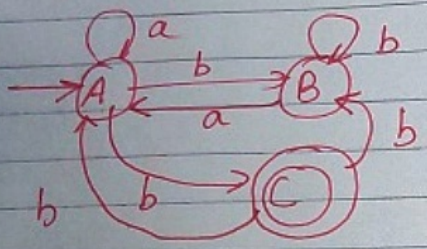
Subset Construction Method
which was used to convert NFA to DFA.

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

string should end with 'ab'.

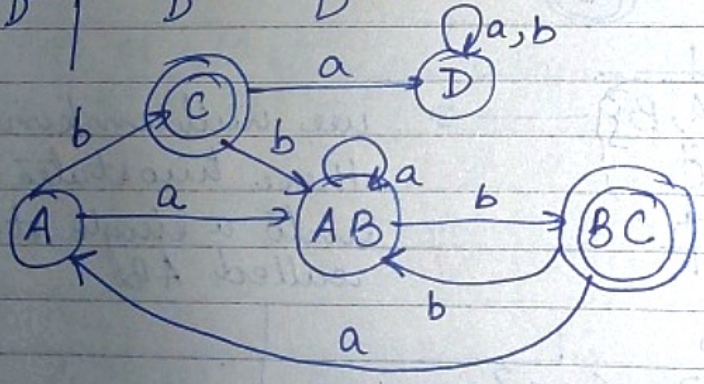
E.g a a b b a b a b

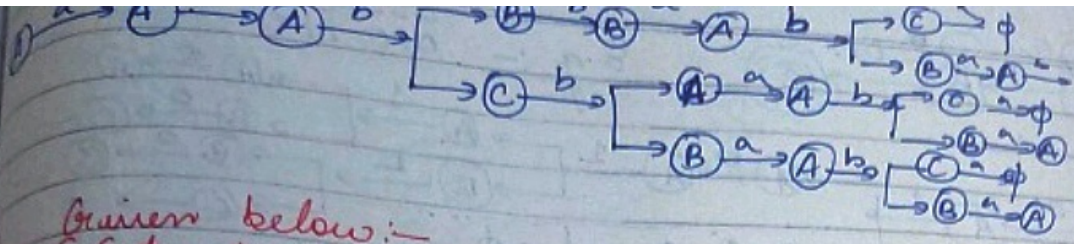


All the states that contain C become the final state.

	a	b
A	{AB}	C
{AB}	{AB}	{BC}
(C)	D	{AB}
{BC}	{A}	{AB}
D	D	D

D → dead state

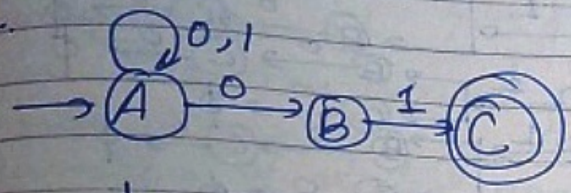




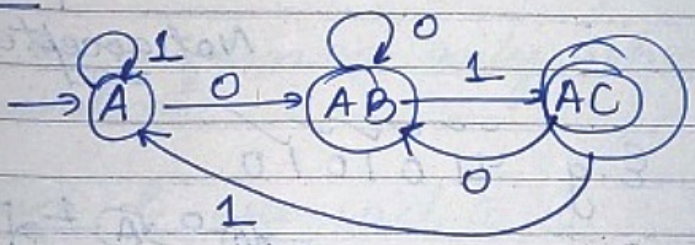
Given below:-

Σ = { Set of all strings over (0,1) that ends with '01' }. Construct its equivalent DFA.

NFA.

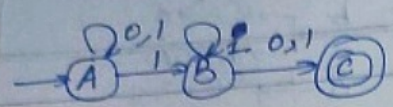


	0	1
→ A	A,B	A
B	ϕ	C
(C)	ϕ	ϕ



	0	1
→ A	{AB}	A
AB	{AB}	{AC}
(AC)	{AB}	{A}

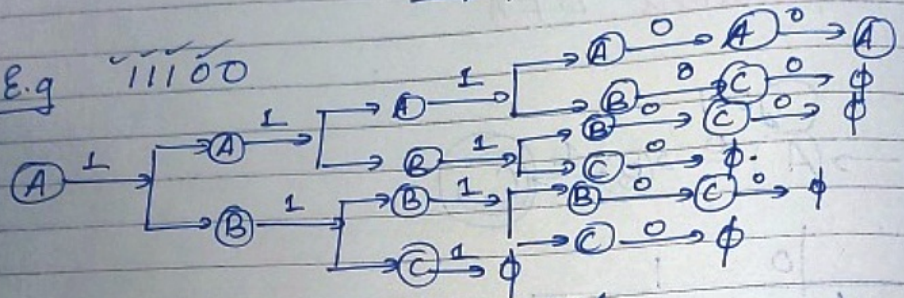
Design an NFA for a language that accepts all strings over {0,1} in which the second last symbol is always '1'. Then convert it to its equivalent DFA.



E.g :- 010110

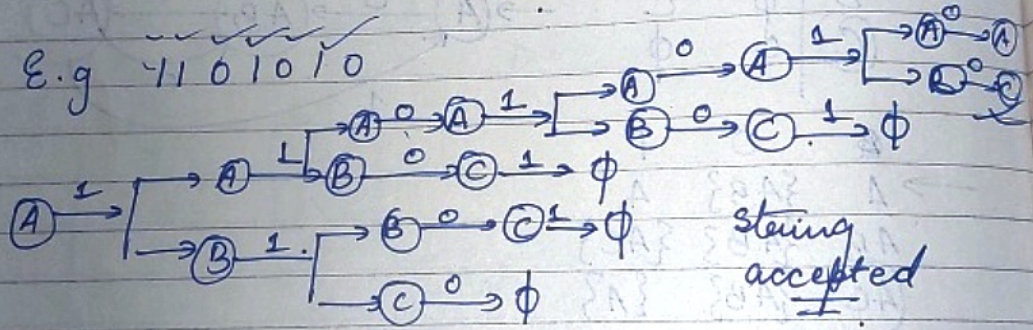
accepted

E.g 11100

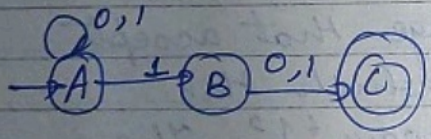


Not accepted.

E.g 1101010

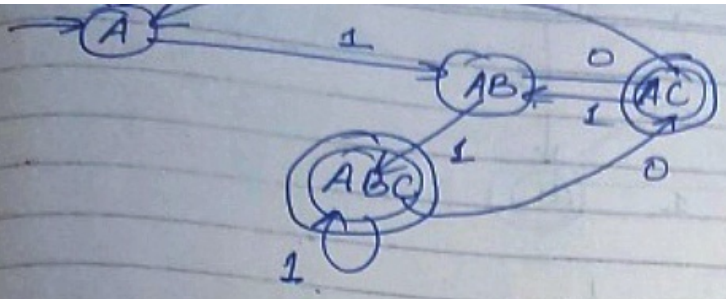


string accepted



	0	1
A	A	{A, B}
B	C	C
C	ϕ	ϕ

	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 number of states possible.

Two states can be combined only when they are equivalent.

Two states A and B are said to be equivalent if :-

$\delta(A, X) \rightarrow F$ and $\delta(B, X) \rightarrow F$

OR

$\delta(A, X) \not\rightarrow F$ and $\delta(B, X) \not\rightarrow F$

where 'X' is any input string

If $|X| = 0$ then A and B are said to be 0 equivalent.

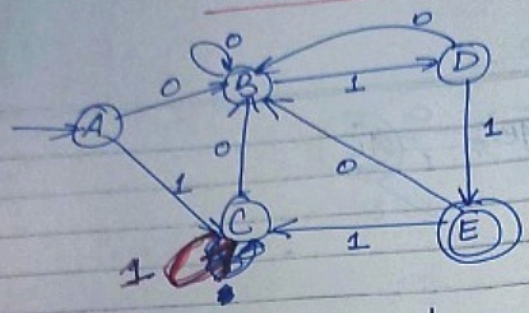
If $|X| = 1$ then A and B are said to be 1 equivalent

:

:

If $|X| = n$ then A and B are said to be n equivalent.

Minimization of DFA — Examples



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

0 Equivalence

make non final states as one set
make final states as another set

{ A, B, C, D } { E }

1 Equivalence

See the states of A and B

$A \xrightarrow{0} B$ ✓ $A \xrightarrow{1} C$ } Both C, D
$B \xrightarrow{0} B$ ✓ $B \xrightarrow{1} D$ } fall into the same set

∴ they are 1 equivalent

Checking C with either A or B.

$A \xrightarrow{0} B$ $A \xrightarrow{1} C$
$C \xrightarrow{0} B$ $C \xrightarrow{1} C$

∴ they are 1 equivalent.

Checking D with A or B or C.

$D \xrightarrow{0} B$
$A \xrightarrow{0} B$

$D \xrightarrow{1} E$
$A \xrightarrow{1} C$ — donot belong to same set

{A,B,C} {D} {E}

2 equivalence — we will check acc to these sets.

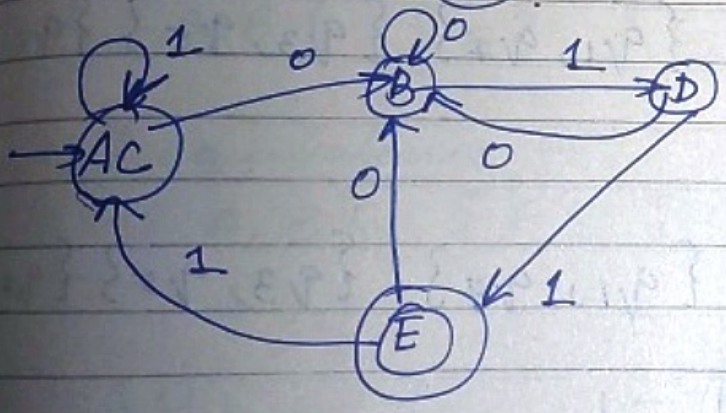
{A,C} {B} {D} {E}

3 equivalence

{A,C} {B} {D} {E}

अगर same हो state आ रहा दोनो का but वो diag set को belong करता then doesn't matter.

