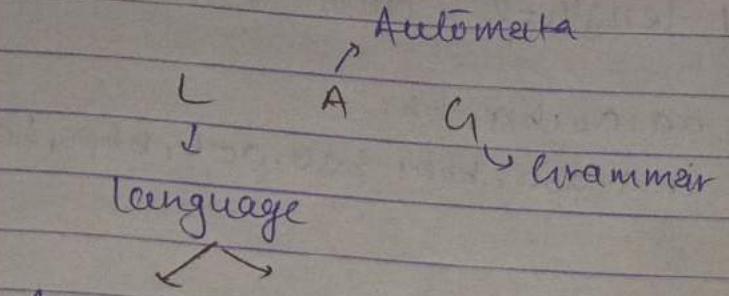
Theory of Computation (ToC)

2 April 2023



Finite state Machine (FSM)

Symbol :- The smallest unit of the language as $\{a, b, c, 0, 1, 2 \ldots\}$

Alphabet :- it is a finite set of symbol. and denoted by Σ (collection of symbol)

$$\Sigma(a, b) = \{a, b\} \quad \Sigma = \{0, 1\}$$

String :- sequence of alphabets

$\Sigma(a, b) = a, b, aa, ab, aba, \ldots$

language :- a finite set of strings / collection of all possible strings.

L_1 : set of all strings of length 2 as 01, 00, 11, ...

L_2 : " " " " 3 as 001, 000 ...

Terminal - It is a symbol which can't be split.

eg. a, b, c, d

Non-Terminal - It is a symbol which can split. It is used to generate strings.

eg, abc, ab, xy etc

Q $\Sigma(a,b)$ if length=2 and 3

Soln L=2 then aa, ab, ba, bb

L=3 " aaa, aba, bbb, bab, aab, bba, baa, abb.

Automata:- it is a maths model which helps to determine that whether the string is a part of language or not by checking the grammar of the string

- Finite Automata (FA)
- Push down " (PDA)
- Linear Bound " (LBA)
- Turing Machine

Power of Σ

Σ represents alphabets, $\Sigma(a,b)$

Σ^0 = set of all strings with length 0 = (λ, Σ) (Null string)

Σ^1 = length 1 $\Rightarrow (a,b)$

Σ^n = set of all string of length n

Σ^* $\Rightarrow$ Kleen closure $\Rightarrow (a+b)^*$ $\rightarrow$ Infinite language

Total possible no = 2^n

Σ^+ = positive closure ← In this all string possible except ε

$$\Sigma^* = \Sigma^+ + \varepsilon$$

$$\Downarrow$$

$$\boxed{\Sigma^+ = \Sigma^* - \varepsilon}$$

↳ Identity element

Cardinality: Toteal no. of elements in a set

$|\omega|$

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

$$\Sigma^* = \varepsilon^0 \cup \varepsilon^1 \cup \varepsilon^2 \cup \cdots \varepsilon^n$$

↳ set of all possible string of all length over $\{0,1\}$

(G) Grammar: it is defined as quadruples and it is a standard way of representing a language

$$G = \{ V, T, P, S \}$$

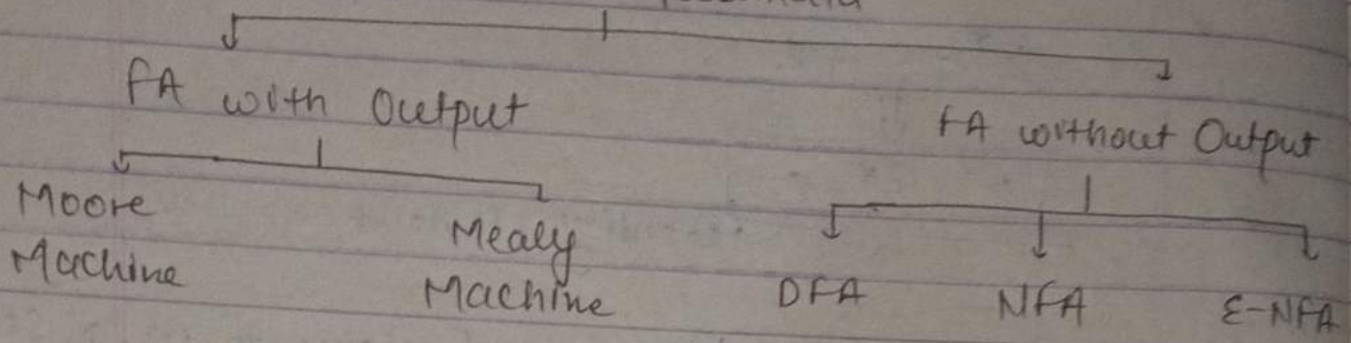
V → variable, T → Terminal, P → Production Rule, S → start symbol

There is two method to identify whether the string is a part of language or not

a) Automata b) Grammar

Language $\Rightarrow$ $n_a(\omega) = n_b(\omega)$ where n = no of string.

Finite State Machine → it is not applicable for all domains



Prop. of FSM

- it is the simplest model of computation.
- very limited Memory
- flexible
- works with only linear Power

Deterministic finite Automata (DFA)

$\{Q, \Sigma, \delta, q_0, f\}$ ← set of all final state

- Q ↓ set of all states
- Σ ↓ set of alphabets {a|b}
- q_0 ↓ start state
- δ ↓ Transition func

- Unique transition in any ~~State on an input symbol~~

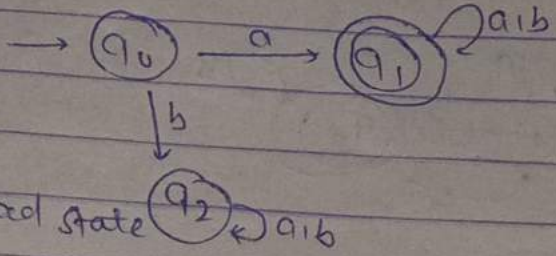
$F \subseteq Q$ → Final states is a subset of all states

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

- It can contain multiple final states. it is used in lexical Analysis in Compiler

Q L = set of all strings starting with 'a'

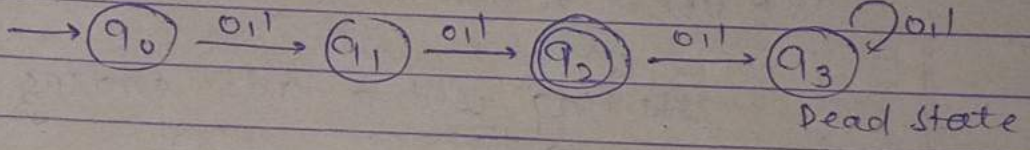
Soln



Q Construct a DFA that accept set of all strings over {0|1} of length 2

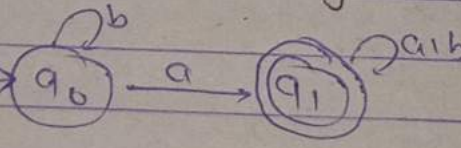
Soln

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

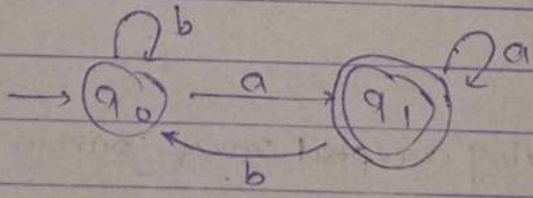


Q Construct a DFA which accept a language of all strings a) Containing a b) End with a

Soln a)



b)

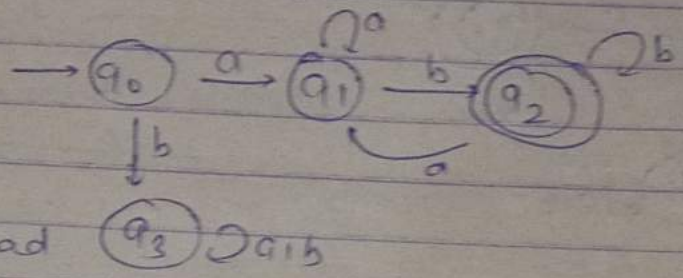


Applications:

- Compiler Design
- Design & Analysis of Digital circuit
- Design Analysis of H/w & S/w system
- Mealy & Moore Machine design
- Programming language, Text Editor, Game design etc.

Q Construct a DFA that accepts all strings starting with 'a' and ending with 'b'.

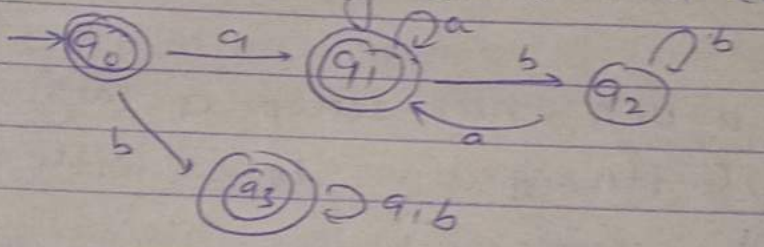
Soln



Q DFA that accepts all strings not starting with "a" or not ending with 'b'

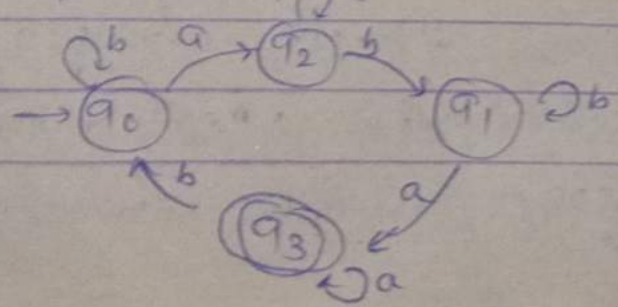
Soln $(A \cup B)^c = A^c \cap B^c$ $L = \{\lambda, \varepsilon, a, b, bb, \ldots\}$

↳ starting with a and ending with b



Q $\Sigma(a|b)$ such that string accepted must contain odd no of occurance of substring ab.

Soln $|w|_{ab} = 1 \pmod 2$



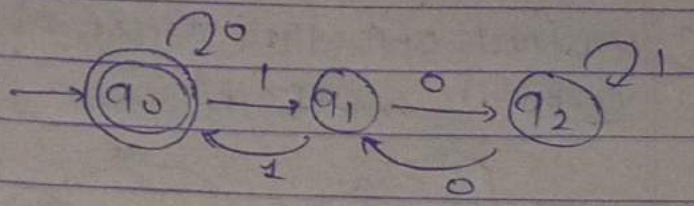
** Q DFA that accepts all binary strings divisible by three over $\Sigma(0,1)$

Soln. ↙ Remainder

$q_0 = 0$

$q_1 = 1$

$q_2 = 2$



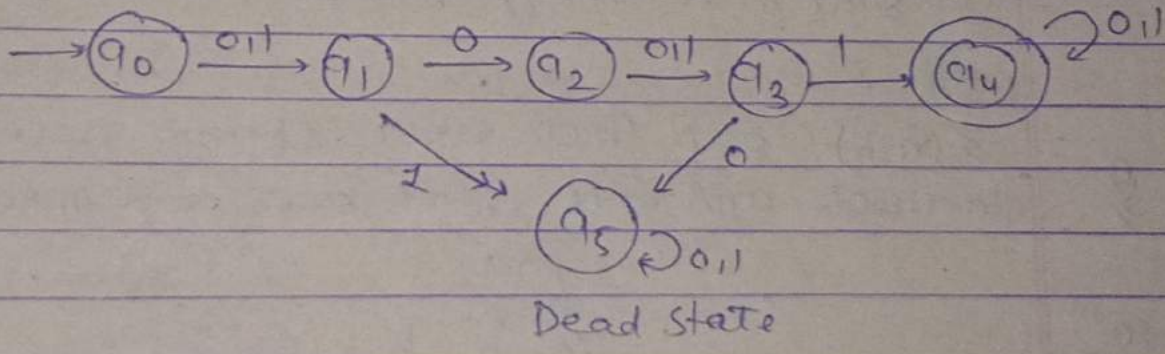
** Q DFA that accepts all strings over {0,1} in which second symbol is '0' and fourth symbol is '1'

Soln

— 0 — 1 —

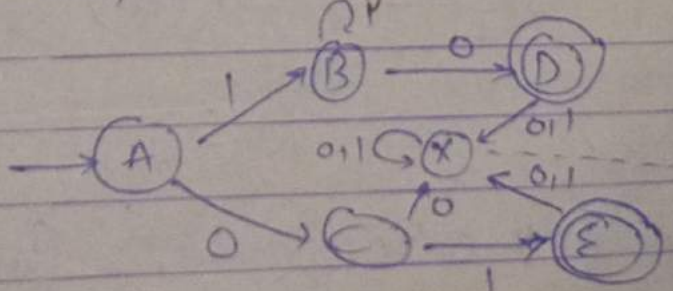
min length of the string = 4

∴ no. of states = n+1 +1 ↳ Trap State



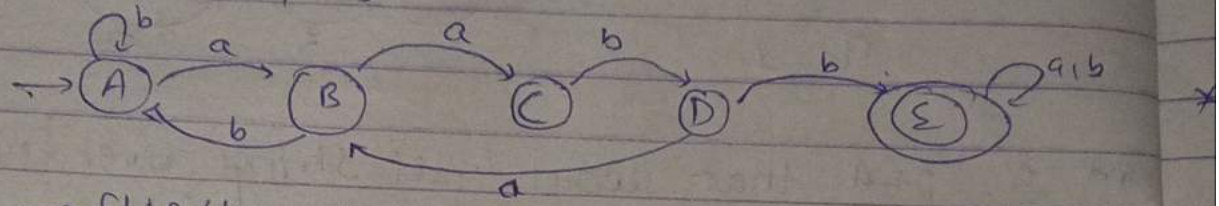
Q DFA that accepts the string 01 or a string of atleast one 1 followed by a '0'.

Soln

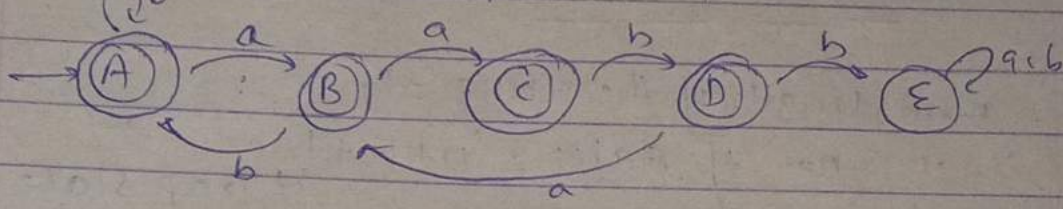


** Q DFA that accepts any strings over {a,b} that doesn't contain the strings 'aabb' in it

Soln Construct DFA that accept all strings over {a,b} that contains aabb init . Σ={a,b}



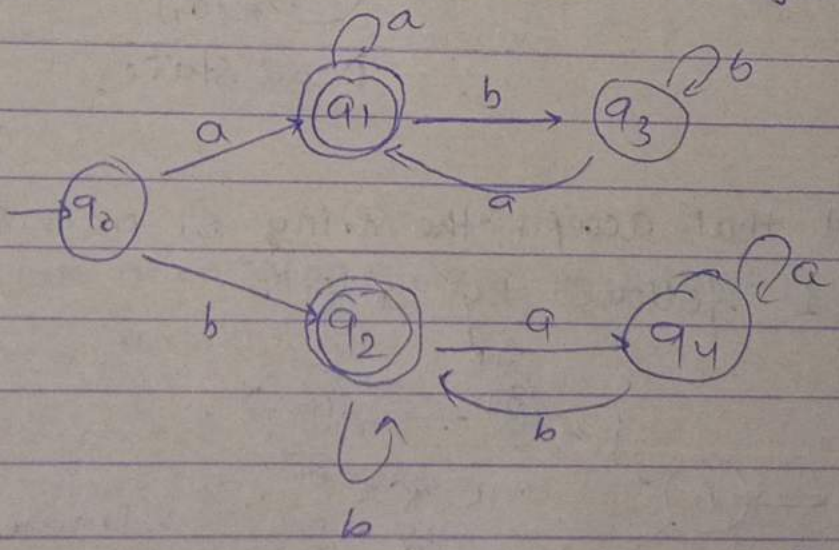
→ flip the states
→ make the final state into non-final state
→ make non final → final state



Since E is not a final state string is not accepted

Q Σ(a,b) such that every string accepted must start ~~with~~ and ends with same symbol.

Soln



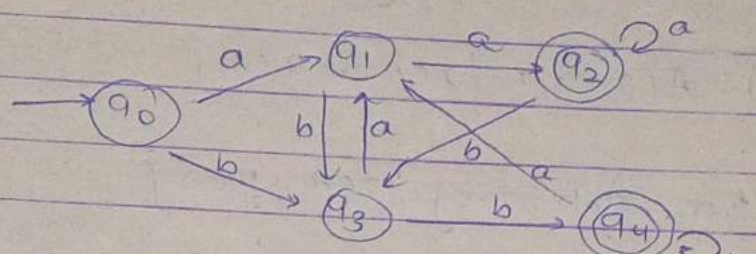
start with & End with diff symbol q_3 & q_4 will be final state.

prop. of DFA

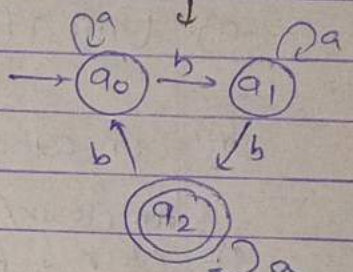
- It has only one unique next state
- it has no choice / Randomness.
- it is simple and easy to design.