$A \xrightarrow{0} B$ $\quad A \xrightarrow{1} C$
$C \xrightarrow{0} B$ $\quad C \xrightarrow{1} C$



	0	1
→ q_0	q_1	q_5 ✓
q_1	q_6	q_2 ✓
q_2	q_0	q_2
q_3	q_7	q_5 ✓
q_4	q_2	q_6
q_5	q_6	q_4 ✓
q_6	q_6	q_2 ✓
q_7		

0 Equivalence

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

1 Equivalence

~~$\{q_0,$ $\{q_1$~~

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

2 equivalence

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

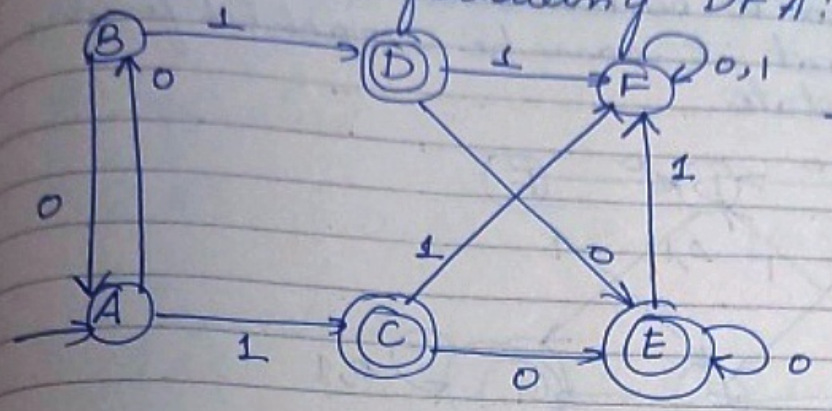
3 equivalence

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

	0	1
→ q_0, q_4	q_1, q_7	q_3, q_5
q_6	q_6	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

final states involved

Minimize the following DFA:-



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

0 equivalences:-

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

1 Equivalence:-

$A \xrightarrow{0} B$, $B \xrightarrow{0} A$ } belongs to same set

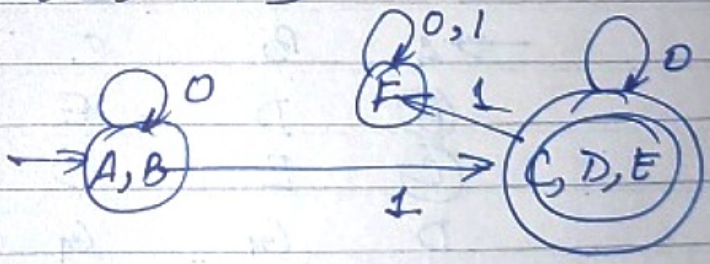
$A \xrightarrow{1} C$, $B \xrightarrow{1} D$ } belongs to same set

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

2 equivalence:-

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

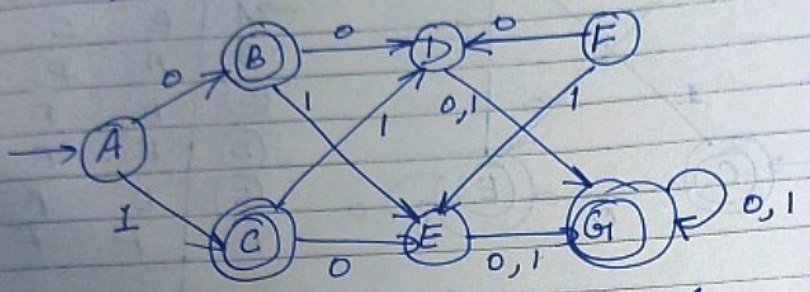
	0	1
→{A, B}	{A, B}	{C, D, E}
{F}	{F}	{F}
{C, D, E}	{C, D, E}	{F}



Minimization of DFA

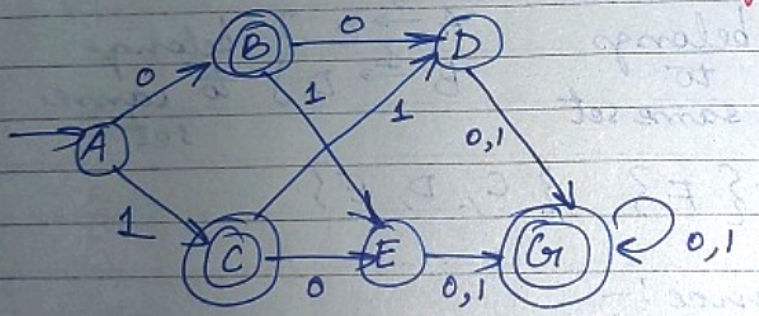
When there are unreachable states involved.

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



Unreachable state will only have outgoing transitions.

So in this :- remove the unreachable state and proceed normally.



	0	1
→ A	B	C
(B)	D	E
(C)	E	D
D	G1	G1
E	G1	G1
(G1)	G1	G1

1 Equivalent

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

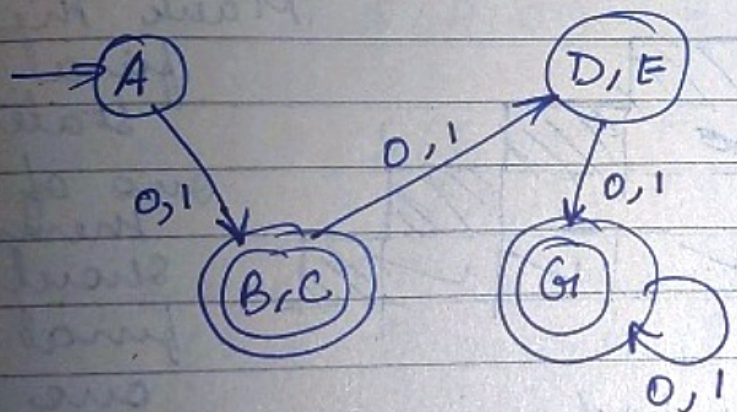
2 Equivalent

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

3 Equivalent

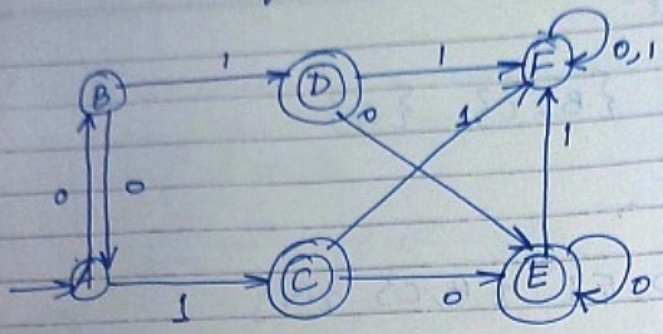
{A} {D, E} {B, C} {G}

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



Minimization of DFA – Table filling method

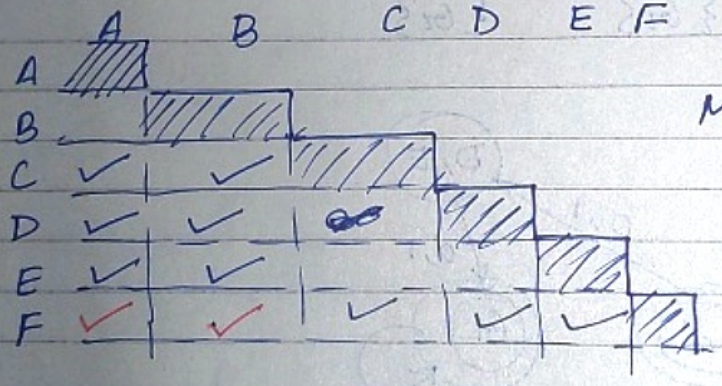
(Myhill – Nerode Theorem)



	A	B	C	D	E	F
A		✓				
B	✓					
C						
D						
E						
F						

There are two cells for A, B

Therefore we will divide the table diagonally.



Mark the final states one of them should be final, one non final

Check the unmarked pairs

(B, A) – $\delta(B, 0) = A$, $\delta(A, 0) = B$ } we will check if AB is marked in the table

$\delta(B, 1) = D$, $\delta(A, 1) = C$ } Both unmarked so we don't have to do anything

(D, C) – $\delta(D,0) = E$, $\delta(C,0) = E$ } Not marked

$\delta(D,1) = F$, $\delta(C,1) = F$ } Not marked.

(E, C) – $\delta(E,0) = E$, $\delta(C,0) = E$ } Not marked

$\delta(E,1) = F$, $\delta(C,1) = F$ } Not marked

(E, D) – $\delta(E,0) = E$, $\delta(D,0) = E$ } Not marked

$\delta(E,1) = F$, $\delta(D,1) = F$ } Not marked

(F, A) – $\delta(F,0) = F$, $\delta(A,0) = B$ } Not marked

$\delta(F,1) = F$, $\delta(A,1) = C$ } marked.

Since (F, C) is marked we will have to mark (F, A)

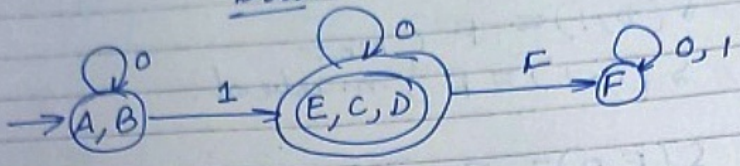
(F, B) – $\delta(F,0) = F$, $\delta(B,0) = A$ } marked $\quad \delta(F,1) = F$, $\delta(B,1) = D$

Repeat the same steps until no more markings can be made.

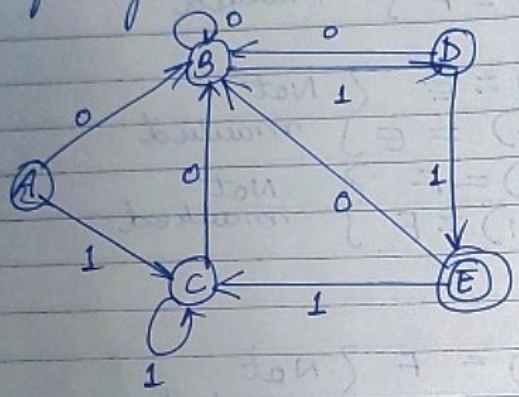
Combine all the unmarked pairs and make them a single state in minimized DFA.

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

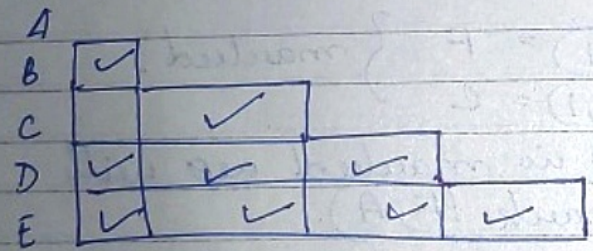
Ans:-



Minimize the following DFA using table filling method.



A B C D E



AB

$\delta(A,0) = B$ $\delta(A,1) = C$ } unmarked

$\delta(B,0) = B$ $\delta(B,1) = D$ }

unmarked

AC ✗ CB ✗

DA ✓ DB ✓ CD ✓

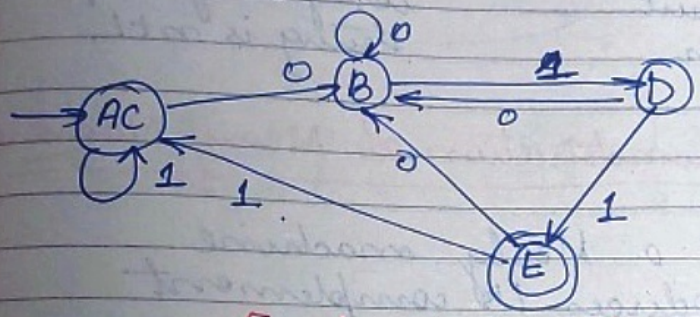
Repeating same steps.

AB ✓

$\delta(C,0) = B$ $\quad \delta(A,0) = B$
$\delta(C,1) = C$ $\quad \delta(A,1) = C$.

$\delta(C,0) = B$ $\quad \delta(B,0) = B$
$\delta(C,1) = C$ $\quad \delta(B,1) = D$

No more marking can be done
Combine unmarked pairs (A,C)



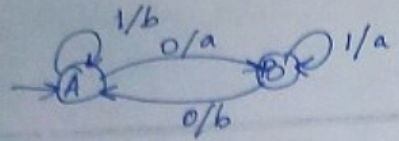
Finite Automata with Outputs

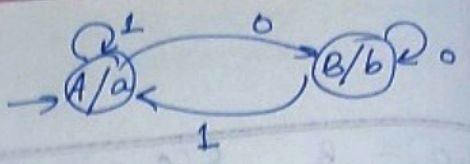
Mealy machine

$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$
where
Q = finite set of states
Σ = finite non empty set of input alphabets
Δ = set of output alphabets
δ = transition function
$Q \times \Sigma \rightarrow Q$
λ = output function
$\Sigma \times Q \rightarrow \Delta$
q_0 = Initial state / start ~~final~~ state

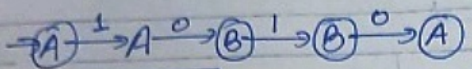
Moore machine

$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$
where,
Q = finite set of states
Σ = finite non empty set of input alphabets
Δ = set of output alphabets
δ = transition function
$Q \times \Sigma \rightarrow Q$
λ = output function
$Q \rightarrow \Delta$
q_0 = Initial / start state





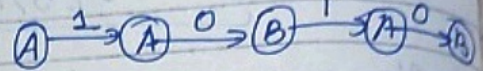
E.g 1010



baab

If length of input string is n then length of output string is n.

E.g 1010

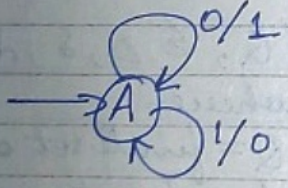


aabab

If length of input string is n then length of output string is n+1.

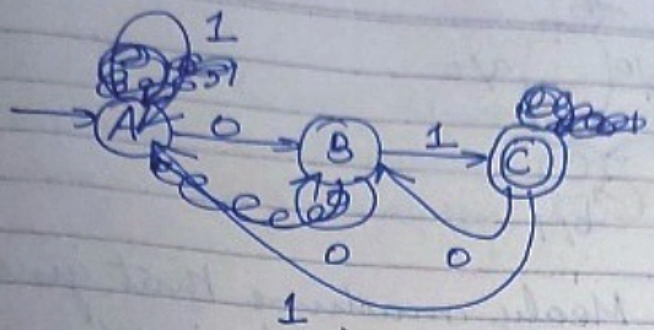
Construction of Mealy Machine

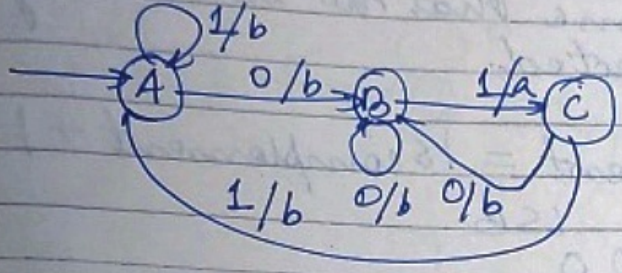
E.g 1 Construct a Mealy machine that produces 1's complement of any binary input string.



10100
01011

E.g 2 Construct a Mealy machine that prints 'a' whenever the sequence '01' is encountered in any input binary string.

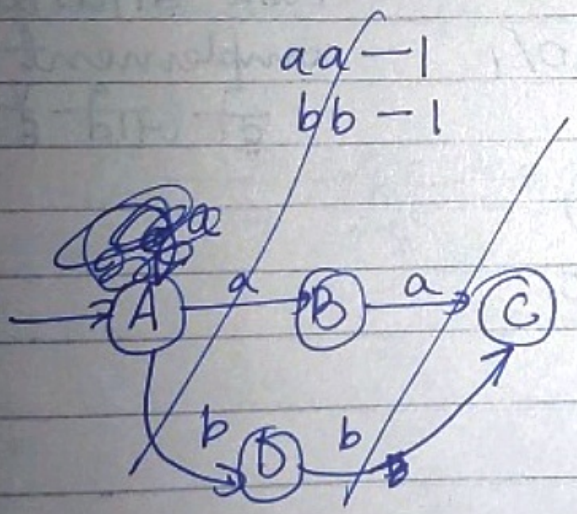


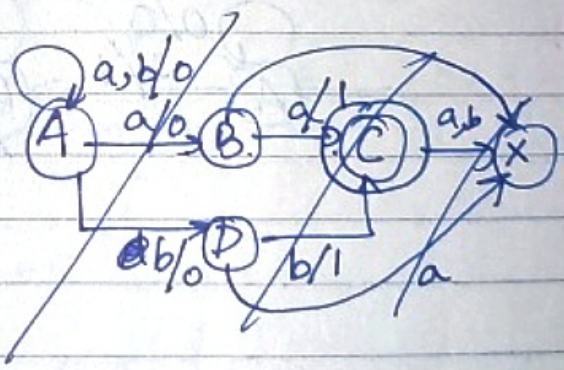


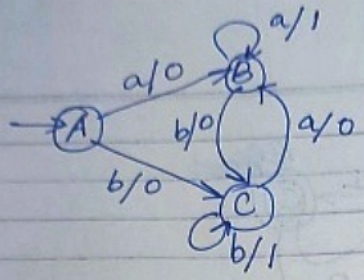
E.g. 0 1 1 0
b a b b

E.g. 1 0 0 0
b b b b

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







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

2's complement = 1's complement + 1

MSB LSB

E.g 1 0 1 0 0

~~0 1 0 0 0~~

0 1 0 1 1

1

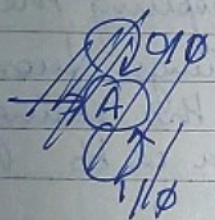
0 1 1 0 0

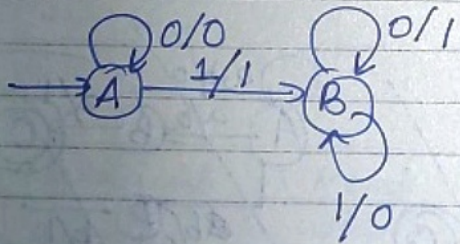
1 1 1 0 0

0 0 0 1 1

1

0 0 1 0 0



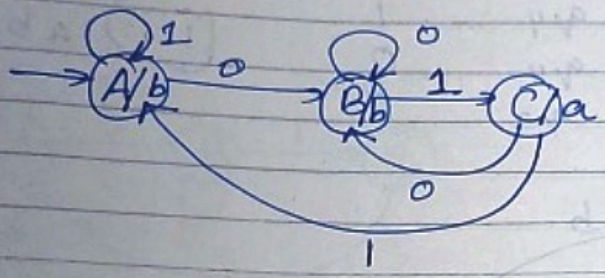


LSB से MSB में पहले '1' के encounter होने तक सब same होता है फिर बाकि digits complement हो जाते हैं।

Construction of Moore Machine

Construct a Moore Machine that prints 'a' whenever the sequence '01' is encountered in any input binary string.

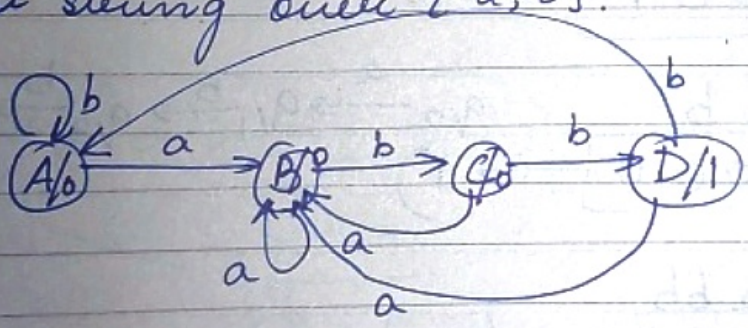
$\Sigma = \{0, 1\}$
$\Delta = \{a, b\}$



E.g :- 0 1 1 0 $\quad A \xrightarrow{0} B \xrightarrow{1} C \xrightarrow{1} A \xrightarrow{0} B$
b b a b b

E.g :- 0 1 0 1 $\quad A \xrightarrow{0} B \xrightarrow{1} C \xrightarrow{0} B \xrightarrow{1} C$
b b a b a

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



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

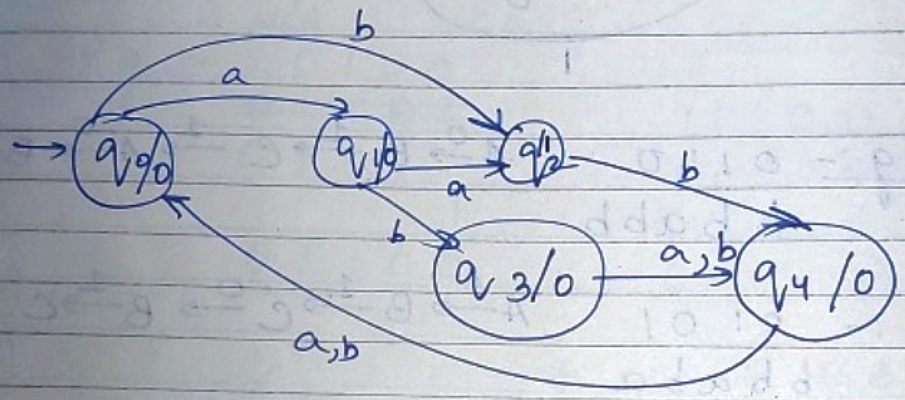
For the following Moore Machine the input alphabet is $\Sigma = \{a, b\}$ and the output alphabet is $\Delta = \{0, 1\}$. Run the following input sequences and find the respective outputs:-

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

(ii) a b b b

(iii) a b a b b.



a a b a b

$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{b} q_4 \xrightarrow{a} q_0 \xrightarrow{b} q_2$

0 0 1 0 0 1 ——(i)

a b b b

$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_3 \xrightarrow{b} q_4 \xrightarrow{b} q_0$

0 0 0 0 0 ——(ii)

a b a b b

0 0 0 0 0 1 ——(iii)

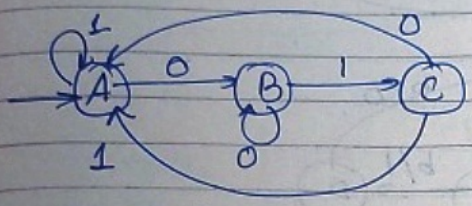
$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_3 \xrightarrow{a} q_4 \xrightarrow{b} q_0 \xrightarrow{b} q_2$

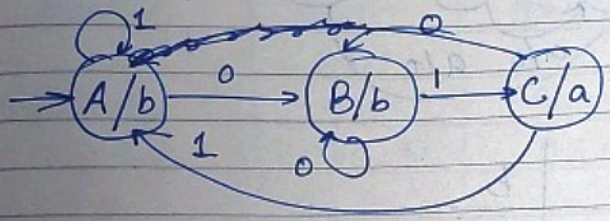
Conversion of Moore Machine to Mealy Machine

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

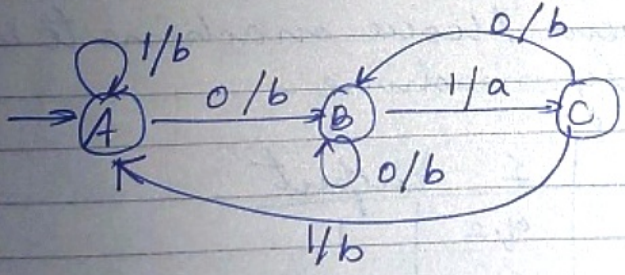
Moore Machine ⟺ Mealy machine

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





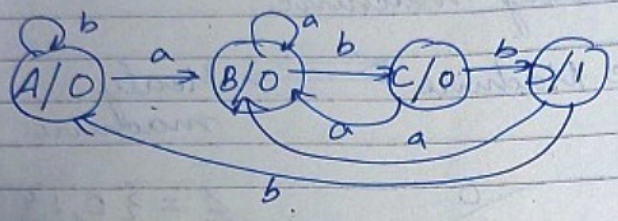
State	0	1	output
→ A	B	A	b
B	B	C	b
C	B	A	a

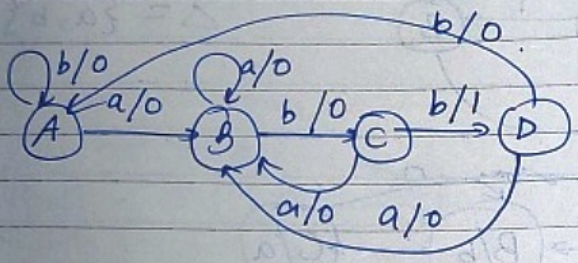


State	0	1
A	B, b	A, b
B	B, b	C, a
C	B, b	A, b

The given Moore machine counts the occurences of the sequence 'abb' in any input binary string over $\{a, b\}$. Convert it to its equivalent mealy machine.