** Q Σ(a|b) such that every string accepted must ends with aa or bb. →

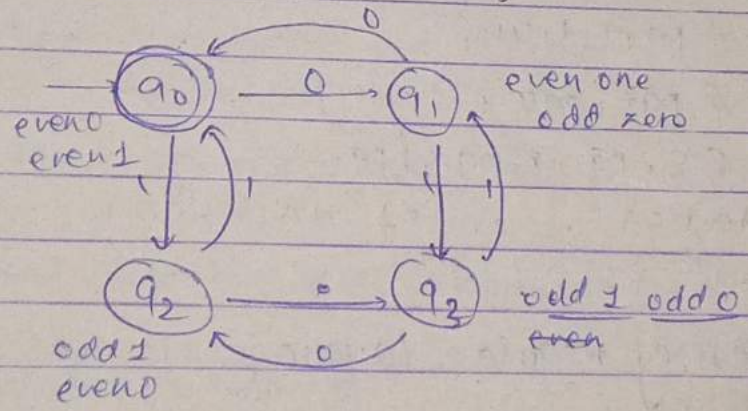
Soln



c) $|w|_b = 2 \pmod 3$



Q Design DFA for Even no of 1's & Even no. of 0's.

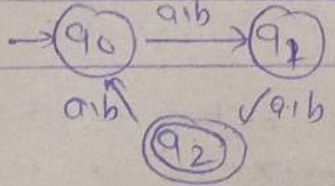


Q Σ(a|b) a) $|w| = 2 \pmod 3$ b) $|w| = 3 \pmod 4$

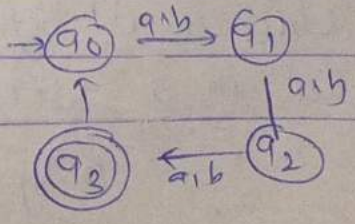
Soln $|w| = r \pmod n$ ← Generalised form

↓ final state ↓ no of states

a)



b)



Regular language: if and only if some FSM recognize it

Language that aren't Regular
- not recognized by any FSM
- require memory

Some operations of Regular languages:-

a) Union (U): $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$

b) Concatenation: $A \circ B = \{xy \mid x \in A, y \in B\}$

c) Star (Kleen closure): $A^* = \{x_1, x_2, x_3, \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$

(Qy) $A = \{pq, r\}$ $B = \{t, uv\}$

Soln

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

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

$A^* = \{\varepsilon, pq, r, pqr, rpq, \ldots\}$

d) Complementation e) Intersection

f) Reversal $\rightarrow L^R \{w^R \mid w \in L\}$

Theorem 1: The class of Regular language is closed under UNION.

Theorem 2: The class of Regular language is closed under Concatenation.

Substring:- Sequence of symbols that appears in the same order in a string.

No of states = $n+2$ → start → Dead ✓

$= n+1$ → end → " ✗

Non-Deterministic finite Automata (NFA)

$(Q, \Sigma, q_0, f, \delta)$ $\quad f \subseteq Q$

- There could be multiple next states
- Next state may be choosen at Random
- it also accept empty / Null states.

→ it is used to check the Regular language is accepted by Machine or not

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

↳ Total no. of transitions.

Diff b/w DFA & NFA

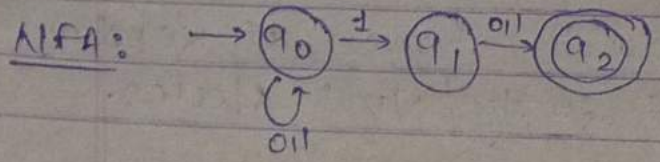
DFA	NFA
• Decid configuration is not allowed	• Allowed
• Multiple choices are not Allowed	• Possible
• ε- move is not possible	• Allowed
• Digital computers are Deterministic.	• Non-Deterministics features isn't associated with Real Computers
• Require more space as compared to NFA.	
• Designing and understanding is Difficult	• Easy

Note:- Max. possible states $= 2^n$, when we convert NFA → DFA.

Q NFA of all Binary strings in which 2nd last bit is 1.

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

NFA:



NFA Transition table:-

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

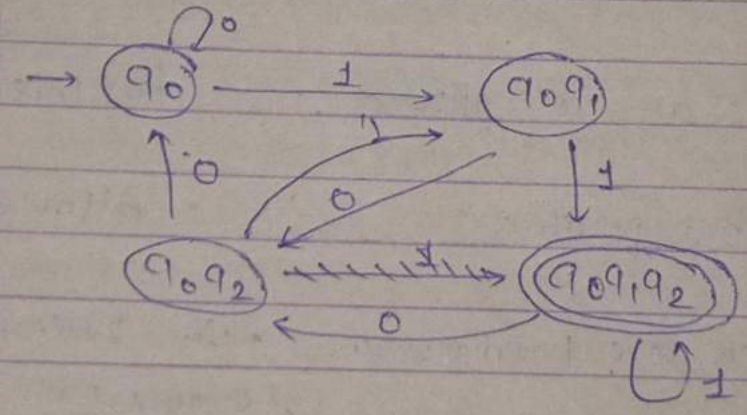
Convert NFA → DFA

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

Step 1: write down first state of NFA transition table

" 2: ~~write~~ Expand state by using NFA transition table

Step 3: Expand Each state.

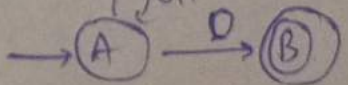


← DFA

→ 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

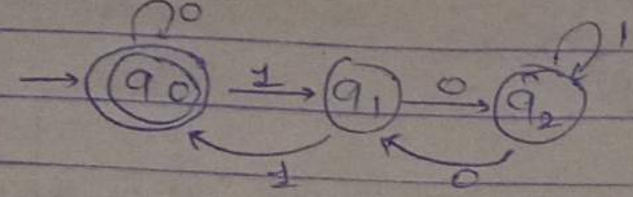
Q L: Set of all strings that ends with 0

Soln $L = 10, 1010, \ldots$



Q DFA which accept a language all binary strings divisible by three over $\Sigma(0,1)$

Sol^n



$q_0 = 0 - 0$
$q_1 = 1 \leftarrow 01$
$q_2 = 2 \leftarrow 10$

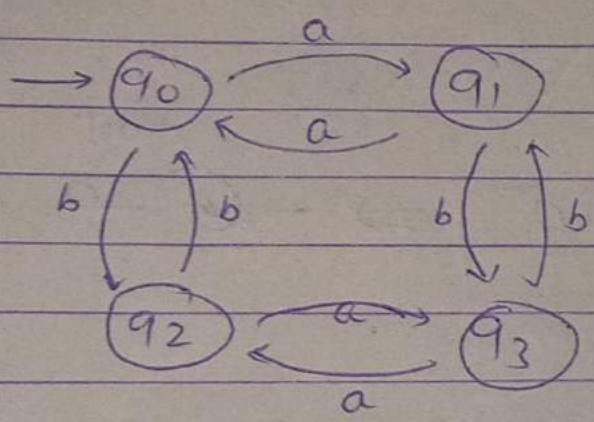
***** imp

Q — Design DFA for the language L that accepts all strings which has even no. of a's and even no of b's

$L = \{ w \mid w \in$ even no of a's & b's $\}$

Sol^n

$L = \{ \varepsilon, aa, bb, abab, aabb, \ldots \}$



q_0 = Even a & b

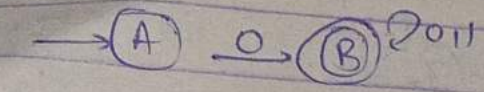
q_1 = odd a, Even b

q_2 = Even a odd b

q_3 = odd a & b

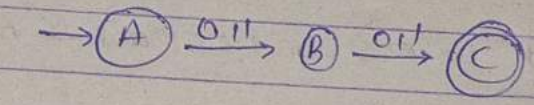
Q. L = set of all strings that start with 0

Sol^n L = 0, 00, 001, 010, ...



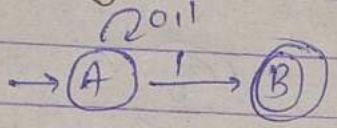
Q. L = set of all strings over {0,1} of length 2

Sol^n L = 10, 01, 00, 11

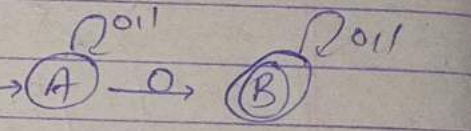


Q. L = set of all strings that end with 1

Sol^n



Q. L = that contain 0 $\Rightarrow$



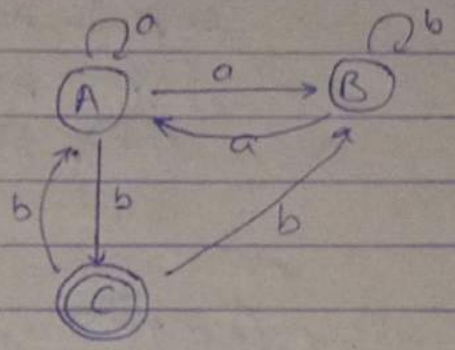
Q.

Q 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

Sol^{n}

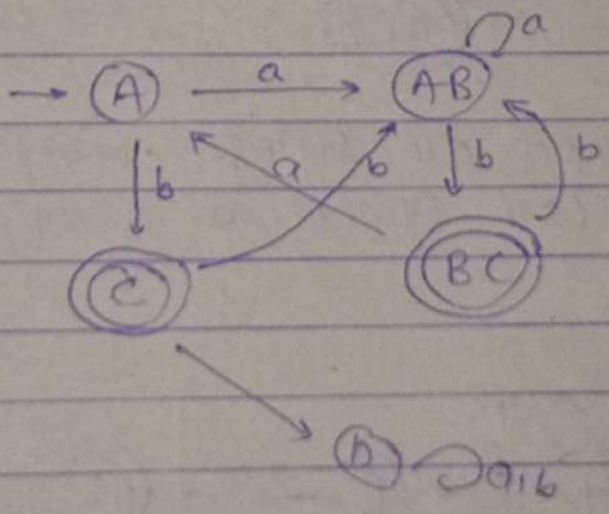
NFA:



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

D = Dead state

DFA:

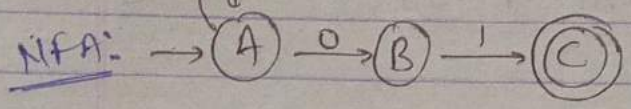


Note: Every DFA is a NFA but not vice versa but there is an equivalent DFA for every NFA

NFA → DFA Construct

Q L = Set of all strings over (0,1) that ends with '01'

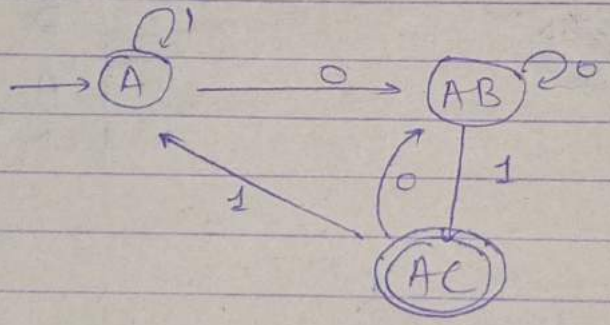
NFA:



	0	1
→ A	A|B	A
B	ϕ	C
(C)	ϕ	ϕ

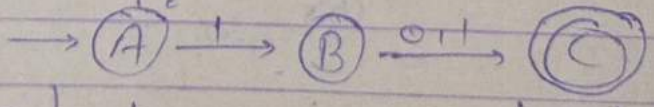
Transition Table

	0	1
→ A	AB	A
AB	AB	AC
(AC)	AB	A



Q L that accepts all strings over {0,1} in which the second last symbol is always '1'. convert NFA → DFA

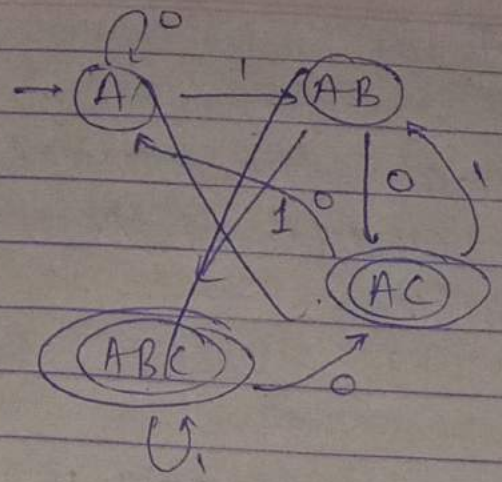
Sol^n NFA:

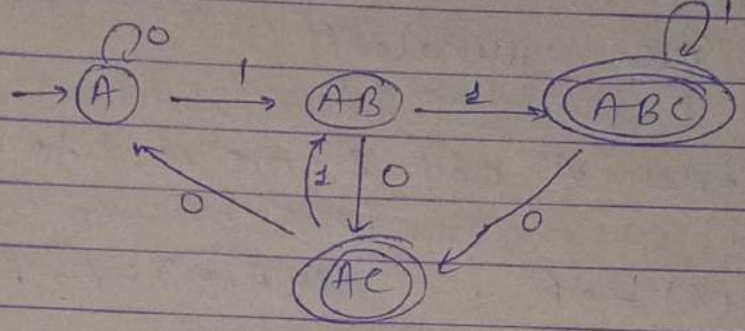


	0	1
A	A	A|B
B	C	C
(C)	ϕ	ϕ

⇒

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





Epsilon-NFA (ε-NFA)

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

$\delta : Q \times \{\Sigma \cup (\epsilon)\} \rightarrow 2^Q$

Note:- Time needed for executing an I/P string is more in NFA.

Minimization of DFA

It is required to obtain the minimal version of any DFA which consists of the minimum no of states possible

→ Two states can be combined only when two states are equivalent

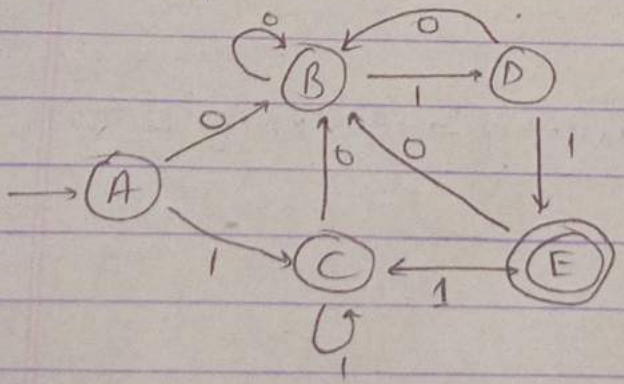
Two states 'A' and 'B' are said to be equivalent if

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

where X = Input string

if $|X| = 0$ then A & B → 0' Equivalent.
if $|X| = 1$ " " → 1 equivalent
if $|X| = 2$ " " → 2 equivalent
:
$|X| = n$ " " → n equivalent.

Q



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

So 1st 0 Equivalence = Set 1 = Non final state
Set 2 = final state

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

1's Equivalence : check Equivalent and their outcomes

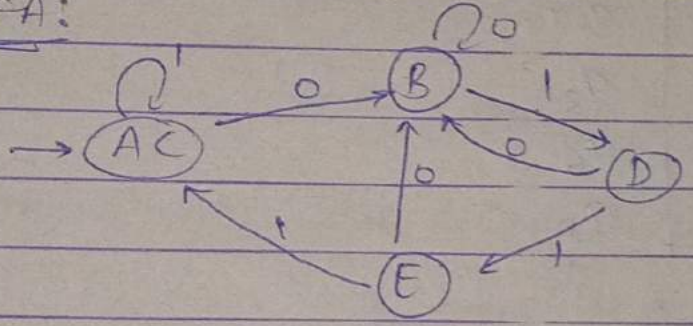
{A, B, C} {D} {E}

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

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

goes on same state

DFA:



← Minimal version of DFA

Q Construct a minimum DFA equivalent to the DFA described by.

	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_0
q_6	q_6	q_4
q_7	q_6	q_2

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

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

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

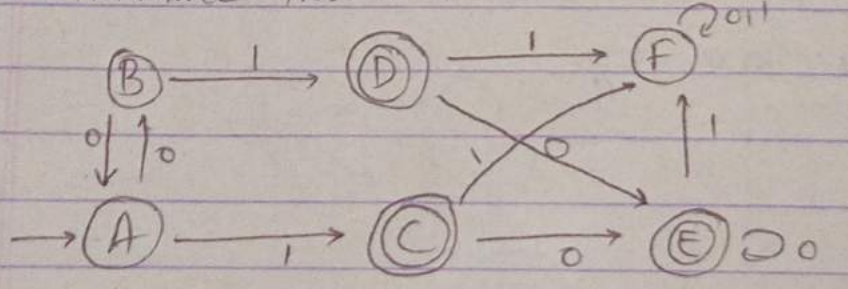
Check every set their Equivalent

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

→ When there are more than one final state involved

Minimize the DFA:



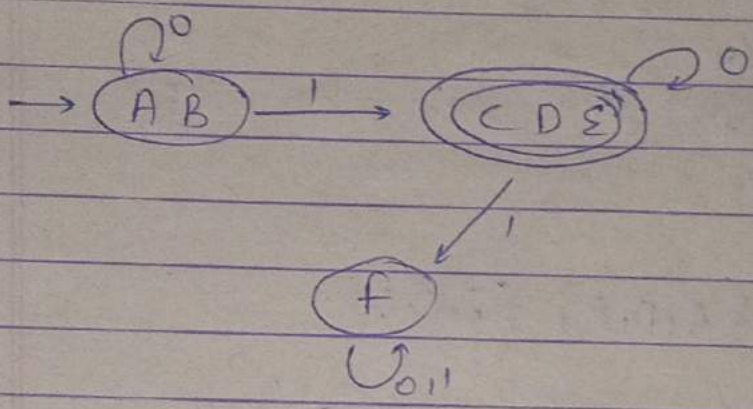
	0	1
→ A	B	C
B	A	D
(C)	E	f
(D)	E	f
(E)	E	f
f	f	f

0 Equivalence: {A, B, F} {C, D, E}

1 Equivalence: {A, B} {F} {C, D, E}

2 Equivalence: {A, B} {F} {C, D, E}

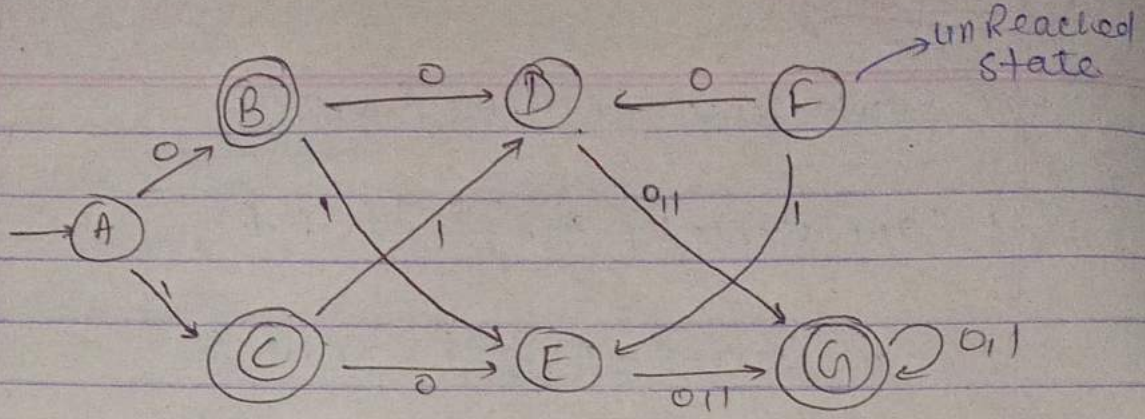
	0	1
→ {A, B}	{A, B}	{C, D, E}
{F}	F	F
{C, D, E}	~~E~~ {C, D, E}	F



→ Where there are Unreachable States involved
↓
if there is no way it can be reached from the initial state.