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

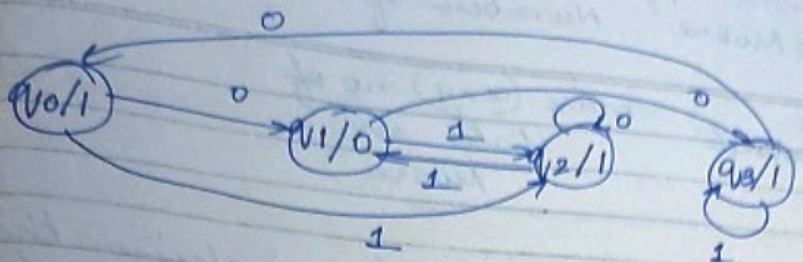


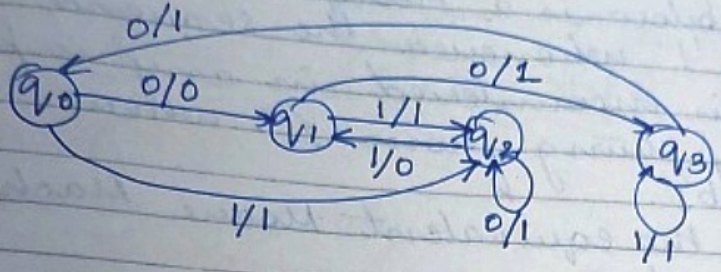


State	a	b
A	B, 0	A, 0
B	B, 0	C, 0
C	B, 0	D, 1
D	B, 0	A, 0.

Convert the given Moore machine to its equivalent Mealy machine

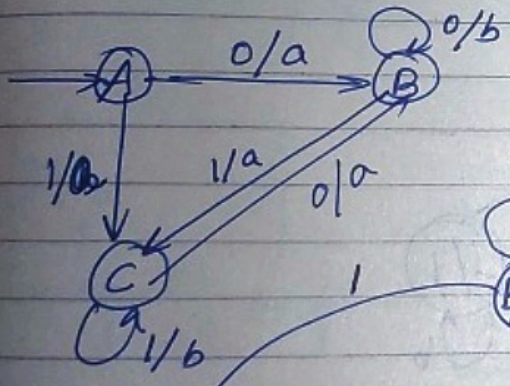
state	0	1	output
$\rightarrow q_0$	q_1	q_2	1
q_1	q_3	q_2	0
q_2	q_2	q_1	1
q_3	q_0	q_3	1



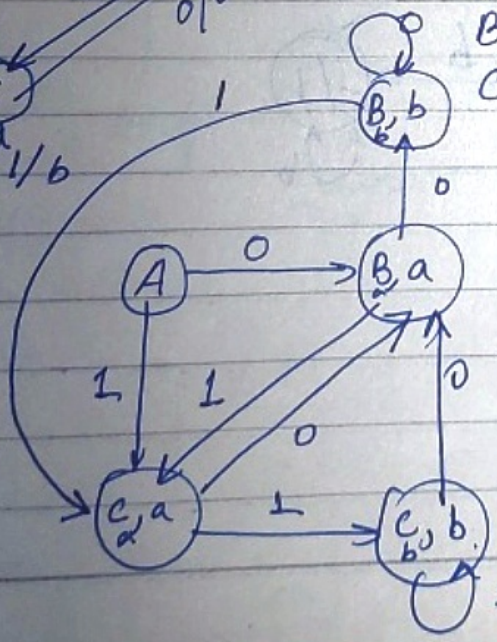


States	0	1
→ q_0	q_1, 0	q_2, 1
q_1	q_3, 1	q_2, 1
q_2	q_2, 1	q_1, 0
q_3	q_0, 0	q_3, 1

Conversion of Mealy machine to Moore Machine



States	0	1
A	B, a	C, a
B	B, b	C, a
C	B, a	C, b



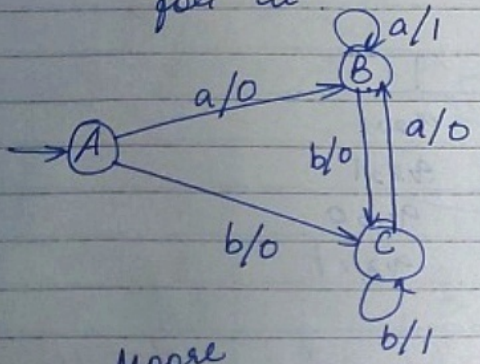
Moore → Mealy: Number of states were same

Mealy → Moore: Number of states increased.

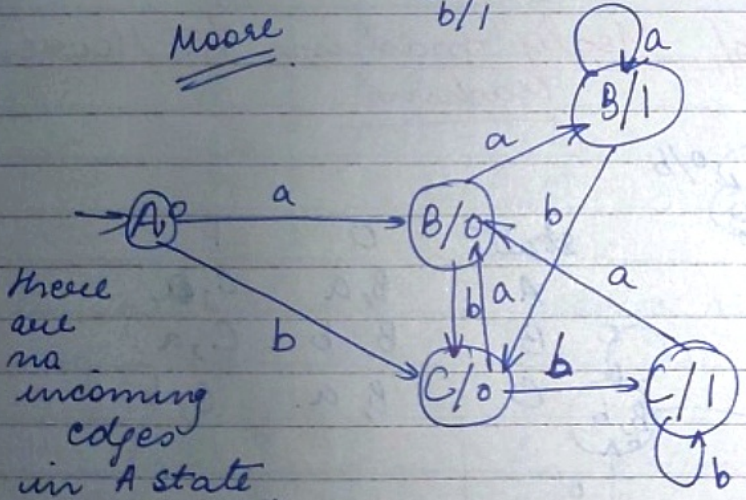
↓

x states × y outputs — max (x×y) no of states in Moore.

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



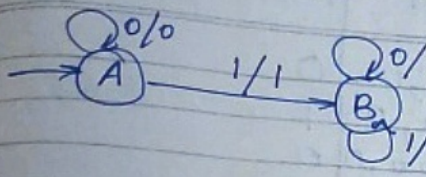
Moore



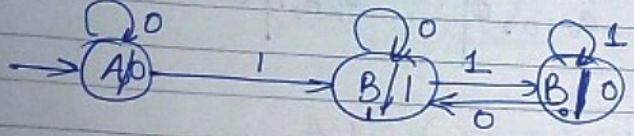
There are no incoming edges in A state.

Convert

Convert the given Mealy machine to Moore machine.



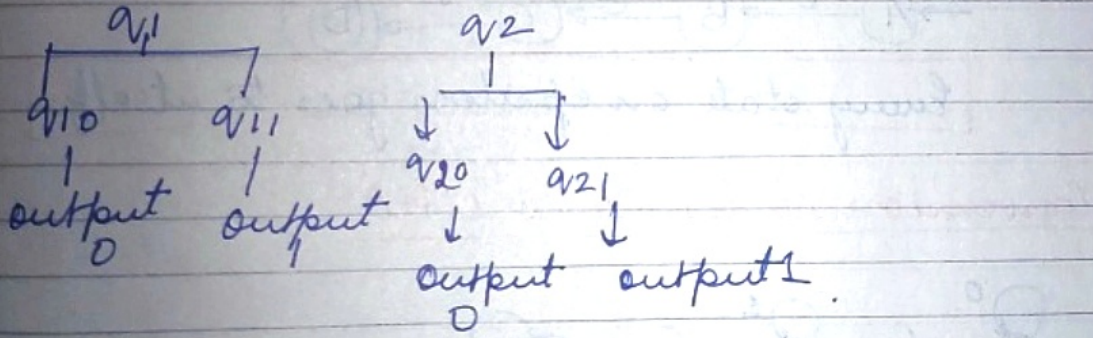
$\Sigma = \{0, 1\}$ $\Delta = \{0, 1\}$



Conversion of Mealy machine to Moore machine using Transition table.

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	$q_1, 0$	$q_0, 1$

Whenever we find a state giving two different kinds of output we split the state.



state	a	b	output
→ q_0	$q_3, 0$	$q_1, 1$	1
q_{10}	$q_0, 1$	$q_3, 0$	0
q_{11}	$q_0, 1$	$q_3, 0$	1
q_{20}	$q_{21}, 1$	$q_{20}, 0$	0
q_{21}	$q_{21}, 1$	$q_{20}, 0$	1
q_3	$q_{10}, 0$	$q_0, 1$	0

Epsilon (ϵ) NFA

ϵ – NFA

Empty symbol

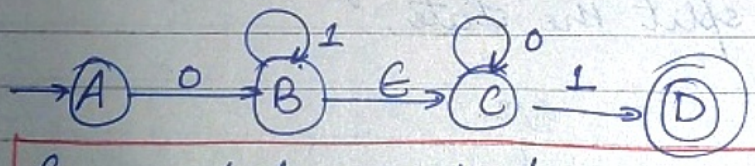
$\{Q, \Sigma, q_0, \delta, F\}$

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

Regular NFA

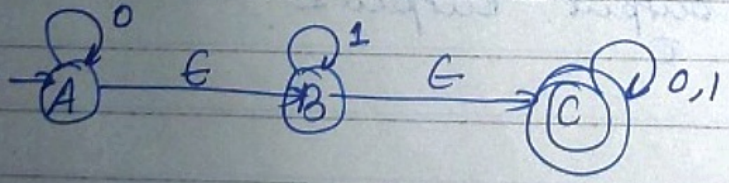
$Q \times \Sigma \cup \epsilon \rightarrow 2^Q$

The state on recieving no input can also go to some other state.



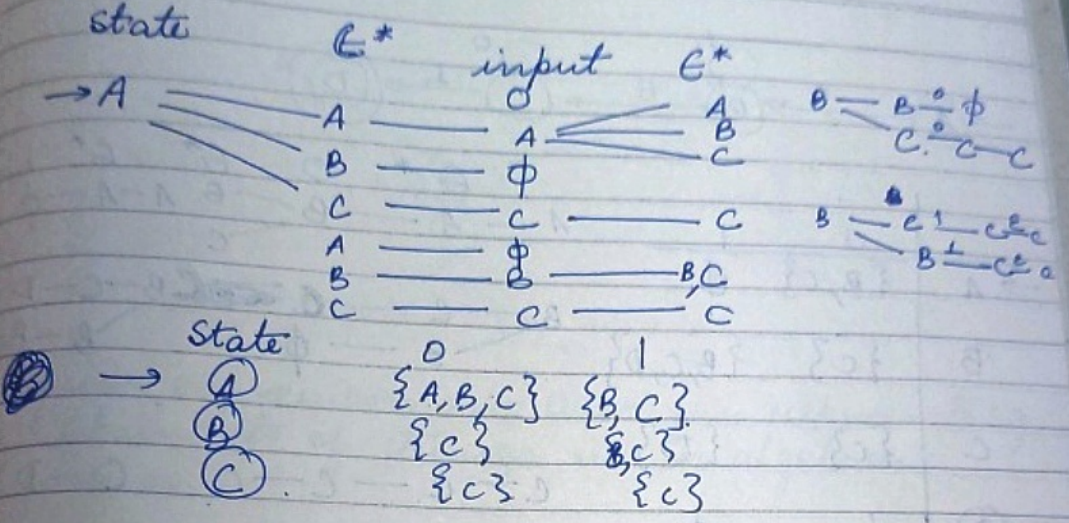
Every state on epsilon goes to itself.