Procedure:- Remove unreached state from Diagram then proceed as previously.

Q



Sol^{n} Remove unreachable state then proceed.

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

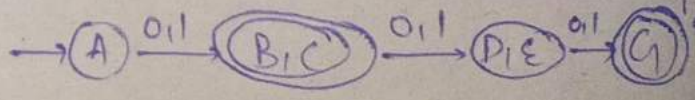
0 Equivalence: {A, D, E} {B, C, G}

1 Equivalence: {A, D, E} {B, C} {G}

2 Equivalence: {A} {D, E} {B, C} {G}

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

	D	1
→ A	BC	BC
D, E	G	G
(B, C)	DE	DE
(G)	G	G



Q

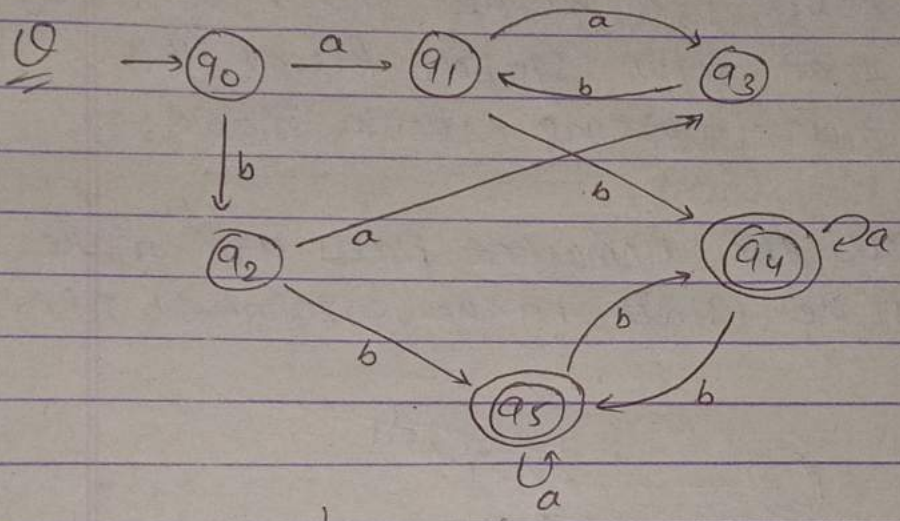
→ Minimization of NFA → DFA

step 1: Remove Unreachable state

step 2: Convert into Equivalent Class

Make group 1: Accepting (final)

2: Non-Accepting (Non-final state)



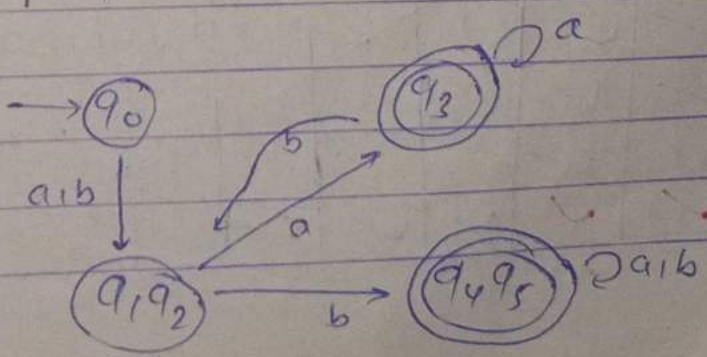
	a	b
→ q_0	q_1	q_2
q_1	q_3	q_4
q_2	q_3	q_5
q_3	q_3	q_1
q_4	q_4	q_5
q_5	q_5	q_4

0 Equivalence: $\{q_0 q_1 q_2\}\{q_3 q_4 q_5\}$

1 Equivalence: $\{q_0\}\{q_1 q_2\}\{q_3\}\{q_4 q_5\}$

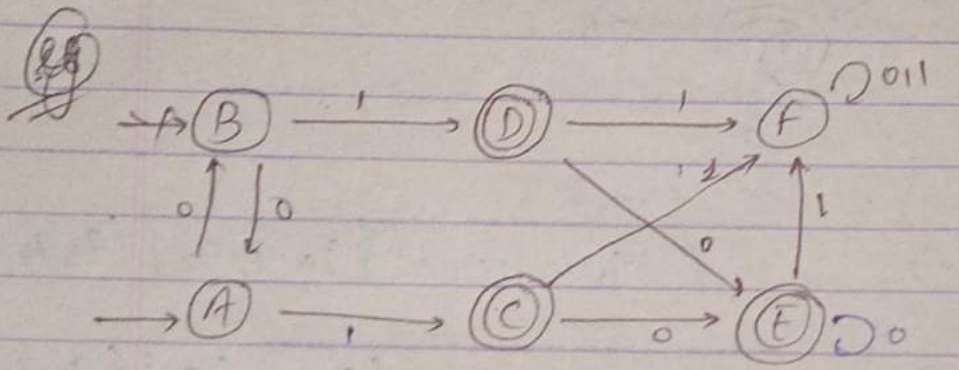
2 Equivalence: $\{q_0\}\{q_1 q_2\}\{q_3\}\{q_4 q_5\}$

DFA:



Myhill Nerode Theorem (Table filling Method)
(Minimization of DFA)

Steps:-
- Draw a table for all pairs of states (P, Q)
- Mark all pairs where P ∈ F and Q ∉ F → Final state = F
- if there are any unmarked pairs (P, Q) such that [δ(P, x), δ(Q, x)] is marked then mark [P, Q] where x is an input symbol. Repeat this until no more marking can be made.
- Combine all the unmarked pairs and make them a single state in the minimized DFA



→ Divide Table diagonally and remove upper part then mark only those that Q ∈ F & Q ∉ F

→ Now check Unmarked pair

(BA) → δ(B,0) → A, δ(A,0) → B } AB δ(B,1) → D, δ(A,1) → C } DC

↓

if unmarked then don't do anything

(DC) → δ(D,0) - E, (C,0) - E } (D,1) → f, (C,1) → f }

(EC) → δ(E,0) - E (E,1) - f
(C,0) - E (C,1) - f

(E,D) - (E,0) - E, (D,0) - E } (E,1) - f, (D,1) - f }

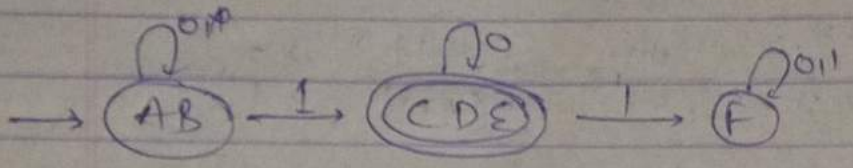
FA - (f,0) - f, (A,0) - B } FA (F,1) - f, (A,1) - C }

if FC is marked then mark also FA

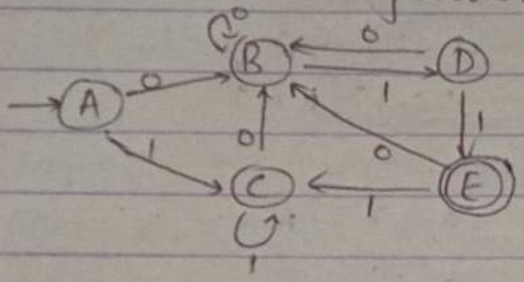
FB - (f,0) - f, (B,0) - A } (f,1), (B,1)

Now Combine all unmarked pair and make them a single state in minimized DFA

(A,B) (D,C) (E,C) (E,D) ← unmarked pair



Q Minimize the following DFA using Table filling



Soln

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

(B,A) → δ(B,0) - B (B,1) - D
(A,0) - B (A,1) - C

CA - δ(C,0) - B δ(C,1) - C
(A,0) - B δ(A,1) - C

CB - δ(C,0) - B (C,1) - C
δ(B,0) - B (B,1) - D

D,A - (D,0) - B (D,1) - E } marked
(A,0) - B (A,1) - C

D,B - (D,0) - B (D,1) - E
(B,0) - B (B,1) - D

D,C - (D,0) - B (D,1) - E
(C,0) - B (C,1) - C

2nd Iteration

(B,A) - B,0 - B (B,1) - D }
A,0 - B (A,1) - C

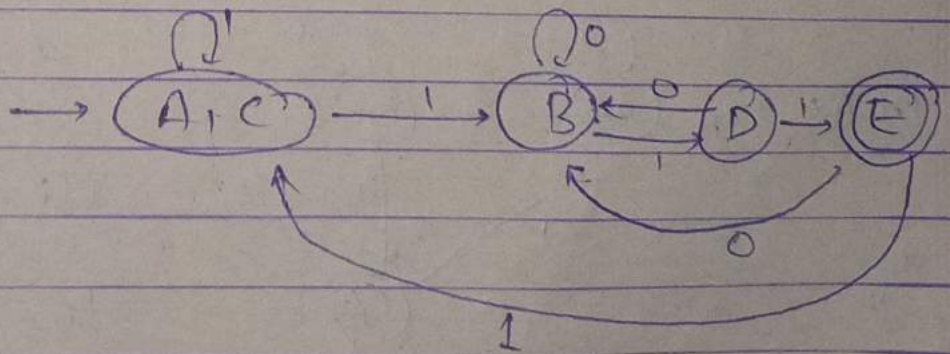
(CA) - C,0 - B C,1 - C
A,0 - B A,1 - C

CB - C,0 - B C,1 - C
B,0 - B B,1 - D

3rd Iteration

CA → X

unmarked pair - (C,A)



Finite Automata with Outputs

⇒ Moore Machine

$M_0 := (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where

Q = finite set of states

$\delta : Q \times \Sigma \to Q$ = Transition fun^n

q_0 = start state

Δ = Output symbol

λ = " function

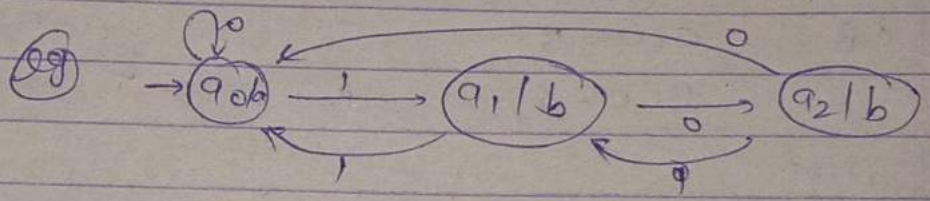
$\lambda : Q \to \Delta$

In this outputs is associated with states

$z(t) = \lambda(q(t))$ → Present state

$z(t)$ ↓ value of o/p func^n ; λ ↓ output func^n

- If we providing n length input the Moore machine gives n+1 numbers



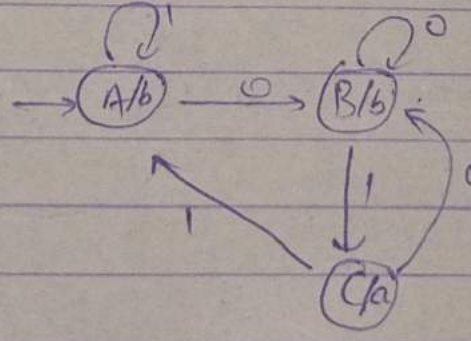
Q. Construct a Moore Machine that prints 'a' whenever the sequence '01' is encountered in any input binary string

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

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

Soln

DFA:



Moore

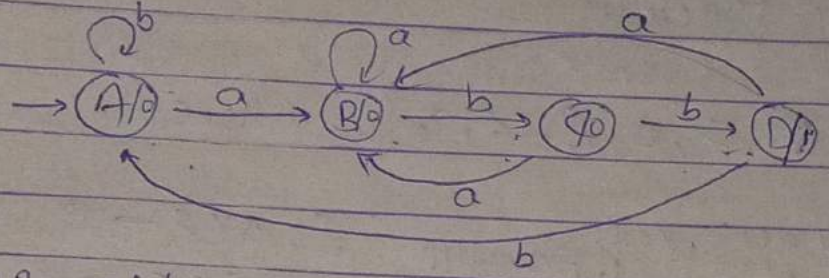
I/P = 0110

O/P = bbabb.

Q Construct a Moore Machine that counts the occurences of the sequence 'abb' in any input strings over {a,b}

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

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



I/P = abb
O/P = 0001

I/P = abbabb
O/P = 0001001

Q I/P alphabet $\Sigma = \{a,b\}$, O/P alphabet $\Delta = \{0,1\}$. Run following I/P a) aababab b) abbb c) ababb

Soln

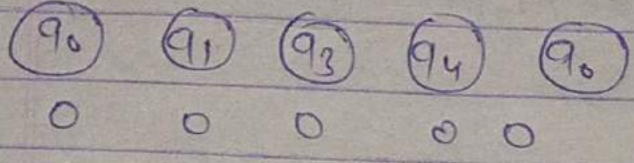
States	a	b	outputs
$\rightarrow q_0$	q_1	q_2	0
q_1	q_2	q_3	0
q_2	q_3	q_4	1
q_3	q_4	q_4	0
q_4	q_0	q_0	0

Soln a) a a b a b

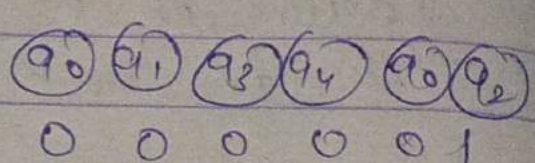
q_0 q_1 q_2 q_4 q_0 q_2

0 0 0 0 0 1 $\Rightarrow$ 001001

b) a b bb



c) a b a bb



→ Mealy Machine

$M = \{Q, \Sigma, \Delta, \delta, \lambda, q_0\}$ where

$z(t) = \lambda(q(t), x(t))$ ↳ Present I/P

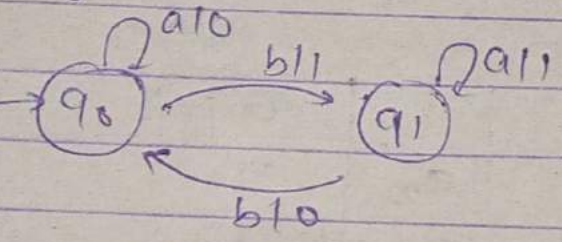
↓ Present state

$\delta : Q \times \Sigma \to Q$

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

- it also depends on the input. & if we give input of n length then output is also of n length.

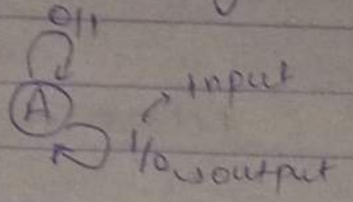
eg



Current State	Next state			
	Input a		Input b	
	state	output	state	output
q_0	q_0	0	q_1	1
q_1	q_1	1	q_0	0

Q Construct a mealy machine that produces the 1's complement of any binary input string

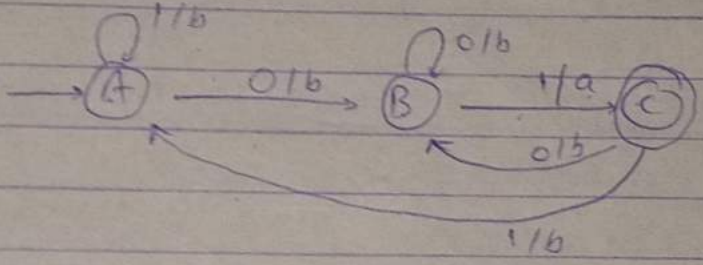
Sol"



Q Construct a mealy machine that prints 'a' whenever the sequence '01' is encountered in any input binary string.

Sol"

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



~~1 0 1 1 0~~

0 1 1 0

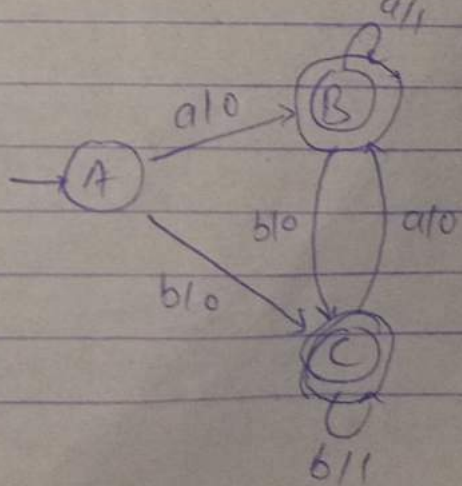
b a b b

Q Design a mealy that accepting the language consisting of strings from Σ^* where $\Sigma = (a, b)$ and the strings should end with either aa or bb.

Sol"

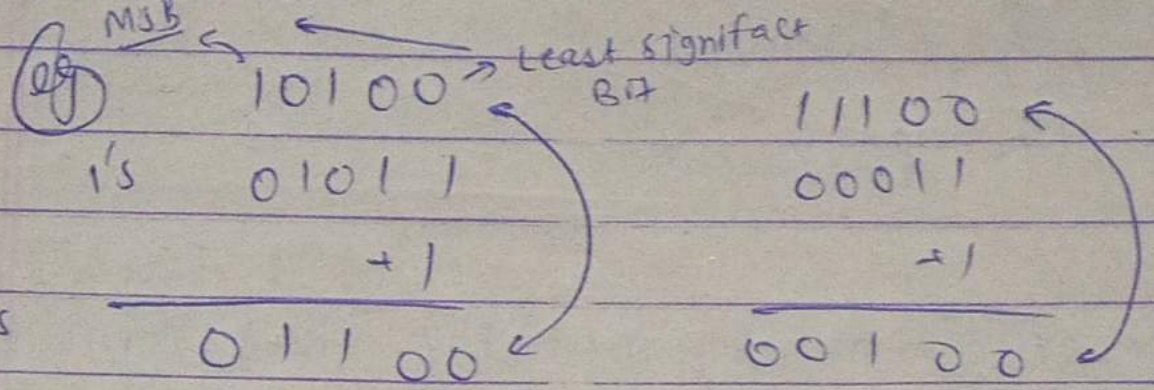
aa - 1

bb - 1



Q Construct a mealy machine that gives 2's complement of any binary input. (Assume last bit is neglected)

So 2's complement = 1's complement + 1



LSB → MSB

Move left to right when we saw first 1 then write as it is and rest bit is being complement before 1 write down as it is.

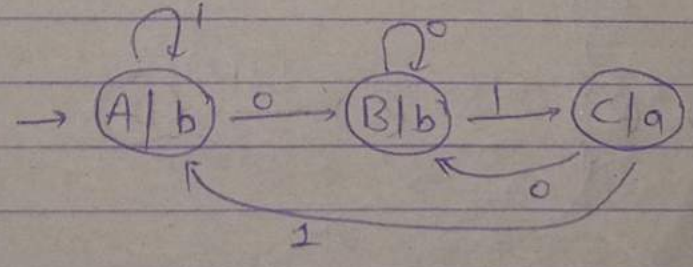
Diff b/w Mealy & Moore Machine

Mealy	Moore
• Output depends on present state and present input.	• depends on present state only
• I/P string of length n → output string is of length n	• I/P → n. then output n+1 string
• $\lambda : Q \times \Sigma \rightarrow \Delta$	• $\lambda : Q \rightarrow \Delta$
• Mealy output is Asynchronous • (faster) as compare to Moore	• Output is synchronous with clock.
$\leq$ x and y states / output	(x y) → no. of states at max.

Conversion of Moore to Mealy Machine

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

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

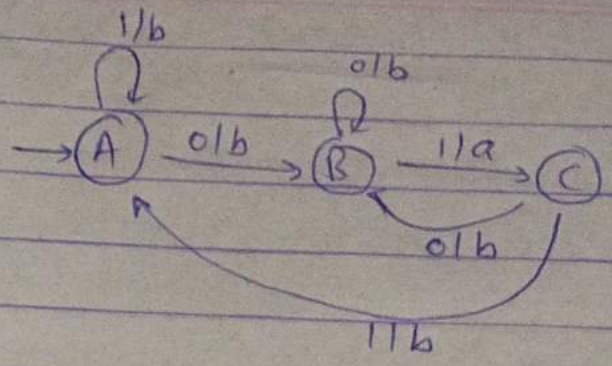


State	0	1	output
→ A	B	A	b
B	B	C	b
C	B	A	a

NOTE: No. of States were ~~state~~ same if we convert Moore → Mealy, it can be less but not more. (atmost)

Mealy

Transition Diagram:

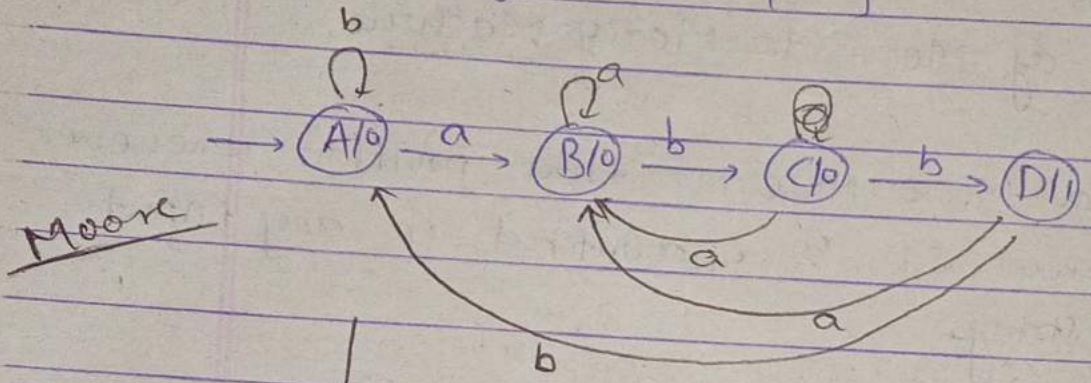


Transition Table:

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

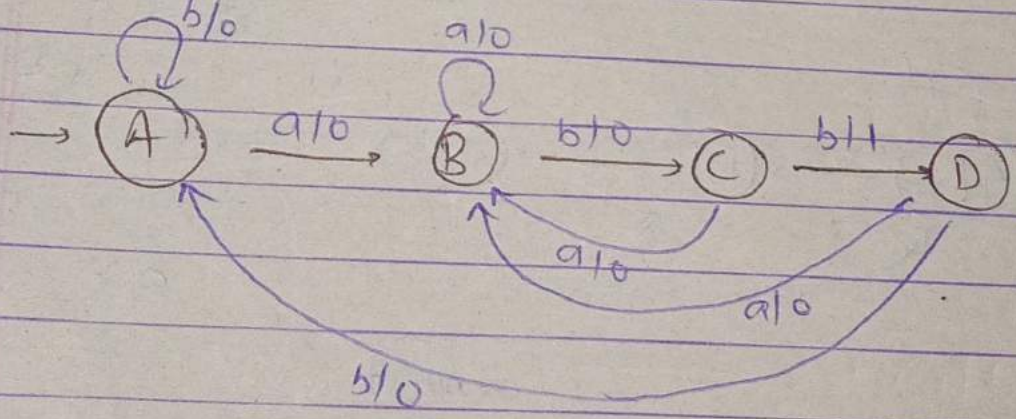
Q Sequence 'abb' of any input string over {a, b}

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



Moore

Mealy



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

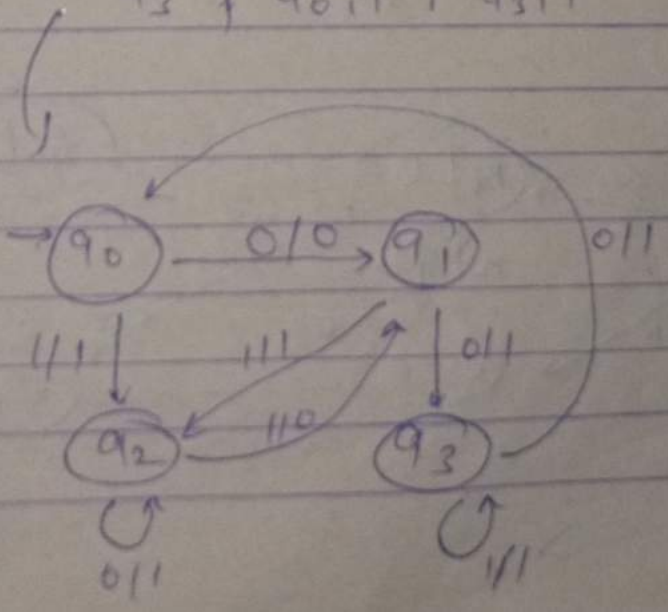
⇒ Moore:

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

State	0	1	output
→ 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

Mealy

state	0	1
→ q_0	$q_1, 0$	q_2, ϕ
q_1	$q_3, 1$	$q_2, 1$
q_2	q_2, ϕ	$q_1, 0$
q_3	$q_0, 1$	$q_3, 1$



Mealy to Moore Machine

No of states increases when we convert Mealy → moore

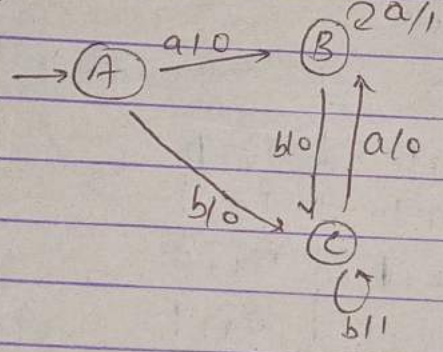
- if there is any conflict occurs on input then create a duplicate state and proceed,

m state, n output ← mealy

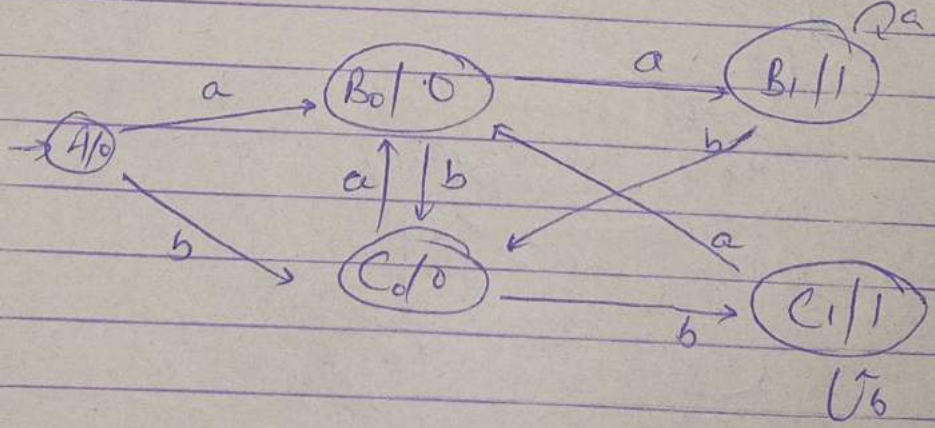
↓

Max state will be m×n ← Moore

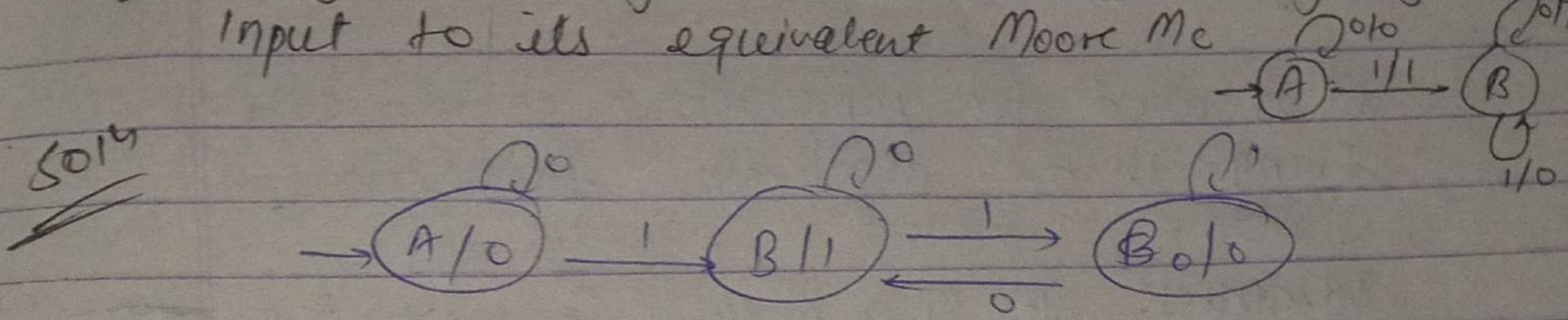
Q Mealy Mc that prints 1 whenever the sequence aa or bb is encountered in any input binary string from Σ^* where $\Sigma = \{a, b\}$



Soln



Q Mealy Mc that give the 2's complement of any binary input to its equivalent Moore Mc



Regular Expression

It is used for representing certain sets of strings in an algebraic fashion.

Identities: It is a method to represent a R. language that accepts FA

collection of strings

Prop:
- Null string ($\in$), Empty (ϕ)
- $R_1 \cup R_2$ - union
- $R_1 R_2 \rightarrow$ concatenation
- R^*

Q. Starting & ending with diff symbol

Soln $a(a+b)^*b + b(a+b)^*a)$

Q. Any no of zero represents

Soln $(00)^*\varepsilon + (00)^*0$

$(00)^*$ → Even

$(00)^*0$ → odd no of 0

Identities.

- $\phi + R = R$
- $\phi R + R\phi = \phi$
- $\varepsilon R = R\varepsilon = R$
- $\varepsilon^* = \varepsilon$ & $\phi^* = \varepsilon$
- $R + R = R$
- $R^* R^* = R^*$
- $RR^* = R^* R$
- $(R^*)^* = R^*$
- $(PQ)^* P = P(QP)^*$
- $\varepsilon + RR^* = \varepsilon + R^* R = R^*$
- $(P+Q)^* = (P^* Q^*)^* = (P^* + Q^*)^*$
- $(P+Q)R = PR + QR$ & $R(P+Q) = RP + RQ$

Arden's Theorem:

if P, Q are Regular expression over Σ and P doesn't contain $\in$ then equation in R given by $R = Q + RP$ has a unique soln i.e. $R = QP^*$.

$$R = Q + RP$$

$R = Q + (Q + RP)P$ {putting value of R}

$R = Q + QP + RP^2$ (again)

$R = Q + QP + QP^2 + QP^3 + \ldots$

$= Q(\varepsilon + P + P^2 + \ldots P^{n+1})$

$R = \underline{QP^*}$ proved

Q Design R.E over $\Sigma\{a,b\}$

a) Accepting string of length exactly 2

Sol^{n} $L_1 = \{aa, ab, ba, bb\}$

$R = aa + ab + ba + bb$

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

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

b) length atleast 2

$L_1 = aa, ab, ba, bba, abb \ldots$

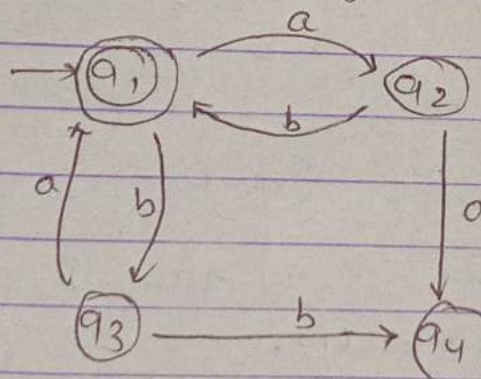
$R_1 = (a+b)(a+b)(a+b)^*$

c) Atmost length 2

$L_1 = (\varepsilon, a, b, aa, bb, ba, ab\}$

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

Q Find R.E for following DFA



Sol^{n} Step 1: write down all incoming transitions

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

$q_2 = q_1 a$ — (ii)

$q_3 = q_1 b$ — (iii)

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

Step 2: Reduce state by putting values.