Conversion of ϵ NFA to NFA

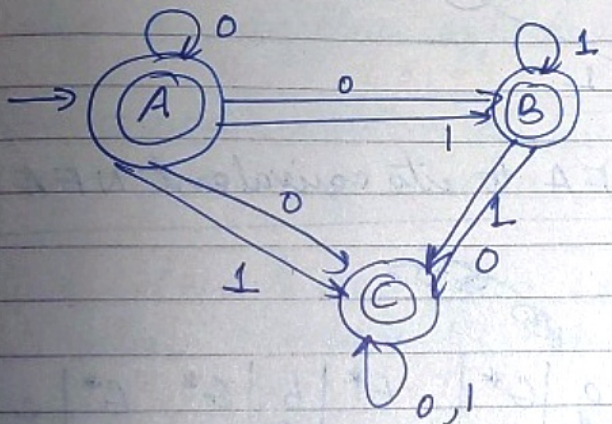


(ϵ*)

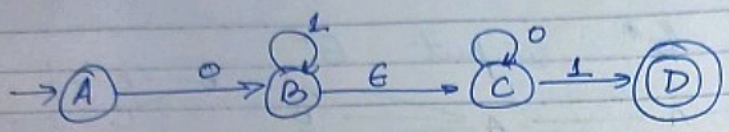
ϵ closure :- All the states that can be reached from a particular state only by seeing the ϵ symbol.



Any state that reaches the final state only on seeing epsilon is final state.



Convert the following ϵ-NFA to its equivalent NFA.



	0	1
→ A	{B, C}	ϕ
B	{C}	{B, C, D}
C	{C}	{D}
(D)	ϕ	ϕ

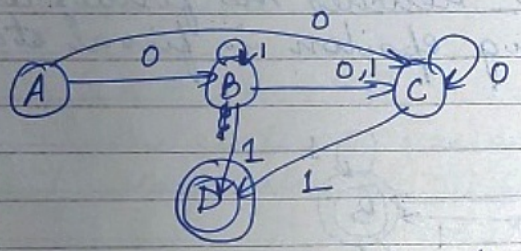
ϵ* 0 ϵ* ϵ* 1 ϵ*

A — A — B — B A — A — ϕ
C

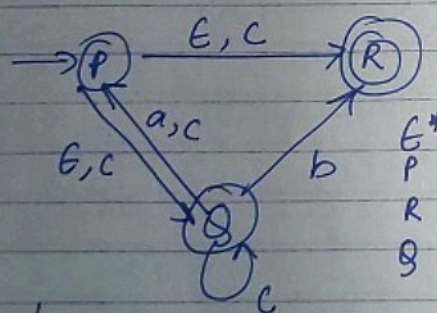
B — C — C — B — C — D — D
B — ϕ — B — B

C — C — C — C C D D

D ϕ



Conversion of ϵ-NFA to its equivalent NFA



	a	b	c
→ (P)	{P, Q, R}	{Q, R}	{Q, R}
Q	{P, Q, R}	{R}	{Q}
(R)	ϕ	ϕ	ϕ

ϵ*	a	ϵ*
P	ϕ	
R	ϕ	
Q	P	P R Q
Q	P	P Q R
R	ϕ	ϕ

ϵ*	b	ϵ*	ϵ*	c	ϵ*
P	ϕ		P	R	R
R	ϕ		R	ϕ	
Q	R	R	Q	Q	Q
Q	R	R	Q	Q	Q
R	ϕ	ϕ	R	ϕ	ϕ

Regular ex
certain
fashion

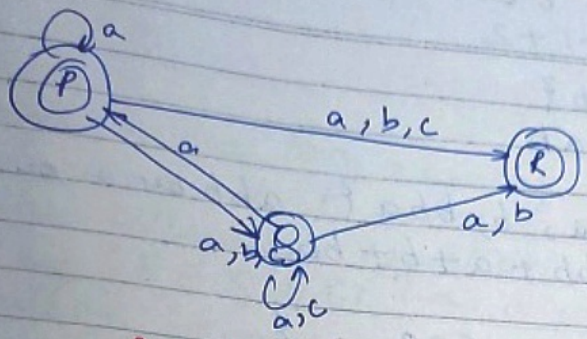
① Any te
middle di
a,

② The u
also
R

③ The co
expre

④ The
is a

⑤ The
preci
by H
rule



Regular Expression

Regular expressions are used for representing certain sets of strings in an algebraic fashion.

1. Any terminal symbol, i.e symbols $\in \Sigma$ including $\wedge$ and ϕ are regular expressions.
   $a, b, c \ldots, \wedge, \phi$ — empty symbol
2. The Union of two regular expressions is also a regular expression.
   $R_1, R_2 \quad (R_1 + R_2)$
3. The concatenation of two regular expressions is also a regular expression.
   $R_1, R_2 \quad (R_1 \cdot R_2)$
4. The iteration (closure) of a regular expression is also a regular expression.
   $R \rightarrow R^* \quad a^* = \wedge, a, aa, aaa, \ldots$
5. The regular expression over Σ are precisely those obtained recursively by the application of the above rules once or several times.

Regular Expression – Examples

① $\{0, 1, 2\}$ 0 or 1 or 2
$R = 0 + 1 + 2$

② $\{\wedge, ab\}$
$R = \wedge ab$

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

④ $\{\wedge, 0, 00, 000, \ldots\}$
any string that can be formed using '0'.
closure of 0
$R = 0^*$

⑤ $\{1, 11, 111, 1111, \ldots\}$
$R = 1^+$
closure of the symbol excluding the empty symbol.

Identities of Regular Expression

1) $\phi + R = R$ ($+ \rightarrow$ union)
2) $\phi R + R\phi = \phi$ (ϕ ↓ concat, union ↑)
3) $\in R = R\in = R$ $(\in = \wedge)$
4) $\in^* = \in$ and $\phi^* = \in$
5) $R + R = R$
6) $R^* R^* = R^*$
closure of R and concatenate them.

7) $RR^* = R^* R$
8) $(R^*)^* = R^*$
9) $\in + RR^* = \in + R^* R = R^*$
just remember $R^+ \rightarrow R^*$
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 doesn't contain ϵ, then the following equation in R given by $R = Q + RP$ has a unique solution :-

$$\boxed{R = QP^*}$$

$$\begin{aligned} R &= Q + RP \quad \text{---} \quad (1) \\ &= Q + QP^*P \\ &= Q(\epsilon + P^*P) \quad (\text{from identity } \epsilon + R^*R = R^*) \\ &= QP^* \end{aligned}$$

(Hence proved)

$$\begin{aligned} R &= Q + RP \\ &= Q + (Q+RP)P \\ &= Q + QP + RP^2 \\ &= Q + QP + (Q+RP)P^2 \\ &= Q + QP + QP^2 + RP^3 \\ &= Q + QP + QP^2 + \cdots + QP^n + \cdots RP^{n+1} \end{aligned}$$

$R = QP^*$

$$\begin{aligned} &= Q + QP + QP^2 + \cdots + QP^n + QP^*P^{n+1} \\ &= Q(\epsilon + P + P^2 + \cdots + P^n + P^*P^{n+1}) \\ &= QP^* \end{aligned}$$

P^* represents the closure of P.

(Hence proved)

An Example Proof using Identities of Regular Expressions

$(1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1)$
is equal to $0^*1\,(0+10^*1)^*$

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

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

$\epsilon + R^*R = R^*$

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

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

$\epsilon R = R$

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

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

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

Designing Regular Expressions

Designing Regular Expression for the following languages over $\{a, b\}$

① Language accepting strings of length exactly 2.

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

$R = aa + ab + ba + bb$

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

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

if we want a string of length exactly 3.

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

if we want a string of length exactly 4.

$(a+b)(a+b)(a+b)(a+b)$.

2) Language accepting strings of length at least 2.

$L_1 = \{aa, ab, ba, bb, aaa, \dots\}$

$R = \underbrace{(a+b)(a+b)}_{2}\underbrace{(a+b)^*}_{\text{or more than 2}}$

अगर एक string चाहिए of length exactly 3 then replace * with 1.

4 length ⟶ " " " 2.

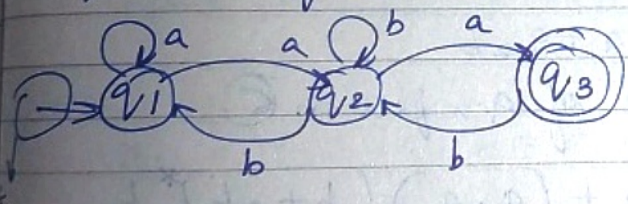
3) $L_1 = \{\Lambda, a, b, aa, ab, ba, bb\}$

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

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

Designing Regular Expression

find the regular expression for the following NFA.



Check all the incoming transitions.

$q_3 = q_2 a$ — (1)

$q_2 = q_1 a + q_3 b + q_2 b$ — (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 from eq ②)

$= q_1 aa + q_2 ba + q_3 ba$ —— ④

② $q_2 = q_1 a + q_2 b + q_3 b$

Putting value of q_3 from eq ①

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

$= q_1 a + q_2 b + q_2 ab$

$q_2 = \underset{R}{} \quad \underbrace{q_1 a}_{Q} + \underbrace{q_2 (b + ab)}_{R \quad P}$

$R = Q + RP.$

$R = QP^*$ by Arden's Theorem

$q_2 = (q_1 a)(b + ab)^*$ —— ⑤

③ $q_1 = \epsilon + q_1 a + q_2 b$

Putting value of q_2 from ⑤

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

$R = \underset{Q}{\epsilon} + \underset{R}{q_1} \underbrace{(a + a(b + ab)^* b)}_{P}$

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

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

Final st

$q_3 =$

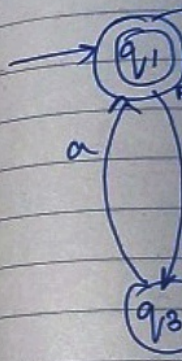
$=$

$= ((a$

$=$

Find

follo

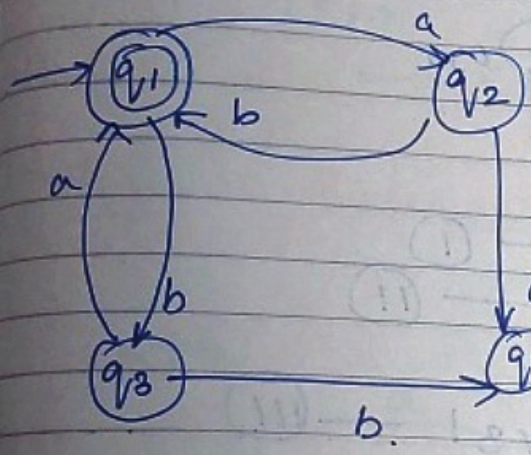


$= (q_1 a)(b + ab)^* a$ (from eq 5)

$= ((a + a(b+ab)^*) b^*) a (b+ab)^* a$ from eq 6.

= req. expression from NFA.

Find the Regular Expression for the following DFA:-



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

$q_2 = q_1 a$ — (ii)

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

$q_3 = q_1 b$ — (iii)

Take the final state and start simplifying it:

$q_1 = \epsilon + q_2 b + q_3 ba$

Putting values of q_2 and q_3 from eq (ii) & (iii)

$q_1 = \epsilon + (q_1 a) b + (q_1 b) ba$

$q_1 = \epsilon + q_1 (ab + ba)$

$q_1 \quad \epsilon \quad (ab+ba)$

$R \quad Q \quad P$

$R = QP^*$ By arden's theorem

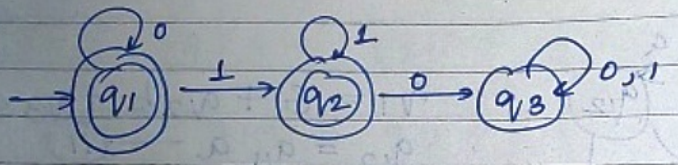
$q_1 = \epsilon (ab+ba)^*$ $\epsilon R = R$

$\therefore q_1 = (ab + ba)^*$

↑ Required Regular Expression.

Designing Regular Expression where there are multiple states

We have to find the regular expression for all the final states and then take their union.



$q_1 = \epsilon + q_1 0$ — (I)

$q_2 = q_1 1 + q_2 1$ — (II)

$q_3 = q_2 0 + q_3 0 + q_3 1$ — (III)

$q_1 = \epsilon + q_1 0$

$R = Q + RP$

$R = QP^*$ By Arden's Theorem

$= \epsilon 0^*$

$\epsilon R = R$

$\therefore R = 0^*$

$\boxed{q_1 = 0^*}$

$q_2 = q_{11} + q_{21}$

$R = Q + RP$

$R = QP^*$

$q_2 = (q_{11})(1^*)$

$q_2 = (0^*1)1^*$

R = union of both final states

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

$= 0^*(\epsilon + 11^*)$

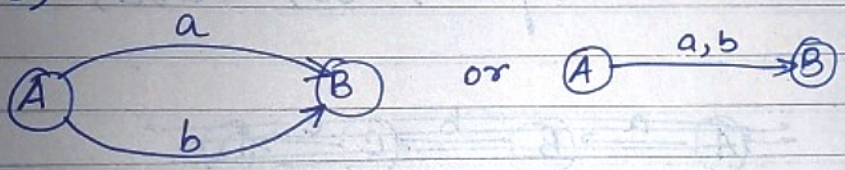
$\epsilon + RR^* = R^*$

$= 0^*1^*$

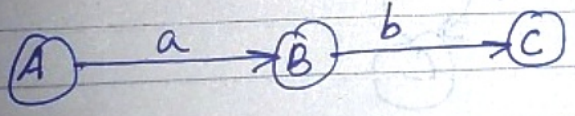
↳ Regular expression

Regular Expression to Finite Automata

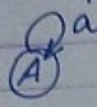
(a+b)



(a b)

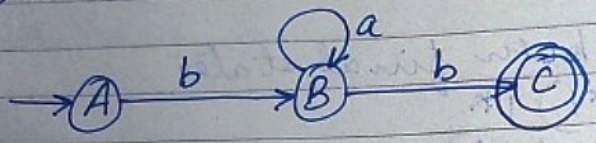


a* (repetition of a)

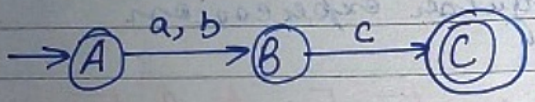


Convert the following regular expression to their finite automata.

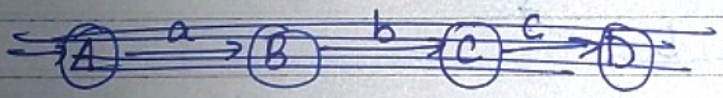
1) ba^*b bb, bab, baab, ...

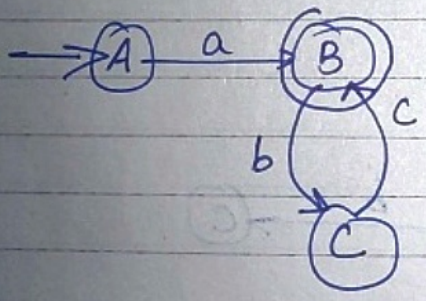


2) $(a+b)c$ ab, bc



3) $a(bc)^*$ E.g:- a, abc, abcbc, ...



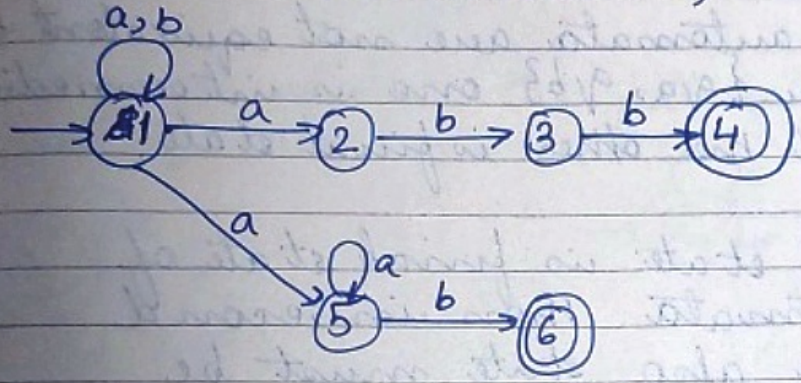


Convert the regular expression to its equivalent finite automata.

$(a|b)^* (abb|a^+b)$

- $(a|b)^*$ → OR same as $+$
- abb | a^+b → make separate states
- → we will have to continue from the state it started.

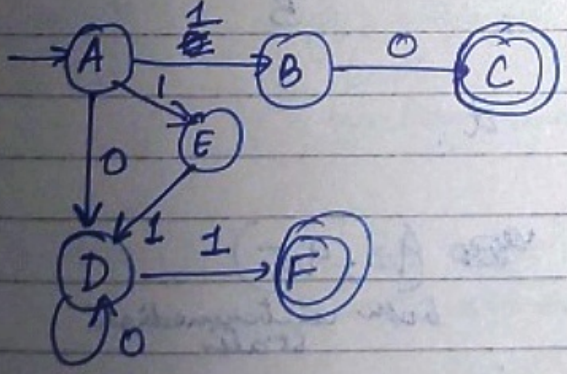
$a^+ = \{a, aa, \ldots\}$

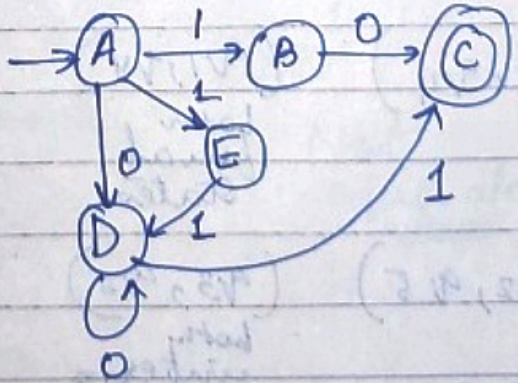


Convert the regular expression to its equivalent finite automata.

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

$\underbrace{(01)}_{a} + \underbrace{(0+11)0^*1}_{b}$





Equivalence of Two Finite Automata

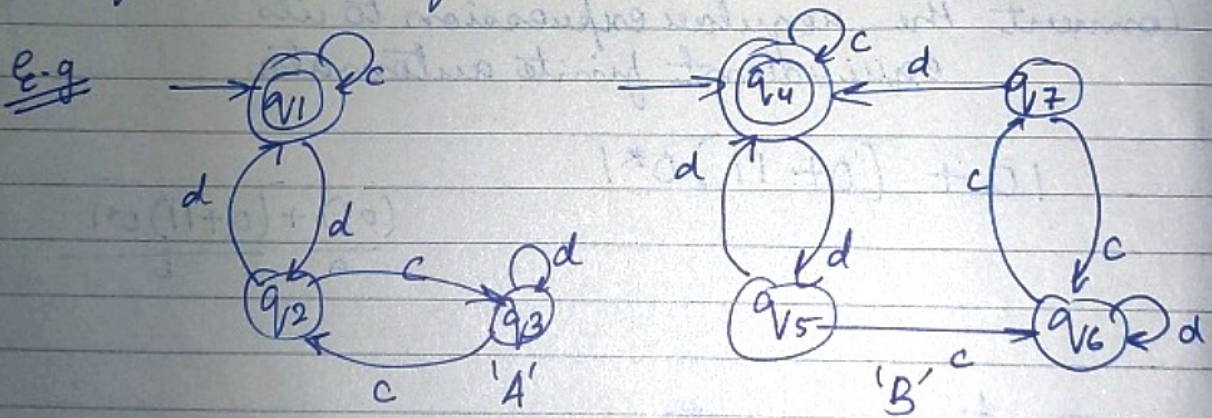
Steps to identify equivalence :-

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.

② If initial state is final state of one automata then in second automata also state must be final state for them to be equivalent.

E.g.

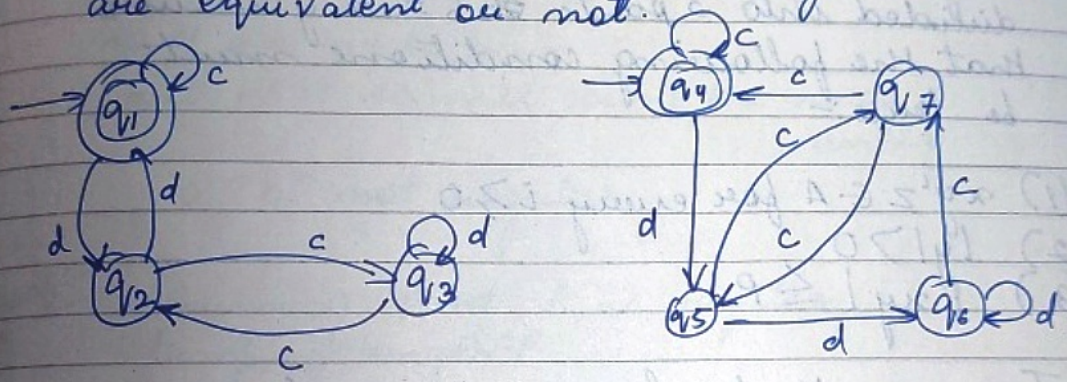


States	c	d
(q_1, q_4)	(q_1, q_4) both final states	(q_2, q_5) both intermediate states
(q_2, q_5)	(q_3, q_6) both intermediate states	(q_1, q_4) both final states

states	c	d
(q_3, q_6)	(q_2, q_7) both intermediate	(q_3, q_6) both intermediate states
(q_2, q_7)	(q_3, q_6) both intermediate	(q_1, q_4) both final.

Therefore the automata A and B are equivalent.

Find out whether the following automata are equivalent or not.



	c	d
(q_1, q_4)	(q_1, q_4) both final states	(q_2, q_5) both intermediate
(q_2, q_5)	(q_3, q_7) both intermediate	(q_1, q_6) FS, IS ~~both intermediate~~ ~~q_3~~
(q_3, q_7)	(q_2, q_5) both intermediate	∴ Not equivalent

Pumping Lemma (for Regular languages)

→ Pumping lemma is used to prove that a language is NOT REGULAR.
→ It cannot be used to prove that a language is REGULAR.

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 3 parts s.t. $s = xyz$ 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:-
We prove 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 xyz
→ Show that none of these can satisfy all the 3 pumping conditions at the same time
→ S cannot be pumped == contradiction

* Show that xy^iz of A for some i
→ Then consider all ways that S can be divided into xyz.

Using Pumping Lemma prove that the language $A = \{a^n b^n \mid n \geq 0\}$ is not Regular.

Regular language is a language that can be designed by finite state machine.

Finite state machine has limited memory.

$a^n b^n$

no of a = no of b

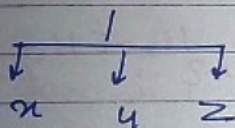
but FSM cannot count the no of a and b.

$\therefore a^n b^n$ not regular.