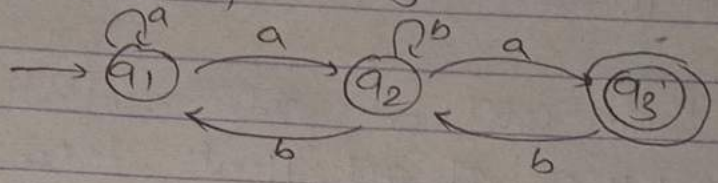
$q_1 = \epsilon + q_2 b + q_3 a$ (put values of q_2 & q_3)

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

$= \epsilon + \{q_1 (ab + ba)\}$ $\quad \{ R = Q + RP \} \Rightarrow R = QP^*$

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

Q. Find R.E for following NFA



Sol^n $q_1 = \epsilon + q_2 b + q_1 a$ — (i)

$q_2 = q_1 a + q_3 b + q_2 b$ — (ii)

$q_3 = q_2 a$

putting the given value in eq (3) (final state) we get

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

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

$= q_1 aa + q_2 ba + q_3 ba$ — (iii)

from (ii)

$q_2 = q_1 a + q_3 b + q_2 b$

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

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

$q_2 = q_1 a (b + ab)^*$ — (iv) $\quad R = QP^*$

$q_1 = \epsilon + q_1 a + q_2 b$ (putting value of q_2)

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

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

$q_1 = (a + a(b+ab)^* b)^*$ — (5)

final state (q3)

$q_3 = q_2 a$

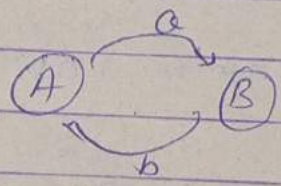
$= q_1 a (b+ab)^* a$ put value of q_2 (4)

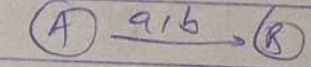
$= (a + a(b+ab)^* b)^* a(b+ab)^* a$

Note:- if there will be multiple final states then take union of all final states $R = R_1 + R_2 + \dots$

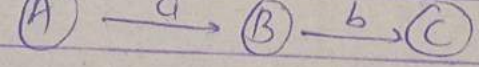
Conversion of R.E → finite Automata

- $(a+b)$

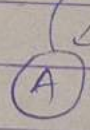


OR 

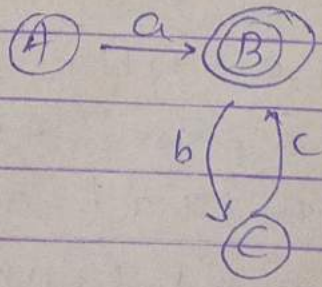
- $(a \cdot b)$



- a^*

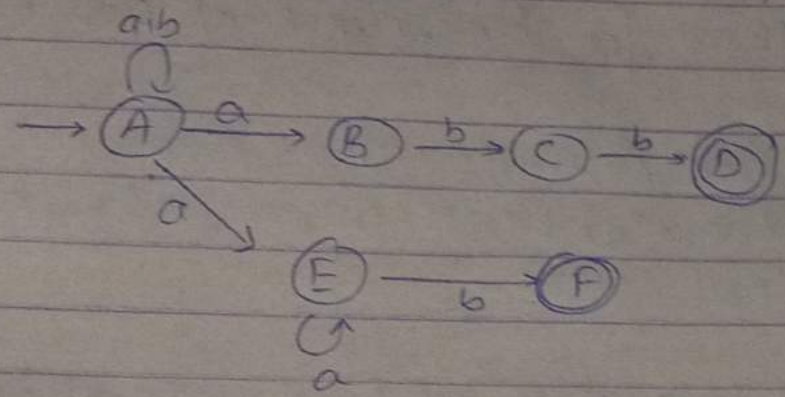


- $a(bc)^*$

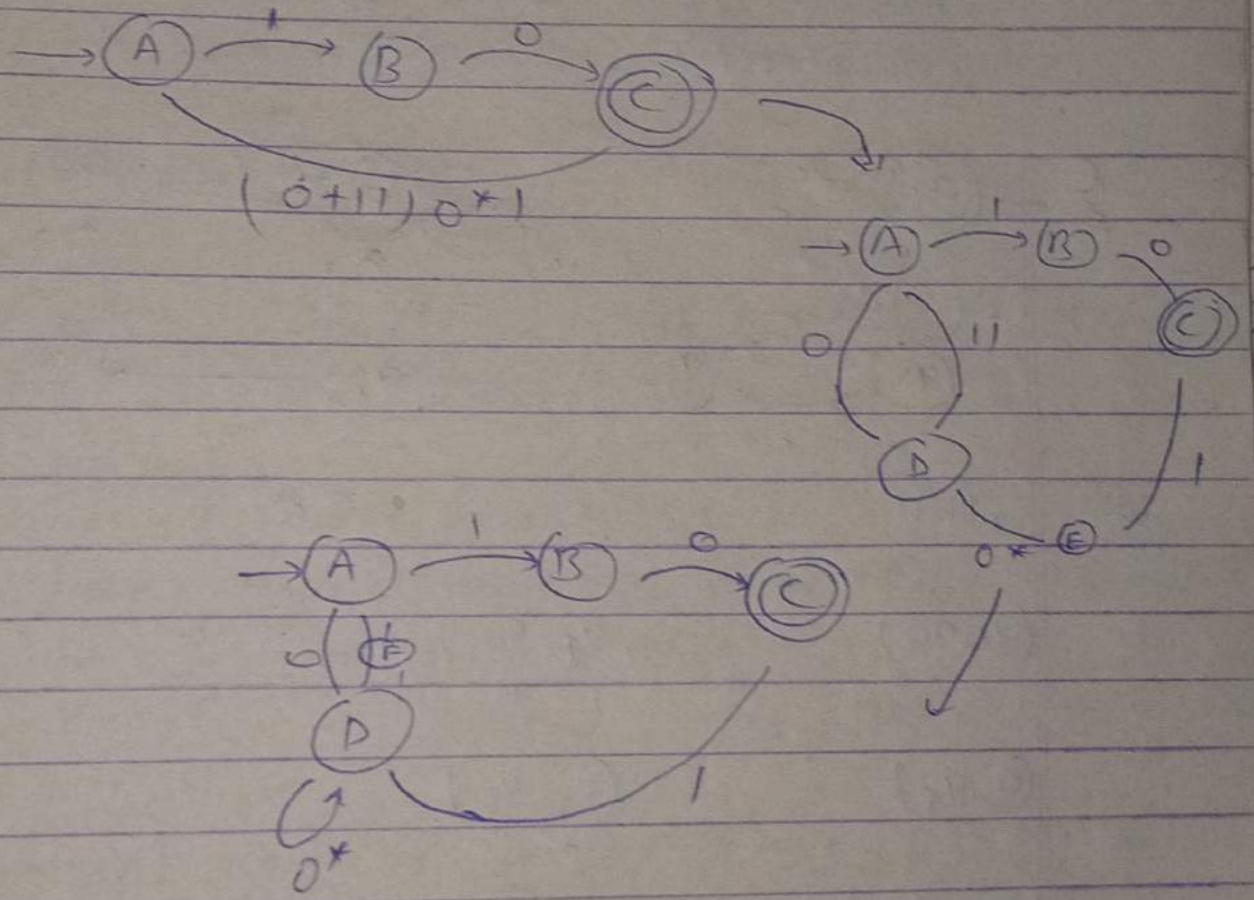


• Convert R.E to equivalent F.A $(a|b)^*(abb|a^*b)$

Sol^n

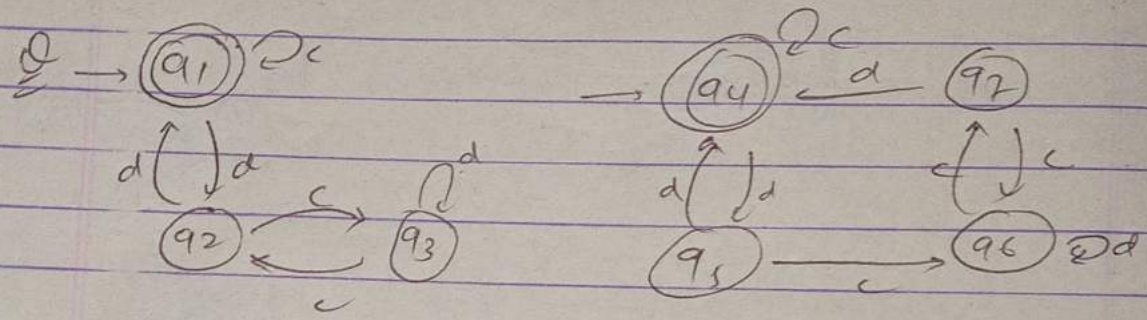


Q $10 + (0+11)0^*1$



Conditions for Equivalence of two FA

- The two Automata aren't equivalent if for a pair (q_a, q_b) one is intermediate state and other is final state
- If initial state is final state of one Automation then second Automation also initial state must be final state for them to be equivalent



So	states	c	d
	(q_1, q_4)	q_3, q_4 F F	q_2, q_5 I I
	(q_2, q_5)	(q_3, q_6) I I	q_1, q_4 F F
	(q_3, q_6)	(q_2, q_7) I I	q_3, q_6 I I
Newly formed	(q_2, q_7)	(q_3, q_6) I I	(q_1, q_4) F F

Closure prop of Regular language

- Union ($L_1 \cup L_2$)
- Concatenation ($L_1 \cdot L_2$)
- Closure (*) L^*
- complementation $\bar{L} = \Sigma^* - L$
- Intersection $L_1 \cap L_2$
- Difference $= L_1 - L_2$
- Reversal L^R
- Homomorphism
- INIT
- Reverse Homomorphism
- Quotient operation
- substitution.

Note:- Regular language is closed under all operations except infinite union

→ R.L closed under Reversal (R^L)
- Make initial state as final state & vice-versa
- Reverse the dirn of edge
- Remove unnecessary states / unReachable state

→ Quotient operation → left quo. (cut prefix)
→ Right quo. (cut suffix)

$\frac{L_1}{L_2} = \{ x \mid xy \in L_1 \text{ for some } y \in L_2 \}$ Right quo. (cutting in Right)

$\frac{L_1}{L_2} = \{ y \mid xy \in L_1 \text{ for some } x \in L_2 \}$ left quo. (cutting in left)

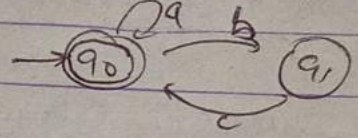
→ INIT operation (initial / prefix)

set of all prefix of $\omega \in L$

Eg $L = \{abiba\}$

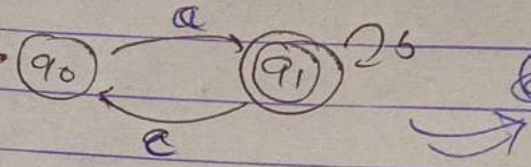
$\Rightarrow \{\epsilon, a, ab, abi, biba\}$

FA → R.E

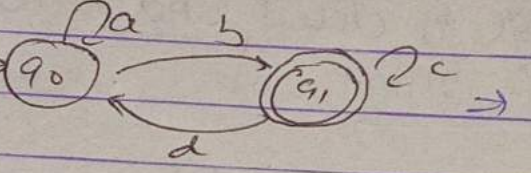
Q 

$(a^* + (bc)^*)^*$

$(a+bc)^*$

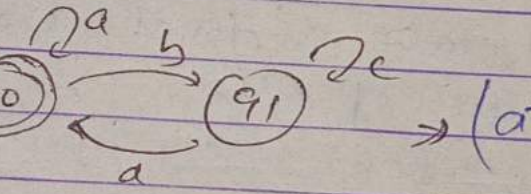
Q 

$a(b+(a)^*$

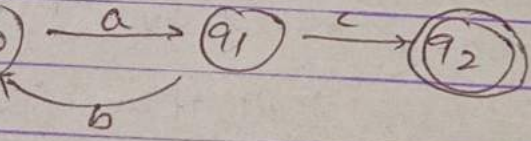
Q 

$\Rightarrow a^*b(c^* + (da^*b)^*$

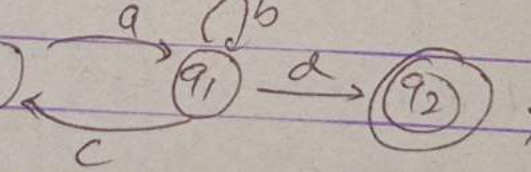
$a^*b(c+da^*b)^*$

Q 

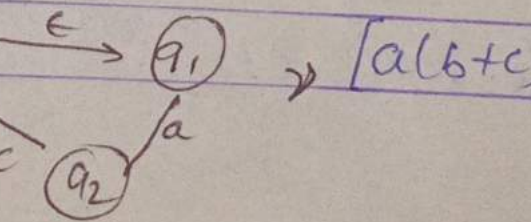
$\Rightarrow (a^* + (bc^*d)^*$

Q 

$\Rightarrow a(ba)^*c$

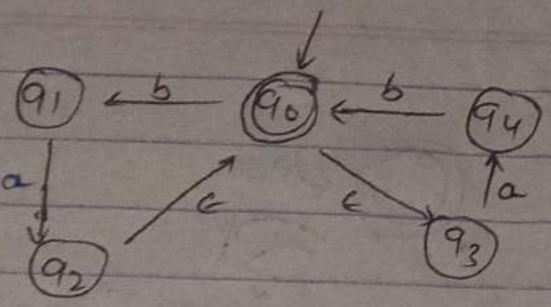
Q 

$\Rightarrow a(b+(a)^*d$

Q 

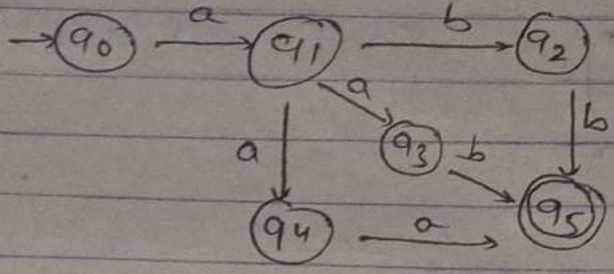
$\Rightarrow [a(b+c)]^*$

Q



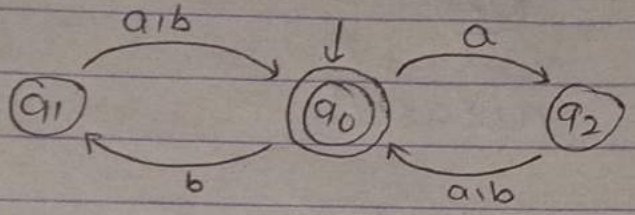
$\Rightarrow (ba+ab)^*$

Q



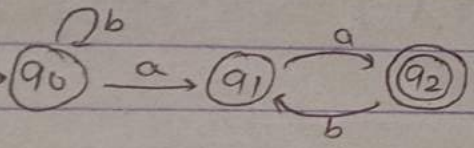
$\Rightarrow a.(bb+ab+aa)$

Q



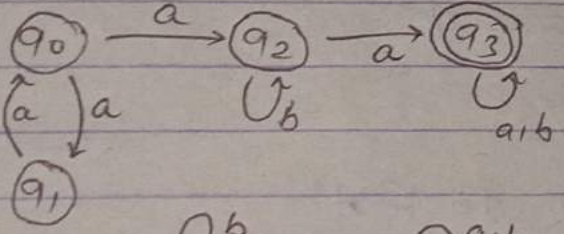
$\Rightarrow [(a+b)(a+b)]^*$

Q



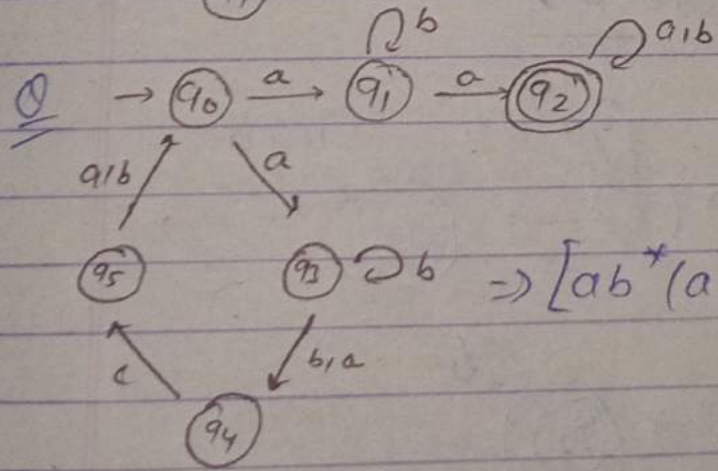
$\Rightarrow b^*aa(ba)^*$

Q

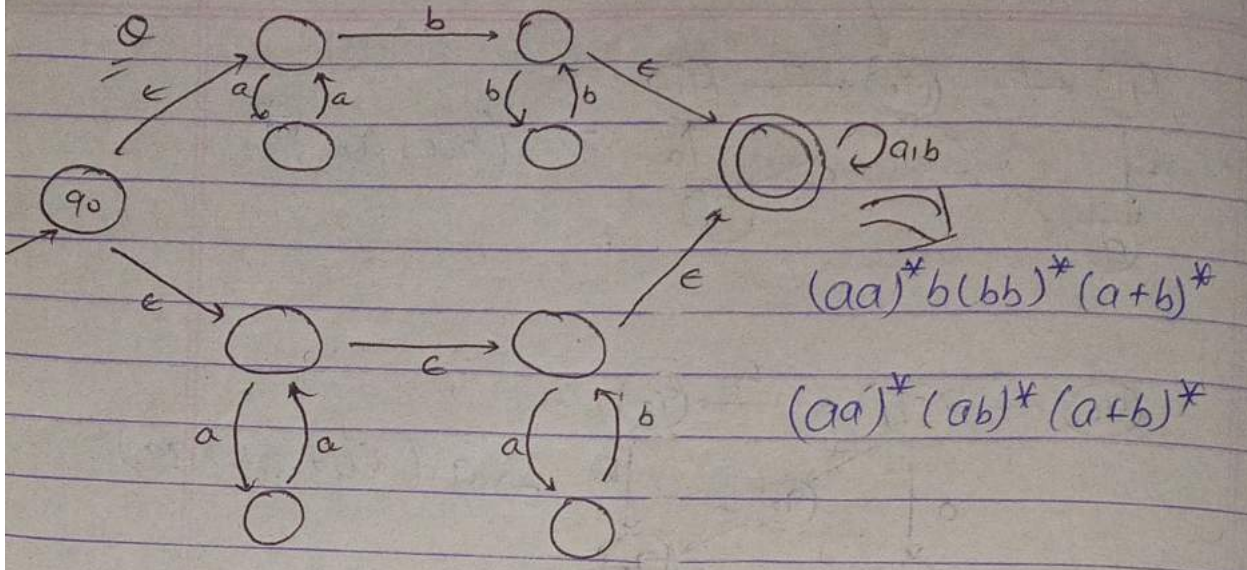


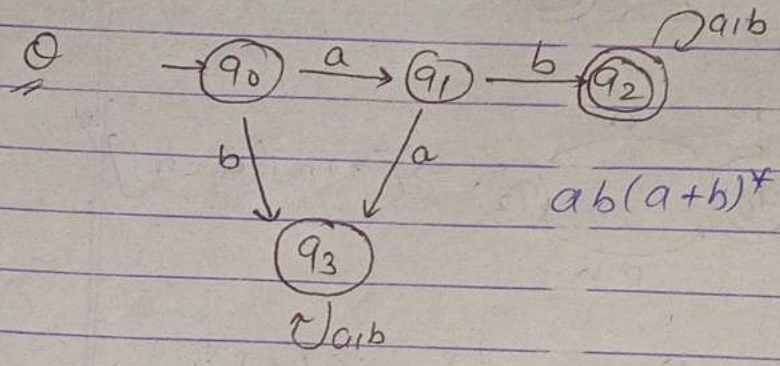
$\Rightarrow (aa)^*ab^*a(a+b)^*$

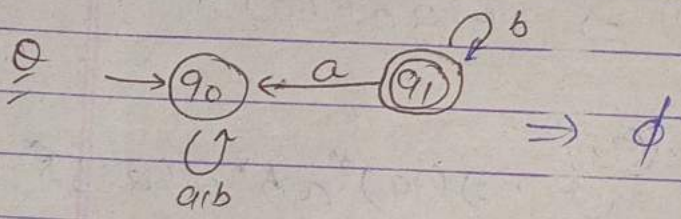
Q



$\Rightarrow [ab^*(a+b)c(a+b)]^*[ab^*a(a+b)^*]$

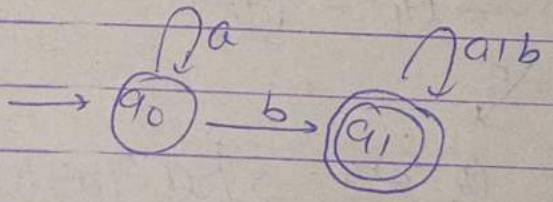




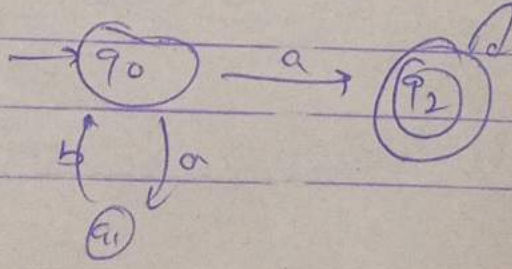


Regular Expression to F.A

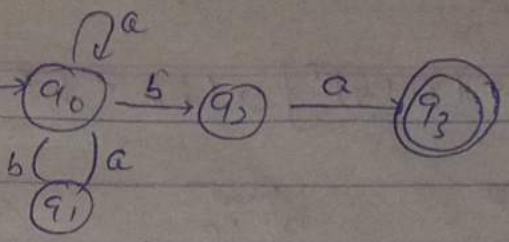
Q $a^*b(a+b)^*$



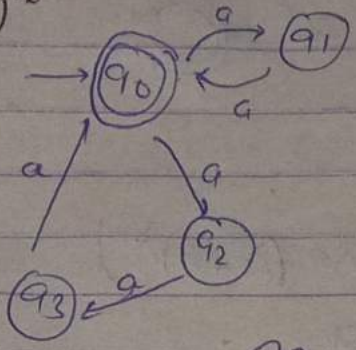
Q $(ab)^*ab^*$



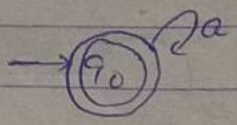
Q $(a+ba)^* ba$



Q $(aa+aaa)^*$

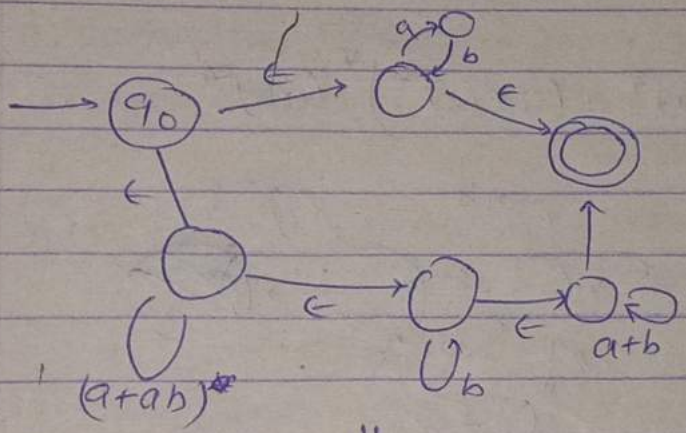


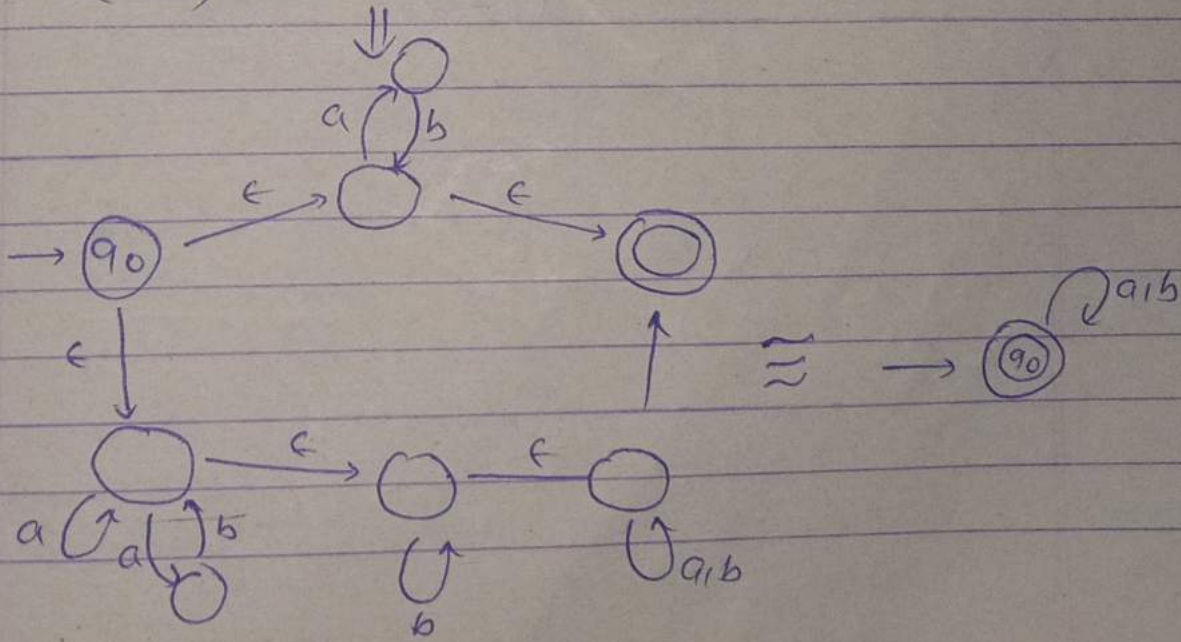
Q $(a+aaaa)^*$



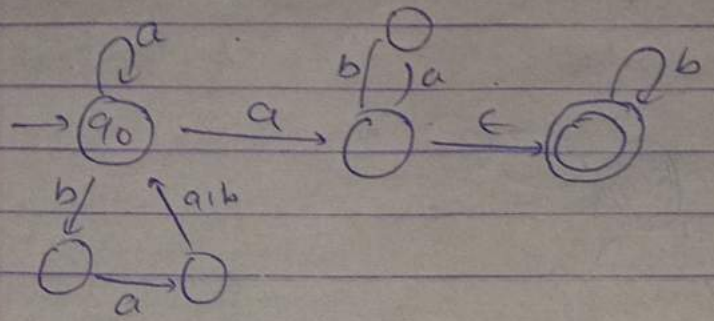
** Q $(ab)^* + (a+ab)^* b^* (a+b)^*$

an this use Null transition to solve this prblm

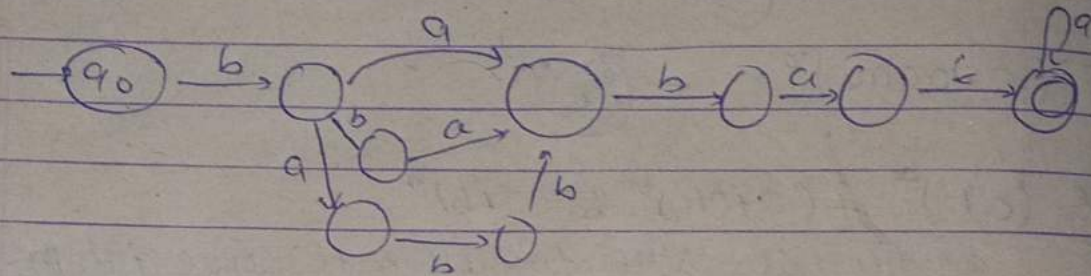




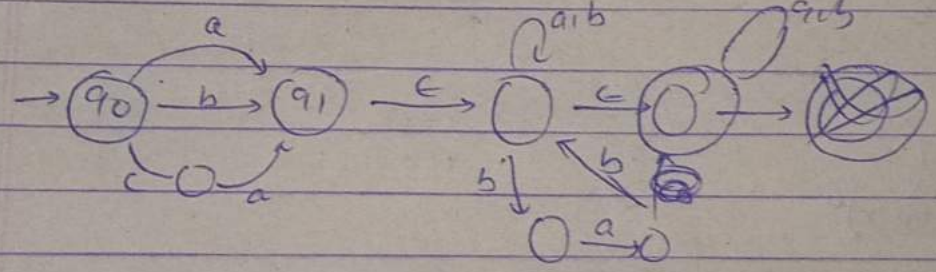
Θ $[a+ba(a+b)^*]a(ba)^*b^*$



** Θ $b(a+ba+abb)(ba+a+b)^*)$



Θ $(a+b+ca)((bab)^* + (a+b)^*)^*(ab)^*$



Grammar

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

V = set of variables / Non-Terminal symbol (capital letter)

T = set of Terminal Symbol (small letter)

P = Production Rules for Terminal & Non-Terminal

S = start symbol

Grammar contains a set of Rules to construct the sentence in a language

A production rules has form $\alpha \to \beta$ where α & β are strings on VUT and atleast one symbol $\in V$

eg $G = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to a, B \to b\})$

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

$T = \{a, b\}$

S = start symbol

$P = S \to AB, A \to a, B \to b.$

Chomsky Hierarchy

Acc^{n} to Noam chomsky, There are 4 types of grammar

Grammar Type	Grammar Accepted	language Accepted	Automation	closed under
Type 0	unrestricted grammar	R.E.L	Turing Machine	union, intersection concatenation Kleen closure
Type 1	context sensitive grammar	C.S.L	LBA	union, intersection complementation concatenation, Kleen
Type 2	CFG	C.F.L	PDA	union, Kleen Concatenation
Type 3 ↓ single non-Terminal in LHS	R.G	R.L	F.S.A	union, intersection complementation concat, Kleen

Language Accepted by

- F.A = R.L
- PDA = CFL, RL
- LBA = CSL, CFL, RL
- Tm = REL, CSL, CFL, RL

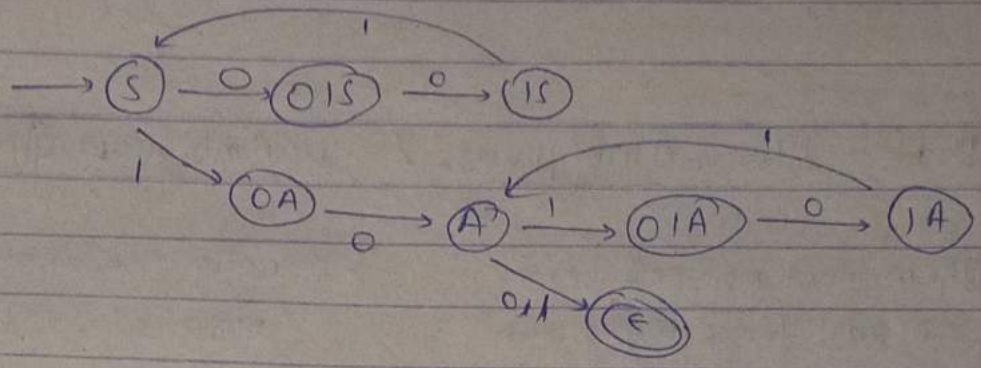
T.M > LBA > PDA > FM

$CFG^{C} = CSL$

Q Regular Grammar → F.A , $S \to 00|S / 10A$

$A \to 101A / 01$

Soln



Note: for left Terminal first reverse then make NFA then change initial → final, final → initial state then Reverse the dirn of edge

Regular Grammar

Right Linear

$A \to xB$

$A \to x$

$\{ A, B \in V, x \in T \}$

Left Linear

$A \to Bx$

$A \to x$

$\{ A, B \in V, x \in T \}$

Derivation: Sequence of steps followed to generate a string from a grammar

Types: a) Right most Derivation

b) left " "

Parse Tree: It is an ordered tree in which node are labelled with left side of production and in which child of node represents corresponding right side.

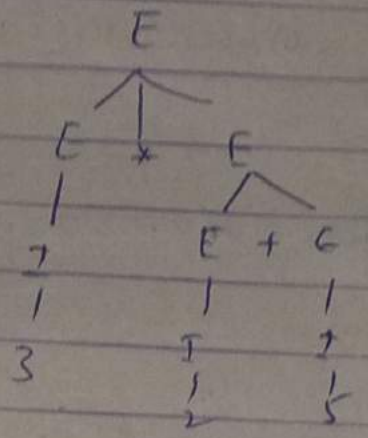
Diff b/w Ambiguous & UnAmbiguous Grammar

Ambiguous	UnAmbiguous
→ There exist more than one LMD/RMD for a string that belong to grammar.	• Exactly one derivation (unique LMD/RMD)
• LMD & RMD represents diff parse trees	• LMD & RMD represent same parse tree
• More than one parse tree for a string	• unique parse tree

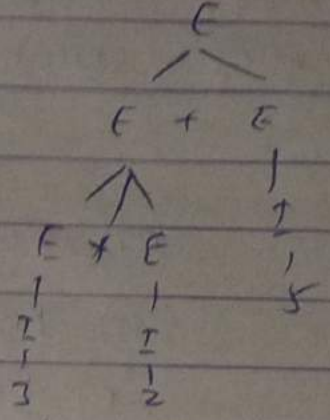
Q $E \to E+E$, $E \to E*E$, $E \to (E)$, $I \to E|0|1|2|\ldots 9$

Sol^n for the string "3 * 2 + 5" the above grammar can generate two parse tree by MSD

I



II



Since, there are two parse tree for a single string the grammar is Ambiguous

→ UnAmbiguous

$X \to aX$ | Right

$X \to Xa$ Left most

Pumping lemma

App.→ It is used to prove that a language is Not Regular.

let L be a Regular language and has pumping length p such that string w, where w is divided into xyz

$|w| \geq p$

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

b) $|y| > 0$

c) $|xy| \leq p$

the string xy^iz is also a language L

Recursive CFL: Generate infinite no. of strings

Non- " " : Generate finite no of strings.

Context free language / Grammar

it is a language which is generated by some CFG. The set of all CFL is identical to the set of language accepted by PDA

CFG is defined by 4 tuples as $G = V, \Sigma, P, S$

(capital letters) V = set of variables / Non-Terminal symbols
(lowercase letters) Σ = set of Terminal symbol $(V \cap T = \phi)$
P = Production Rule
S = start symbol

CFG has Production Rule of the form $A \to \alpha$ where
$\alpha = \{V \cup \Sigma\}^*$ and $A \in V$

- it is a formal Grammar which is used to generate all possible patterns of string in a given formal language

Q Language that generates equal no of a & b in the form $a^n b^n$ the CFG will be defined as $G = \{(S, A), (a, b), (S \to aAb, A \to aAb/\epsilon)\}$

Sol^n

$S \to aAb$
$\to aaAbb$
$\to aaaAbbb \quad (A \to \epsilon)$
$\to a^3 b^3 \to a^n b^n$

Application of CFG

→ used in Natural language processing
→ used in speech Recognition.

Simplification of CFG

In CFG, sometimes all production Rules and symbols are not needed for the derivation of strings. There may also be some NULL production and unit production. Elimination of these productions and symbols is called simplification of CFG.

It consists of 3 steps:

a) Reduction of CFG / Removal of useless symbol
b) Removal of unit production
c) " " Null "

a) Removal of useless symbol : if it doesn't appear on the R.H.S of production & it doesn't take part in the derivation of any string.

(eg) $S \to aaB \mid bA \mid aaS$
$A \to aA$ ✗
$B \to ab \mid b$
$C \to ad$ ✗

⇒

New Production

$S \to aaB \mid aaS$
$B \to ab \mid b$ &

Here variable C is useless bcz it never occur in derivation string. so we can eliminate. Now $A \to aA$ is also useless bcz there is no way to terminate, to remove this useless production $A \to aA$, we will first find all the variables which will never lead to a terminal string such as variable A. Then we'll remove all the production in which variable B occurs.

→ Removal of Null

The production of type $S \to \epsilon$ are called ϵ production. These type of production can only be removed from the grammar that do not generate ϵ

Step1: find all nullable non-terminal which derive ϵ

Step2: For each production $A \to a$, construct all production $A \to x$ where x is obtained from a by removing 1 or more non-terminal from step 1

Step3: Combine Result of step 2 with original production & remove ϵ production.