Proof:- Assuming that A is regular

Pumping length = P

$S = a^P b^P$



P = 7 (let) $|xy| \leq P$

S = a a a a a a a b b b b b b $6 < 7$ ✓

case 1:- y is in the 'a' part.

case 2:- y is in the 'b' part. $|xy| \leq P$

S = a a a a a a a b b b b b b $13 < 7$ X

case 3:- y is the 'a' as well as 'b' part $|xy| \leq P$

a a a a a a a b b b b b b $9 < 7$ X

$xy^iz \Rightarrow xy^2z$

case(i)

aa aaaaaaaaa bbbbbbbb

11 ≠ 7

$\therefore$ this string doesnot belong to Σ

case(ii) xy^2z

aaaaaaa bb bbbbbbbbb

7 11

$7 \neq 11$

$\therefore S \notin \Sigma$

case(iii) xy^2z

aaaaa aabbaabb bbbbb

this doesnot follow the pattern

$|xy| \leq p$

Using Pumping lemma prove that the language $A = \{yy \mid y \in \{0,1\}^*\}$ is not Regular.

any string that can be formed with 0, 1

0101

the first half repeats in the second half.

This is not regular because, the first half is to be remembered. what is in

$\therefore$ ~~not~~ requires memory

$\therefore$ cannot be designed by FSM

$\therefore$ not regular.

Proof:- Assume that A is regular then it must have a pumping length = p

$S = 0^p 1 0^p 1$

x y z

$p = 7$ (let)

0000000 1 00000001

x y z

$xy^i z$ $i = 2$ (let)

$|xy| \leq p$

$6 \leq 7$ ✓ satisfies

xy^2z

00 000000000 1 00000001

11 7

$\notin A$

∴ A is not regular.

Regular Grammar

Noam Chomsky gave a mathematical mode of Grammar which is effective for writing computer languages.

The four types of Grammar according to him are:- P to

mmar cepted	Language accepted	Automata
estricted ammar	Recursively Enumerable Language	Turing machine
ntext Sensitive ~~Language~~ Grammar	Context Sensitive Language	linear Bound Automata
Context free Grammar	Context free Language	Push down Automata
Regular Grammar	Regular Language	Finite State Automaton

mmar:-

grammar 'Gr' can be formally described sing 4 tuples as Gr = (V, T, S, P) where:-

= Set of variables or non terminating symbols

= Set of terminal symbols

S = Start Symbol

P = Production rules for terminals and non terminals.

A production has the form $\alpha \rightarrow \beta$ where α and β are strings on V U T and at least one symbol of α belongs to V.

Regular

Right

A gra be ei all pe the

Automaton

Turing machine

linear bound Automaton

Pushdown Automata

inite state ulomaton.

ribed
ue:-
ng

l

here

Eg :- $S \to AB$
$\to aB$
$\to ab$

$V = \{S, A, B\}$
$T = \{a, b\}$
$S = S$
$P = S \to AB, A \to a, B \to b$

Regular grammar can be divided into two types:-

Right linear grammar

A grammar is said to be right linear if all productions are of the form

$A \longrightarrow xB$
$A \longrightarrow x$

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

If ~~not~~ non terminal symbol lies to the right of terminal symbol.

left linear grammar

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

$A \longrightarrow Bx$
$A \longrightarrow x$

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

E.g $S \to ab\underset{\text{non terminal}}{S} \mid b$ → right linear grammar

terminal symbols

$S \to \underset{\text{non terminal}}{S}bb \mid b$ → left linear grammar.

terminal

Derivations from a Grammar

The set of all strings that can be derived from a grammar is said to be the language generator from that Grammar.

Eg1 Consider the grammar $G1 = (\{S, A\}, \{a, b\}, S, \{S \to aAb, aA \to aaAb, A \to \epsilon\})$

$S \to aAb$ (by $S \to aAb$)
$\to aaAbb$ (by $aA \to aaAb$)
$\to aa\epsilon b$ (by $A \to \epsilon$)
$\to aab$
$\to aaAbb$ (by $aA \to aaAb$)
$\to aaaAbbb$ (by $aA \to aaAb$)
$\to aaabbb$ (by $A \to \epsilon$)

Language generator

Eg2 $G2 = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to a, B \to b\})$

$S \to Ab$
$S \to aB$ (by $A \to a$)
$\to ab$ (by $B \to b$)

$L(G2) = \{ab\}$

only string that can be generated.

Eg3 $G3 = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to aA \mid a, B \to bB \mid b\})$

$S \to AB$
$\to ab$ ($A \to a$, $B \to b$)

$S \to AB$
$\to aAB$ ($A \to aA$)
$\to aAbB$ ($B \to bB$)
$\to aabb$ ($A \to a$, $B \to b$)

$S \rightarrow AB$
$\rightarrow aAb \quad (A \rightarrow aA, B \rightarrow b)$
$\rightarrow aab$

$S \rightarrow AB$
$\rightarrow abB \quad (A \rightarrow a, B \rightarrow bB)$
$\rightarrow abb \quad (B \rightarrow b)$

$L(G3) = \{ab, aabb, aab, abb, \ldots\}$
$= \{ab, a^2b^2, a^2b, ab^2, \ldots\}$
$= \{a^m b^n \mid m \geq 0 \text{ and } n > 0\}$

Context Free Language

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 as $G = \{V, \Sigma, S, P\}$ where
V = set of variables or non terminal symbols
Σ = set of terminal symbols
S = start symbol.
P = production rule

Context free grammar has production rule of the form.

$$A \longrightarrow \alpha$$

where $\alpha = \{V \cup \Sigma\}^*$ and $A \in V$

E.g :- For generating a language that generates equal number of a's and b's in the form $a^m b^n$ the context free grammar will be defined as :-

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

$S \rightarrow aAb$
$\rightarrow aaAbb$ (by $A \rightarrow aAb$)
$\rightarrow aaaAbbb$ (by $A \rightarrow aAb$)
$\rightarrow aaabbb$ (by $A \rightarrow \epsilon$)
$\rightarrow a^3b^3$ no of a's = no of b's
$a^n b^n$
=.

Method to find whether a string belongs to a Grammar or not

1. Start with start symbol and choose the closest production that matches to the given string.
2. Replace the variables with its most appropriate production. Repeat the process until the string is generated or until no other productions are left.

E.g Verify whether the grammar
$S \rightarrow 0B|1A \quad A \rightarrow 0|0S|1AA|\lambda$,
$B \rightarrow 1|1S|0BB$ generates the string
00110101

$S \rightarrow 0B$
$\rightarrow 00BB$ ($B \rightarrow 0BB$)
$\rightarrow 001B$ ($B \rightarrow 1$)
$\rightarrow 0011S$ ($B \rightarrow 1S$)
$\rightarrow 00110B$ ($S \rightarrow 0B$)
$\rightarrow 001101S$ ($B \rightarrow 1S$)
$\rightarrow 0011010B$ ($S \rightarrow 0B$)
$\rightarrow 00110101$ ($B \rightarrow 1$)

∴ grammar generates the same string
=.

Example :- Verify whether the grammar $S \to aAb$, $A \to aAb \mid \wedge$ generates the string aa bbb

$S \to aAb$
$\to aaAbb$ (by $A \to aAb$)
$\to aaaAbbb$ (by $A \to aAb$)
$\to aaabbb$ (by $A \to \wedge$)
∴ does not generate the string

Derivation Tree

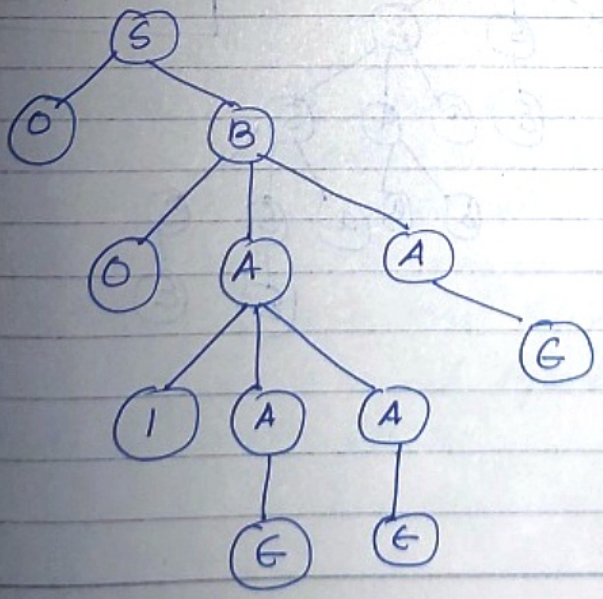
A derivation tree or parse tree is an ordered rooted tree that graphically represents the semantic information of strings derived from a context free grammar.

E.g:- For the grammar $G = \{V, T, P, S\}$ where $S \to 0B$, $A \to 1AA \mid \in$, $B \to 0AA$.

Root vertex: Must be labelled by the start symbol

Vertex: Labelled by non terminal symbols

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



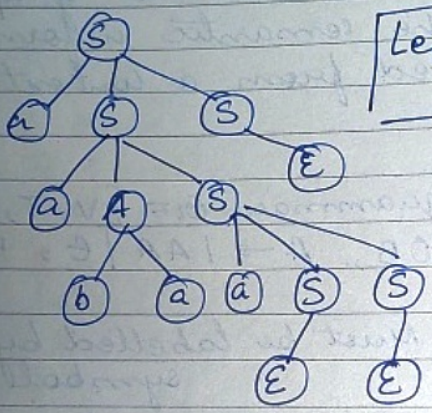
Left Derivation tree

A left derivation tree is obtained by applying production to the left most variable in each step.

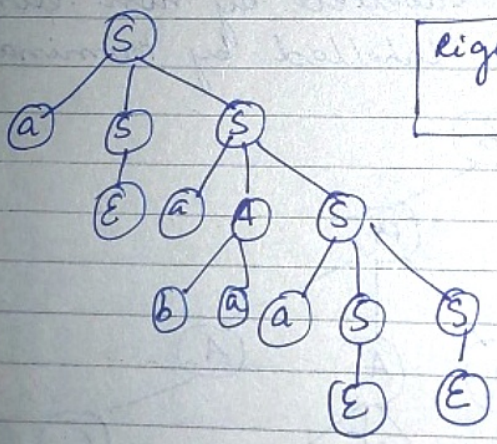
Right Derivation Tree

A right derivation tree is obtained by applying production to the right most variable in each step.

For generating the string aabda from the grammar
S → aAS | aSS | ε A → SbA | ba



Left Derivation tree



Right Derivation tree

A grammar is said to be ambiguous if there exists two or more derivation tree for a string w (that means two or more left derivation tree).

E.g $G_1 = (\{S\}, \{a, +, *\}, P, S\}$ where P consists of $S \to S+S \mid S*S \mid a \mid b$

The string a + a * b can be generated as :-

$S \to S+S$
$\to a+S$
$\to a+S*S$
$\to a+a*S$ $(S \to a)$
$\to a+a*b$ $(S \to b)$

$S \to S*S$
$\to S+S*S$
$\to a+a*S$ $(S \to a)$
$\to a+a*b$ $(S \to b)$

This string were generated using two derivation trees. Thus, this grammar is ambiguous.

Simplification of Context Free Grammars

In CFG sometimes, all the production rules and symbols are not needed for the derivation of strings. Besides this, there may also be some NULL productions and UNIT productions. Elimination of these productions and symbols is called simplification of CFG.

Simplification consists of the following steps :-

① Reduction of CFGs :-

* CFGs are reduced in two phases

Phase 1: Derivation of an equivalent grammar G' from the CFG G, such that each variable derives some terminal string.

Derivation Procedure :-

① Include all symbols W_1 that derives some terminal and initializes $i = 1$.
② Include symbols W_{i+1} that derives W_i.
③ Increment and repeat step 2 until
$W_{i+1} = W_i$
④ Include all production rules that have W_i in it.

Phase 2: Derivation of an equivalent grammar G'', from the CFG, G', such that each symbol appears in a sequential form.

Derivation Procedure :-

① Include the start symbol in Y_1 and initialize $i = 1$.
② Include all the symbols Y_{i+1} that can be derived from Y_i and include all production rules that have been applied.
③ Increment i and repeat step 2 until
$Y_{i+1} = Y_i$.

rules. production

P: S → AC/B, A → a, C → c/BC, E → aA/c

Phase 1 :- T = {a, c, e}

W_1 = {A, C, E} all the symbols that derive a terminal symbol

W_2 = {A, C, E, S} all the symbols that can derive symbols of W_1

W_3 = {A, C, E, S}

G_1' = {(A, C, E, S), {a, c, e}, P, (S)}

(A, C, E, S): non terminal; {a, c, e}: terminal; P: production rule; (S): start symbol

we did not have B in any of the sets present here so we will not include B.

P: S → AC
A → a
C → c
E → aA/c

Phase 2 :- Y_1 = {S} start symbol

Y_2 = {S, A, C} → all the symbols that can derive from the start symbol

Y_3 = {S, A, C, a, c}

Y_4 = {S, A, C, a, c}

G_1'' = {(S, A, C), {a, c}, P, {S}}

P: S → AC
A → a
C → c

Removal of Unit Productions

Any production rule of the form $A \rightarrow B$ where $A, B \in$ Non terminals is called Unit production.

Procedure for Removal

Step 1:- To remove $A \rightarrow B$, add production $A \rightarrow x$ to the grammar rule whenever $B \rightarrow x$ occurs in the grammar [$x \in$ Terminal, x can be Null]

Step 2:- Delete $A \rightarrow B$ from the grammar

Step 3:- Repeat from step 1 until all Unit Productions are removed.

E.g:- Remove Unit productions from the grammar whose production rule is given by:-

P: $S \rightarrow XY$, $X \rightarrow a$, $Y \rightarrow Z \mid b$, $Z \rightarrow M$, $M \rightarrow N$, $N \rightarrow a$

$Y \rightarrow Z$ $Z \rightarrow M$ $M \rightarrow N$ form $A \rightarrow B$ where both A, B are non terminal.

(1) Since $N \rightarrow a$ we add $M \rightarrow a$

P: $S \rightarrow XY$ $X \rightarrow a$ $Y \rightarrow Z \mid b$ $Z \rightarrow M$ $M \rightarrow a$ $N \rightarrow a$

(2) Since $Z \rightarrow M$, $M \rightarrow a$

Then we will add $Z \rightarrow a$

P: $S \rightarrow XY$ $X \rightarrow a$, $Y \rightarrow Z \mid b$, $Z \rightarrow a$, $M \rightarrow a$, $N \rightarrow a$

8) Since $Z \to a$, we can add $Y \to a$

P: $S \to XY$, $X \to a$, $Y \to a|b$, $Z \to a$, $M \to a$, $N \to a$

Remove the unreachable symbols, Z, M, N.

P: $S \to XY$, $X \to a$, $Y \to a|b$

Removal of Null Production

In a CFG, a non terminal symbol 'A' is a nullable variable if there is a production $A \to \epsilon$ or there is a derivation that starts at A and leads to ϵ. (like $A \to \ldots \to \epsilon$)

Procedure for removal:-

Step 1:- To remove $A \to \epsilon$, look for all productions whose right side contains A.

Step 2:- Replace each occurences of 'A' in each of these productions with ϵ.

Step 3:- Add the resultant productions to the Grammar.

E.g Remove null productions from the following grammar.

$S \to ABAC$, $A \to aA | \epsilon$, $B \to bB | \epsilon$, $C \to c$.

$A \to \epsilon$ $B \to \epsilon$

① To eliminate $A \to \epsilon$

$S \to ABAC$ $(A \to \epsilon)$

② $S \to ABC | BAC | BC$

$A \to aA$
$A \to a \quad (A \to \epsilon)$

New Production P:

$S \to ABC \mid BAC \mid BC \mid AABC$
$A \to a \mid aA$
$B \to bB \mid \epsilon$
$C \to c$

(2) To eliminate $B \to \epsilon$

$S \to AAC \mid AC \mid C \quad (B \to \epsilon)$
$B \to b \quad (B \to \epsilon)$

P: $S \to ABC \mid BAC \mid BC \mid AABC \mid AAC \mid AC \mid C$
$A \to a \mid aA$
$B \to b \mid bB$
$C \to c$

Chomsky Normal Form

In Chomsky Normal Form (CNF) we have a restriction on the RHS which is elements in RHS should either be two variables or a terminal.

A CFG is in Chomsky Normal form if the productions are in the normal forms:-

$A \to a$
$A \to BC$

where A, B, C are non terminals and a is a terminal.

Conversion of CFG to Chomsky Normal Form

P: $S \to ASA \mid aB$ $A \to B \mid S$ $B \to b \mid \epsilon$

(1) Check if S appears on the right hand side.
∴ Add a new state S' $S' \to S$.

P: $S' \to S$, $S \to ASA \mid aB$, $A \to B \mid S$, $B \to b \mid \epsilon$

(2) Remove the null productions: $B \to \epsilon$ and $A \to \epsilon$.

~~After~~ $B \to E$

$S \to ASA \mid aB$

$S \to aB$ $(B \to E)$ $B \to b$.

$S \to a$

$\boxed{S \to ASA \mid aB \mid a}$

$A \to B$ $(B \to \epsilon)$

$A \to \epsilon$

$\boxed{A \to B \mid S \mid \epsilon}$

P: $S \to ASA \mid aB \mid a$, $A \to S \mid B \mid \epsilon$, $B \to b$

$A \to \epsilon$

$S \to ASA$

P: $S \to AS \mid SA \mid S \mid ASA \mid aB \mid a$, $A \to S \mid B$, $B \to b$

↓

$S' \to S$

(3) Remove the unit productions.

~~Since $B \to b$~~

~~$A \to S \mid b$~~

P: $S' \to S$ $S \to AS/SA/S/ASA/aB/a$
$A \to B/B$ $B \to b$

$S' \to S$ $S \to S$ $A \to S$ $A \to B$

(1) Removing $S \to S$:

$S' \to S$
$S \to AS/SA/ASA/aB/a$
$A \to S/B$
$B \to b$

(2) Removing $S' \to S$:

$S' \to AS/SA/ASA/aB/a$
$S \to AS/SA/ASA/aB/a$
$A \to S/B$
$B \to b$

(3) Removing $A \to S$:

$S' \to AS/SA/ASA/aB/a$
$S \to AS/SA/ASA/aB/a$
$A \to AS/SA/ASA/aB/a/B$
$B \to b$

(4) Removing $A \to B$:-

$S' \to AS/SA/ASA/aB/a$
$S \to AS/SA/ASA/aB/a$
$A \to AS/SA/ASA/aB/a/b$
$B \to b$.

/aB/a
/B B→b.
→B

(4) Now find out the productions that has more than 2 variables:-
S' → ASA S→ASA and A→ASA
After removing we get,

P: S' → AX | aB | a | AS | SA
S → AX | aB | a | AS | SA
A → b | AX | aB | a | AS | SA
B → b
X → SA

(5) Now change the productions:-

S' → aB
S → aB
A → aB

S' → AX | YB | a | AS | SA
S → AX | YB | a | AS | SA
A → b | AX | YB | a | AS | SA
B → b
X → SA
Y → a

which is the req chomsky normal form for given CFG.

Greibach Normal Form

A CFG is in Greibach Normal Form if the productions are in the following forms:-

$A \rightarrow b$

$A \rightarrow b C_1 C_2 \ldots C_n$

where $A, C_1, \ldots C_n$ are non terminals and b is a terminal.

Steps to convert a given CFG to GNF:-

Step 1:- Check if the given CFG has any unit productions or null productions and remove if any.

Step 2:- Check whether the CFG is in Chomsky Normal Form (CNF) and convert into CNF if not.

Step 3:- Change the names of the non terminal symbols into some A_i in ascending order of i.

order →

Eg $S \rightarrow CA \mid BB$
$B \rightarrow b \mid SB$
$C \rightarrow b$
$A \rightarrow a$

Replace:-
S with A_1
C with A_2
A with A_3
B with A_4.

$A_1 \rightarrow A_2 A_3 \mid A_4 A_4$
$A_4 \rightarrow b \mid A_1 A_4$
$A_2 \rightarrow b$
$A_3 \rightarrow a$

Step 4:-

Step

Step 4:- Alter the rules so that the non terminals are in ascending order such that the production is of the form $A_i \rightarrow A_j x$ $(i<j)$

$A_1 \rightarrow A_2 A_3 \mid A_4 A_4$ ✓

$= 1<2$

$1<4$

✓ already in GNF

$A_4 \rightarrow b \mid A_1 A_4$

↑ replace A_1

$A_4 \rightarrow b \mid A_2 A_3 A_4 \mid A_4 A_4 A_4$

replace A_2 with b

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

terminal symbol followed by anything GNF ✓ — left recursion

Step 5:- Remove left recursion.

Introduce a new variable to remove the left recursion.

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

↓ the variables that follow the problematic variable.

$Z \rightarrow A_4 A_4 Z \mid A_4 A_4$

$A_4 \rightarrow b \mid b A_3 A_4 \mid bZ \mid b A_3 A_4 Z$

In GNF form.

Now the grammar is:-

$A_1 \rightarrow A_2 A_3 \mid A_4 A_4$
$A_4 \rightarrow b \mid b A_3 A_4 \mid bZ \mid bA_3 A_4 Z$
$Z \rightarrow A_4 A_4 \mid A_4 A_4 Z$
$A_2 \rightarrow b$
$A_3 \rightarrow a$

Change to GNF:-
replace A_2 with b

$A_1 \rightarrow bA_3 \mid bA_4 \mid bA_3 A_4 A_4 \mid bZA_4 \mid bA_3 A_4 A_4$
$A_4 \rightarrow b \mid bA_3 A_4 \mid bZ \mid bA_3 A_4 Z$
$Z \rightarrow bA_4 \mid bA_3 A_4 A_4 \mid bZA_4 \mid bA_3 A_4 ZA_4 \mid$
$bA_4 Z \mid bA_3 A_4 A_4 Z \mid bZA_4 Z \mid$
$bA_3 A_4 ZA_4 Z.$

$A_2 \rightarrow b$
$A_3 \rightarrow a$

Pumping Lemma (for content free language)

Pumping lemma (for CFL) is used to prove that a language is not content free.

Content Free language.

In formal language theory a content free language is a language generated by some content free grammar.

The set of all CFL is identical to the set of languages accepted by Pushdown Automata.

$G = \{V, \Sigma, S, P\}$
V = set of variables / non terminal symbols

Σ = Set of terminal symbols
S = Start symbol
P = Production rule.

Context free grammar has production rule of the form

$$A \rightarrow a$$

where $a = \{V \cup \Sigma\}^*$ and $A \in V$.

Pumping lemma is used to prove that a language is NOT Context free.

If A is a Context free language then A has a pumping length 'p' such that any string 's' where $|s| \geq p$ may be divided into 5 pieces $s = uvxyz$ such that the following conditions must be true:-

(1) $uv^i xy^i z$ is in A for every $i \geq 0$
(2) $|vy| > 0$
(3) $|vxy| \leq p$

Show that $L = \{a^N b^N c^N \mid N \geq 0\}$ is not context free.

→ Assume that L is Context free
→ L must have a pumping length (say P)
→ Now we take a string such that

$$S = a^p b^p c^p$$

→ We divide into parts $uvxyz$