(8) $S \to XYX$, $X \to 0X \mid \epsilon$ $Y \to 1Y \mid \epsilon$

Sol^n $X \to \epsilon$. $Y \to \epsilon$

$S \to XYX$

→ if first X at RHS is ϵ then $S \to YX$

→ Wlg if last X ... " " $S \to XY$

if $y \in \epsilon$ then $S \to XX$

if Y & X are ϵ then $S \to X$

if both $X \to \epsilon$ then $S \to Y$

Now new production is $S \to XY \mid YX \mid XX \mid X \mid Y$

Now, $X \to 0X \mid \epsilon$

$X \to 0$

$\therefore X \to 0X \mid 0$ similarly $Y \to 1Y \mid 1$

finally CFG with Removal of ϵ production as,

$S \to XY \mid YX \mid XX \mid X \mid Y$
$X \to 0X \mid 0$
$Y \to 1Y \mid 1$

→ Removal of Unit Production.

an this one non-terminal gives another Non-terminal

Step 1: To Remove $X \to Y$, add production $X \to a$ to the grammar rule whenever $y \to a$ occurs in the grammar

Step 2: Delete $X \to Y$ from the grammar

Step 3: Repeat with all unit production

(Q) $S \to 0A \mid 1B \mid C$
$A \to 0S \mid 00$
$B \to 1 \mid A$
$C \to 01$

Sol^{n} ~~S → C~~

~~B → A~~

Here S → 01 & B → 0S|00

finally we can write CFG without unit production as

S → 0A|1B|01

A → 0S|00

B → 1|0S|00

C → 01

Q Construct a CFG for language L = {wcw^R where w ∈ (a,b)*}

Sol^{n} L = {aacaa, abcba, abbcbba ---}

The grammar is

S → aSa
S → bSb
S → c

Q $L = a^n b^{2n}$ where $n \geq 1$

L = {abb, aabbbb, aaabbbbbb, ---}

S → aSbb
S → abb

Q $a^n b^{n+2}, n \geq 0$

Sol^n $S \to aSb \mid bb$

Q $a^{2n+2} b^n, n \geq 0$

Sol^n $S \to aaSb \mid aaa$

Q $\{w \mid n_a(w) = n_b(w)\}$

Sol^n $S \to aSb \mid bSa \mid SS \mid \lambda$

Q $a^m b^m c^n, m, n \geq 0$

Sol^n $S_1 \to aS_1b \mid \lambda$

$C \to cC \mid \lambda$

$S \to S_1C$

Q $a^{2n} b^n, n \geq 0$

$S \to aaSb \mid \lambda$

Q $a^m b^n, m > n, n \geq 0$

$S_1 \to aS_1b \mid \lambda$

$A \to aA \mid a$

Q Palindrome : $\underbrace{ww^R}_{\text{even}} \cup \underbrace{w(a+b)w^R}_{\text{odd}}$

$S \to aSa \mid bSb \mid a \mid b \mid \lambda$

CYK Algo: Checks whether a string is valid member of CFL, it is applicable on CNF only

$A \to BC$

$A \to a$

Chomsky Normal form

A CFG is in CNF if all production Rules which is present on RHS should either be

a) start symbol generating ϵ

eg $A \rightarrow \epsilon$

b) A non-terminal generating two non-terminals.

eg $S \rightarrow AB$

c) A non-terminal generating a terminal

eg $S \rightarrow a$

Steps for Converting CFG into CNF

Step 1: Eliminate start symbol from the RHS. if the start symbol T is at the RHS of any production create a production as

$S1 \rightarrow S$ where $S_1 \rightarrow$ new start symbol

Step 2: Remove, Null, unit and useless symbol

Step 3: Eliminate terminals from the RHS of the production if they exists with other non-terminals / terminals.

eg $S \rightarrow aA$ can be decomposed as

$$\{ S \rightarrow RA \}$$
$$\{ R \rightarrow a \}$$

Step4: Eliminate RHS with more than two non-terminal

eg $S \to ABS$ can be decomposed as

$$\{ S \to RS \quad R \to AB. \}$$

Q Convert CFG to CNF. $S \to a \mid aA \mid B$
$A \to aBB \mid \epsilon$
$B \to Aa \mid b$

Sol^n

Step1 $S_1 \to S$ // new production
$S \to a \mid aA \mid B$
$A \to aBB \mid \epsilon$
$B \to Aa \mid b$

Step2 Now, $A \to \epsilon$ // Remove
$S_1 \to S$, $S \to a \mid aA \mid B$ $B \to aBB$, ~~$B \to a \mid b$~~
$B \to Aa \mid b \mid a$

// Remove unit production
~~$S \to S$~~ $S_1 \to a \mid aA \mid Aa \mid b$~~a~~
$S \to a \mid aA \mid Aa \mid b$
$A \to aBB$
$B \to Aa \mid b \mid a$

Step 3. // not follows CNF Rules

$S_1 \to aA \mid Aa$
$S \to aA \mid Aa$
$A \to aBB$
$B \to Aa$

// Replace $X \rightarrow a$

$S_0 \rightarrow a \mid XA \mid AX \mid b$

$S \rightarrow a \mid XA \mid AX \mid b$

$A \rightarrow XBB$

$B \rightarrow AX \mid b \mid a$

$X \rightarrow a$

// $R \rightarrow XB$

$S_0 \rightarrow a \mid XA \mid AX \mid b$

$S \rightarrow a \mid XA \mid AX \mid b$

$A \rightarrow RB$

$B \rightarrow AX \mid b \mid a$

$X \rightarrow a$

$R \rightarrow XB$

So, Here the given grammar is in CNF.

Griebach Normal form (GNF)

A CFG is in GNF if all the production Rules satisfy the following conditions:

a) A start symbol generating ϵ eg $S \to \epsilon$

b) A non-terminal generating a terminal, eg. $A \to a$

c) A non-terminal generating a terminal which is followed by any no. of non-terminals.

eg $S \to aASB$
$a\,AB$
$a\,ABC$

(a: Terminal)

Steps for converting CFG into GNF

Step 1. Convert the CFG into CNF, if the grammar is not in CNF then convert it into CNF

Step 2 if the grammar exists left Recursion eliminate it

Step 3: In the grammar, convert the given production Rule into GNF form. if any production Rule in the grammar is not in GNF then convert it

CFG → CNF → left Recursion → GNF conversion.

Q $S \to XB \mid AA$
$A \to a \mid SA$
$B \to b$
$X \to a$

} CNF ✓ L.R X

Soln $A \to SA$ is not in GNF, so substitude $S \to XB \mid AA$ In production Rule

$S \to XB \mid AA$
$A \to a \mid XBA \mid AAA$
$B \to b$
$X \to a$

$S \to XB$, $A \to XBA$ is not in GNF, so substitud $X \to a$

$S \to aB \mid AA$
$A \to a \mid aBA \mid AAA$

~~Pumping lemma for CFL~~

Diff b/w CNF & GNF

• Chomsky Normal form	• Greibach Normal form
• $A \to BC$, $A \to a$ $A, B \in V$, $a \in T$	• $A \to aXAC$ $A \to b$
• used in Membership algo.	• used to convert CFG to PDA
• No of steps required to generate a string of length n is $2\|n\| - 1$	• length n is n
• length of each production Restricted	• Not Restricted
• Derivation obtained from CNF is always Binary Tree	• Not Always.

Pumping lemma for CFL

L is a cfl, there is a pumping length n such that any string $w \in L$ of length $\geq n$ can be written as

$|w| \geq n$

w divided into 5 strings $w = uvxyz$ such that

- $|vxy| \leq n$
- $|vy| \neq 0$
- for all $k \geq 0$ the string $uv^kxy^kz \in L$

Prove that L is not CF using pumping lemma follow the steps (we prove using contradiction)

- Assume that L is context free
- pumping length = n
- All strings longer than n can be pumped $|w| \geq n$
- find a string w in L such that $|w| \geq n$
- Divide $w \rightarrow uvxyz$
- show that $uv^kxy^kz \notin L$ for some k
- show none of these can satisfy all the 3 pumping cond^n at same time
- w can't be pumped (contradiction)

40×5 = 200 → 800

Q find out whether $L = \{a^n b^n c^n \mid n \geq 1\}$ is C.F or not

Soln let L be C.F in which $x^n y^n z^n$

case I let n = 4, v & y contain only one type of symbol $w = uvxyz$

$\underbrace{x}_{u}\underbrace{xx}_{v}\underbrace{xyyyy}_{x}\underbrace{z}_{y}\underbrace{zzz}_{z}$

Now

$uv^k xy^k z$ let $k = 2$

$uv^2 xy^2 z$

$x\ xxxx\ xyyyyzzzzz \Rightarrow x^6 y^4 z^5 \notin L$

case II Either v or y has more than one kind of symbol

let n = 4 $\underbrace{xxxx}_{u}\ \underbrace{yyyy}_{v}\ \underbrace{zzzz}_{z}$ (x, y)

let k = 2 $uv^k xy^k z$

$uv^2 xy^2 z$

$xx\ xxyy\ xxyy\ xyyzzzz \Rightarrow x^4 y^2 x^2 y^5 z^4 \notin L$

$\not\equiv$

$x^n y^n z^n$

Push Down Automata (PDA)

- It is a way to implement a CFG in a similar way we design FA for Regular grammar
- it is powerful than fsm and has more memory

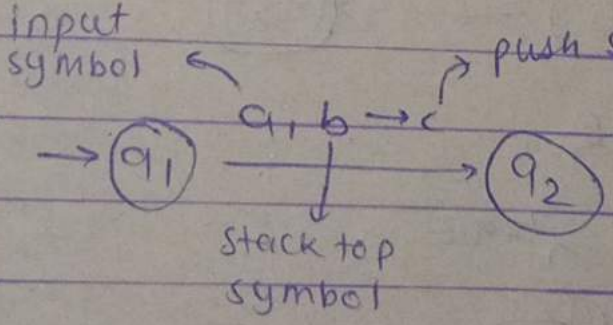
PDA = FSM + stack

- An input tape
- A finite control unit
- A stack with infinite size

$P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

- Q = finite set of states
- Σ = " " " input symbol
- Γ = finite stack alphabet (push into stack)
- δ = transition func
- q_0 = start state
- z_0 = start stack symbol
- F = set of final / accepting state

$\delta(q_1, a, x)$



Instantaneous Description (ID)

ID of a PDA is represented by triple (q, w, s)

- q is the state
- w is unconsumed input
- s is the stack contents

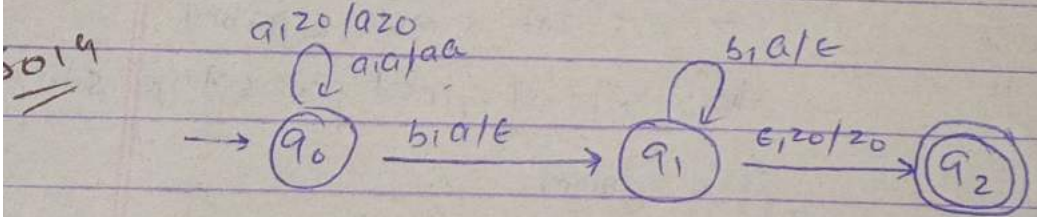
ID is an informal notation of how a PDA computes an input string and make a decision that string is accepted / Rejected.

(⊢) Turnstile Notation is used for connecting pairs of IDs that represent one or many moves of a PDA

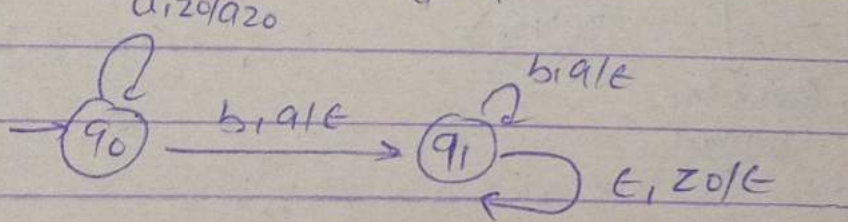
Acceptance of PDA → Final state Acceptability
→ Empty stack ,,

Q Construct PDA that accepts $L = \{a^n b^n, n \geq 1\}$

Sol^n



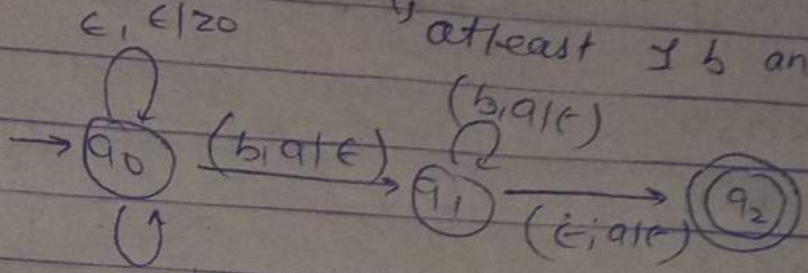
OR



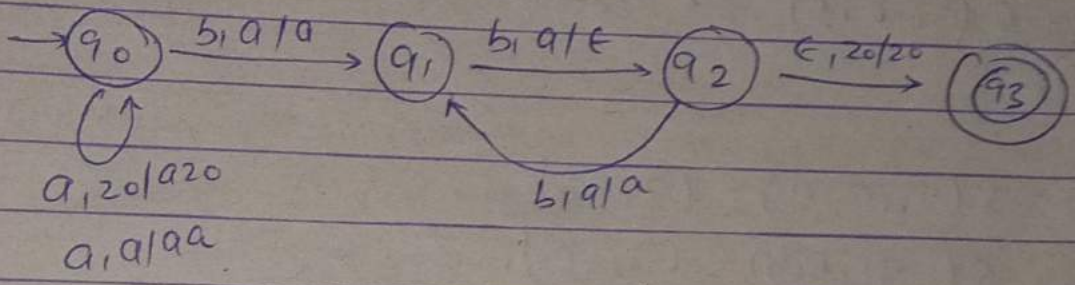
ID ⟹ $\delta(q_0, a, Z_0) = (q_0, aZ_0)$

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

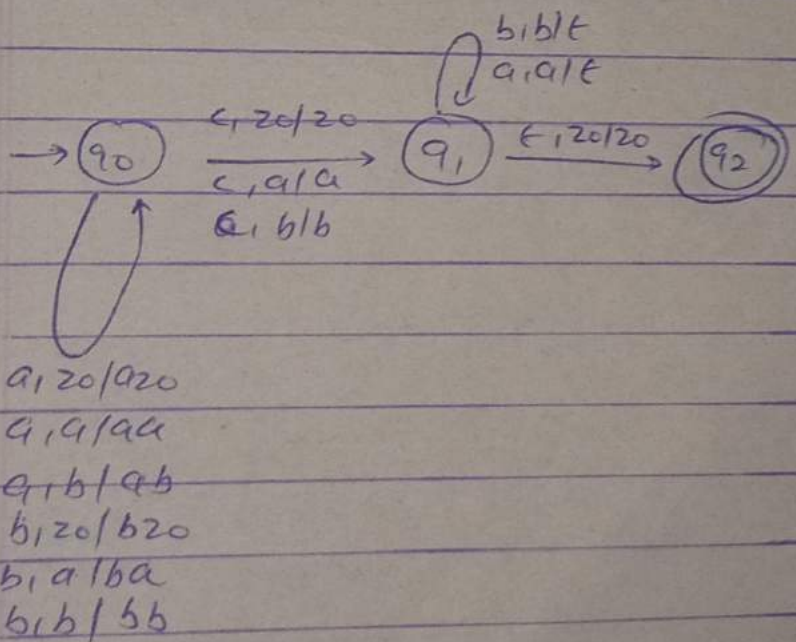
↳ atleast 1 b and 2 a should be there.



Q $L = \{a^n b^{2n} \mid n \geq 1\}$



Q $L = \{w c w^R \mid w \in (a,b)^*\}$



conversion of CFG to PDA

Ex $S \to aSa$

$S \to bSb$

$S \to c$

Write the production Rules & pop the element

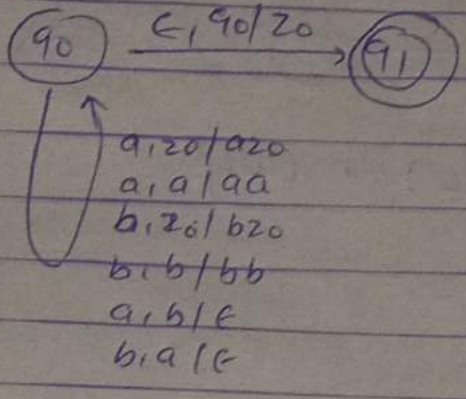
Production Rules

- $\delta(q_0, \epsilon, \epsilon) = (q_0, \epsilon)$ (state; value in stack)
- $\delta(q_0, \epsilon, S) = (q_0, aSa)$
- $\delta(q_0, \epsilon, S) = (q_0, bSb)$
- $\delta(q_0, \epsilon, S) = (q_0, c)$

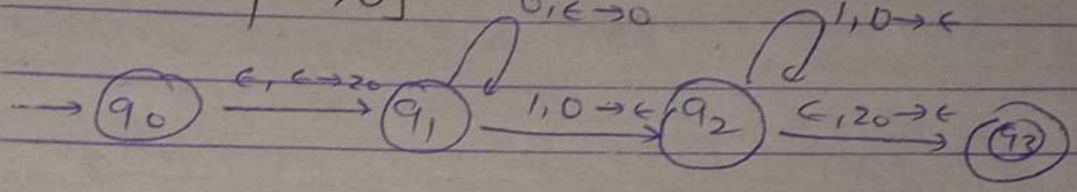
- $\delta(q_0, a, a) = (q_1, \epsilon)$
- $\delta(q_1, b, b) = (q_2, \epsilon)$
- $\delta(q_2, c, c) = (q_3, \epsilon)$

Q Construct a PDA that accepts $L = \{ w | w | a = b \mid w \in (a+b)^* \}$

Sol:



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

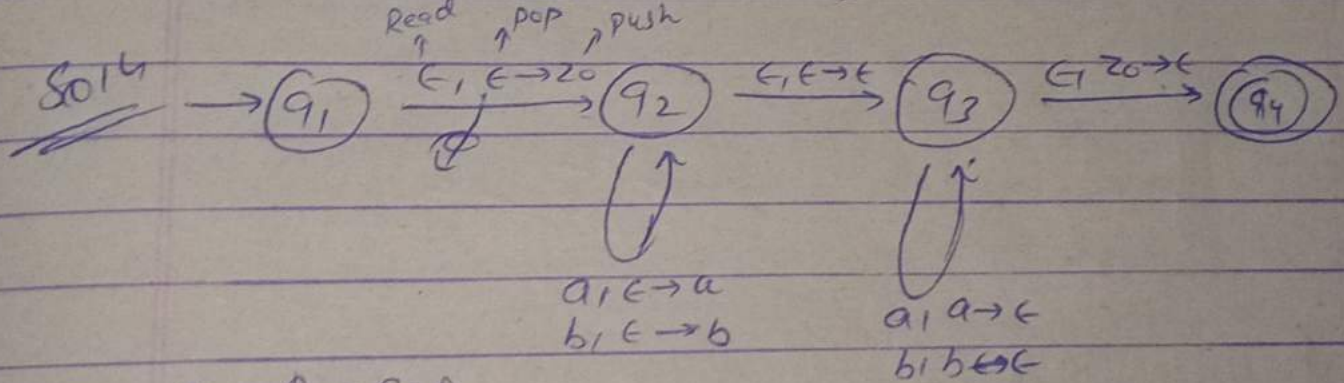


Q Construct a PDA that accepts even palindrome

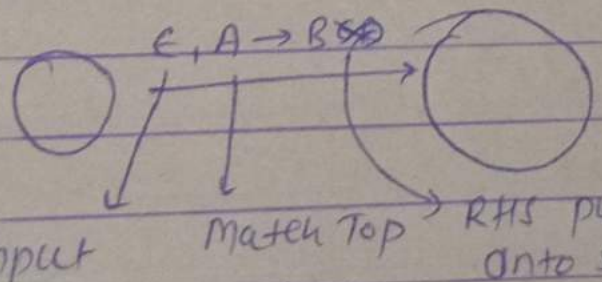
$L = \{ w w^R \cdot / w = (a+b)^* \}$

Sol:

Read, pop, push

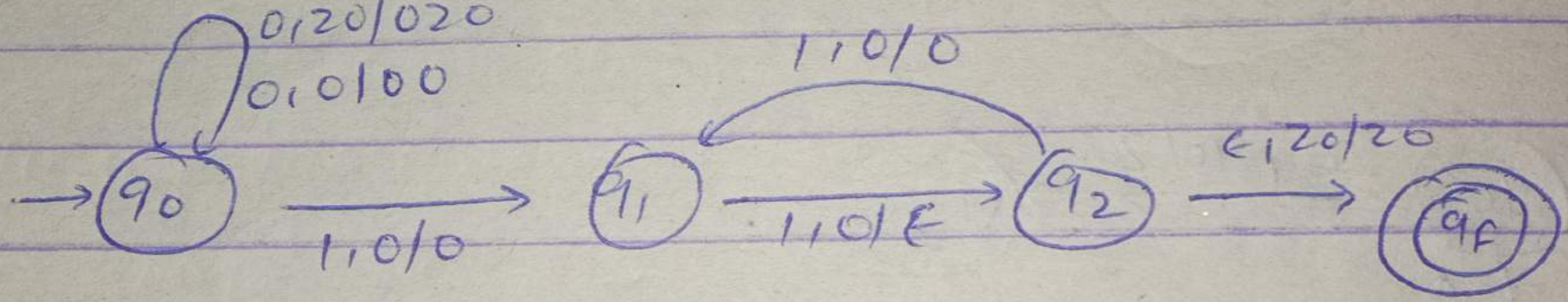


Rule $A \rightarrow BCD$.



Q $BL = \{ 0^n 1^{2n} , n>0 \}$

Soln



Q $L = \{ \omega \in (0+1)^* \mid n_a(\omega) = n_1(\omega) \}$

Soln

→ The language which can be accepted by NPDA that can't be accepted by DPDA

DPDA (Deterministic PDA)

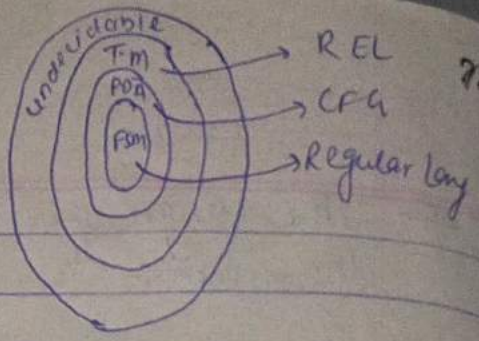
M is deterministic if there is no configuration for which M has a choice of more than one move

Application of PDA

→ Parsing < Top Down / Bottom Up

Language accepted by PDA

DPDA	NPDA
• Every I/P string has a unique path through the machine	• several path for I/P string
• less powerful than NPDA	• More Powerful



Turing Machine

Turing Machine has infinite size tape and it is used to accept Recursive Enumerable language. It doesn't accept ϵ

$$(Q, \Sigma, \Gamma, \delta, q_0, b, F)$$

Q = Non-Empty set of States
Σ = set of symbols
Γ = Not-Empty set of tape symbol
δ = Transition func
q_0 = initial state
b = Blank symbol
F = final state

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times (R/L)$$

At Each step of computation

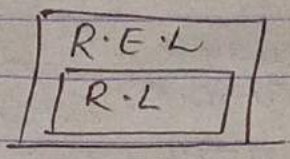
a, b, R
- → Read the current symbol
- → write/update the same cell
- → move exactly one cell either left/Right

Recursive language: If there exist a T.M that accepts every string of the language and rejects every string ~~of the language~~ that is not in language.

It is closed under all except Homomorphism & substitution.

(R.E.L) Recursively Enumerable language: If there exist a TM that accepts every string of the language and doesn't accept the string that aren't in the language.

It is closed under all except subset and complement



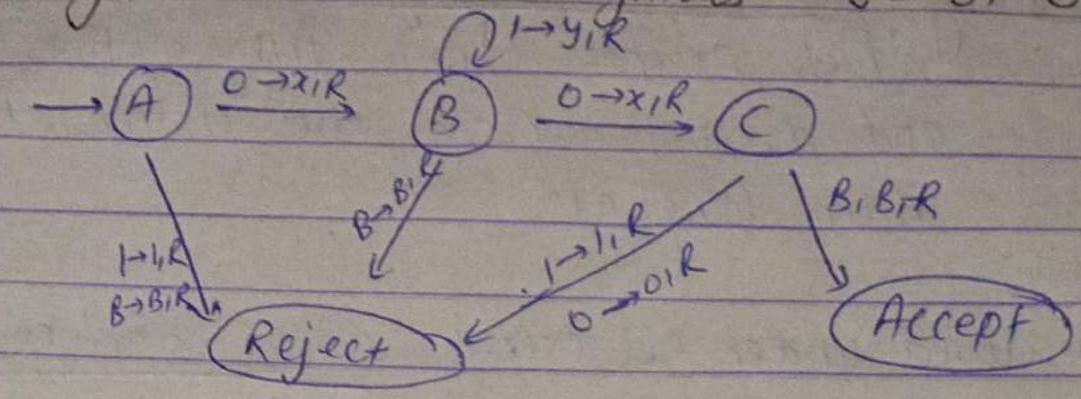
T.M
↳ Recursive language → Turing Decidable → T.M always Halt

R.E.L → Turing Recognizable → Sometimes may Halt

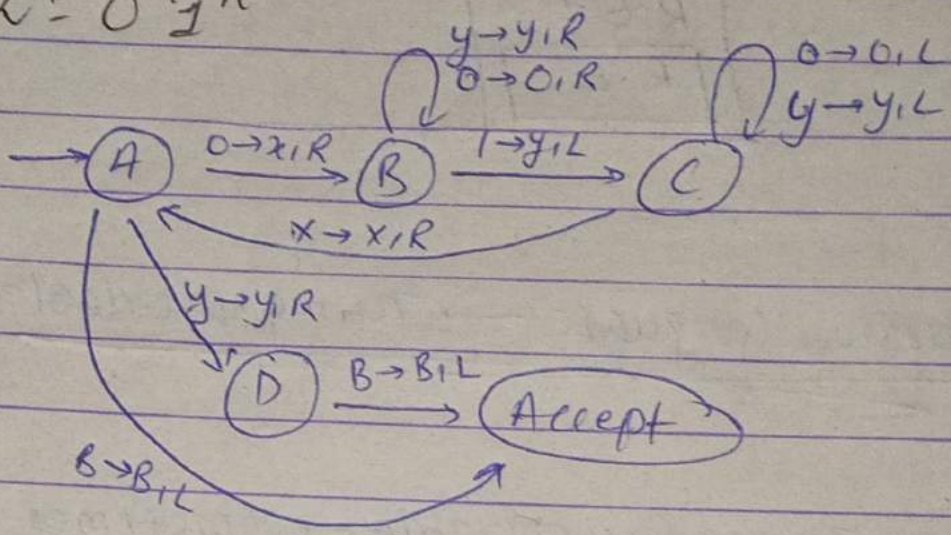
→ Acceptance pblm of a T.M is an undecidable pblm.

⇒ Recursive language ⟵ Decidable language
(REL) Recursively Enumerable language ⟵ Partially- " "
No TM for that language ⟵ Undecidable " "

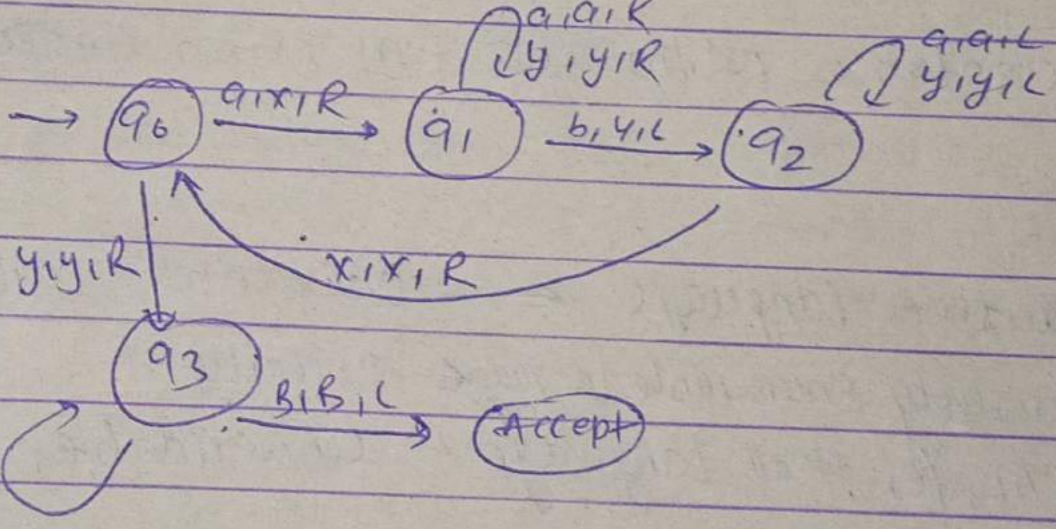
Q Design a TM that recognizes $L = 01^*0$



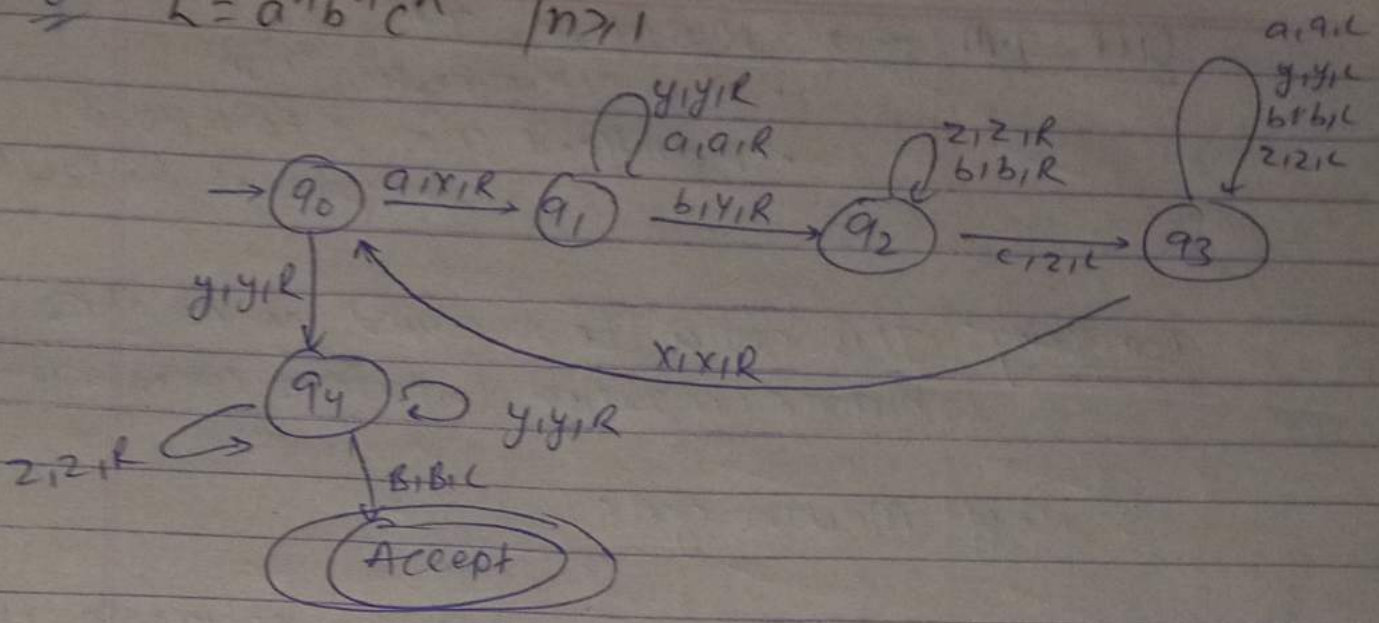
Q $L = 0^n 1^n$



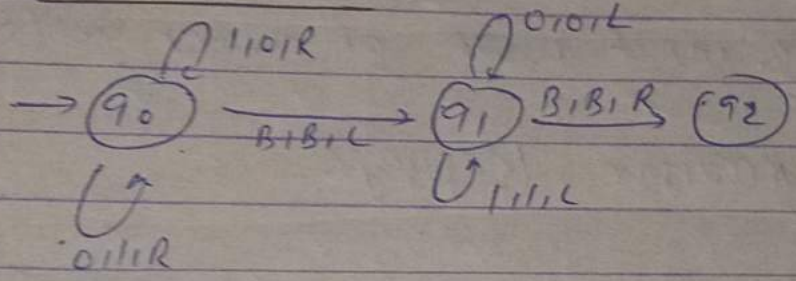
Q $a^n b^n, n \geq 1$



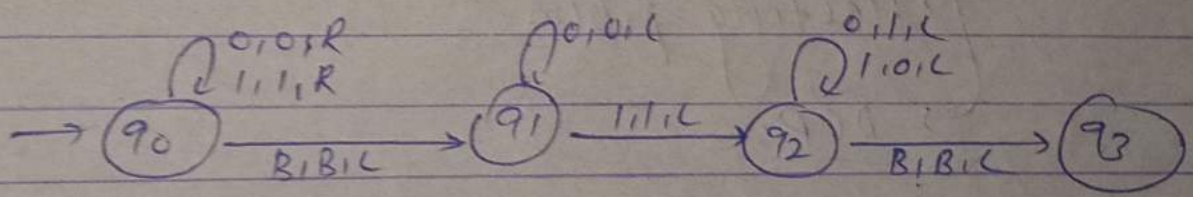
** Q $L = a^n b^n c^n \quad |n \geq 1$



Q Q. 1's complement



Q 2's complement = L



Diff T.M → · Multitape T.M
· Non-Deterministic T.M
· Multihead T.M (Each Head independent for Read/Write)

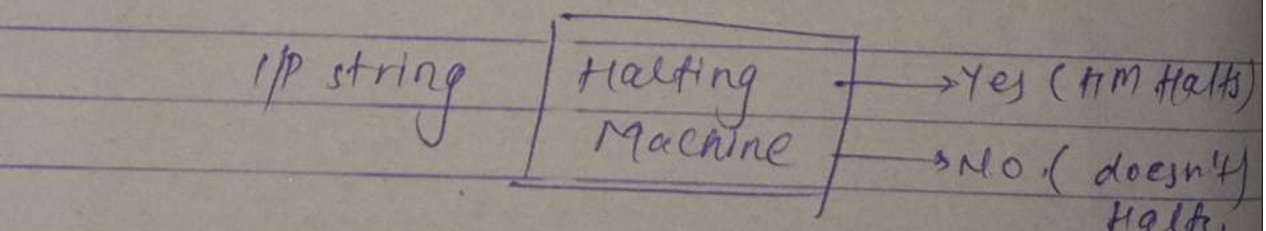
Complexity: This could be measured as no. of moves which are required to perform computation

∴ No of Machine cycle

Decidability: A problm is said to be decidable if there exist a corresponding T.M which halts on every input with an answer yes/No.

It is a Recursive language

Undecidability: May sometimes be partially decidable.
↳ Halting problm
↳ PCP.



P Class: set of decision problm is solvable in poly^{n} time in the class P

eg Minimum spanning tree.

NP Class: It is a "non-deterministic poly" time solvable in the class NP.

The prblm belongs to it easy to check a sol" that may have been very tedious to find.

PCP: It is the prblm of deciding whether a set of ~~son~~ dominos has a match or not.

It is ~~Indoce~~ introduced by Emil Post, 1946. It is an undecidable decision prblm over an alphabet Σ

MPCP: It is just like PCP except that we specify both the set of tiles and also a special tile.

C.S.L:- In this grammar more than one terminal / non terminal symbol may appear on the LHS of production Rule.

Eg: $a^n b^n c^n$, $n \geq 1$

Church Thesis: A functional natural no is computable by an algorithm iff it is computable by T.M

1936

L.B.A → It accepts C.S.L

LBA = T.M + Input size tape

$L = a^n b^n c^n : n \geq 1$ is accepted by LBA

	REG	DCFL	CFL	CSL	REC	RE
Membership prblm $w \in \Sigma^*$	✓	✓	✓	✗	✗	✗
Infiniteness prblm	✓	✓	✓	✗	✗	✗
Emptiness prblm, $L=\phi$	✓	✓	✗	✗	✗	✗
Equality prblm $L_1=L_2$	✓	✓	✗	✗	✗	✗
Ambiguous	✓	✓	✗	✗	✗	✗
completeness $L=\Sigma^*$	✓	✓	✗	✗	✗	✗
$L_1 \cap L_2 = \phi$	✓	✗	✗	✗	✗	✗
$L_1 \subseteq L_2$ subset	✓	✗	✗	✗	✗	✗

REG = Recursively Enumerable Grammar

DCFL = Deterministic Context free ~~Grammar~~ lang.

CFL = Context free language

CSL = Context Sensitive language

RE