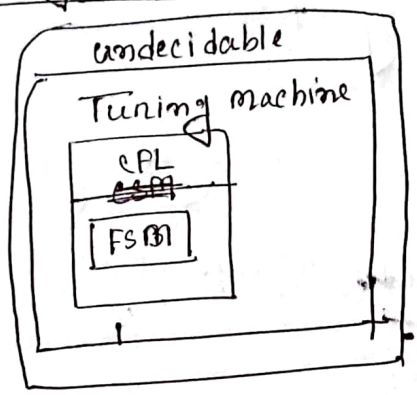
Theory of Computation.



Finite state Machine

Pre requisites.

(i) Symbol → 0, 1, 2, a, b, c . . . .

(ii) Alphabet — Σ → collection of symbols {a, b}, {d, e, f, g} {0, 1}

(iii) String → sequence of symbols a, b, 0, 1
aa, bb, 01 .

(iv) Language → set of strings

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

L_1 = set of all strings of length 2.

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

L_2 = set of all strings length 3.

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

L_3 = set of all strings that begin with 0.

$= \{0, 00, 01, 000, 001, 010, 011, 0000, \dots\}$

↳ infinite.

Powers of sigma (Σ):

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

Σ^0 = set of all strings of length 0 → set of all string length 0.

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

Σ^1 = " " " " " " 1: $\Sigma^1 = \{0,1\}$

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

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

⋮

Σ^n = " " " " " " n: Σ^n =

Cardinality: Number of element in a set.

$\Sigma^0 = 1$

$\Sigma^2 = 2^4$

$\Sigma^3 = 8$

$\Sigma^n = 2^n$

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cdots$

$= \{\epsilon\} \cup \{0,1\} \cup \{00, 01, 10, 11\} \cup \{\cdots\}$

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

↳ infinite set.

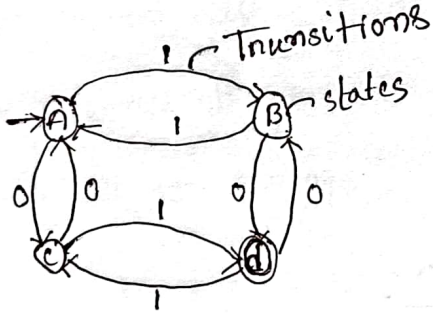
Finite state Machine

Finite Automata.

- FA without output
  - Moore Machine
  - Mealy machine
- FA without output.
  - DFA
  - NFA
  - ϵ-NFA

DFA → Deterministic Finite Automata

- it is the simplest model of computation.
- it has very limited memory.



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

Q = set of all states

Σ = inputs

q_0 = start state / initial state.

F = set of final states.

δ = transition function that maps $Q \times \Sigma \rightarrow Q$

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

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

$q_0 = A$

$F = \{D\}$

$\delta =$

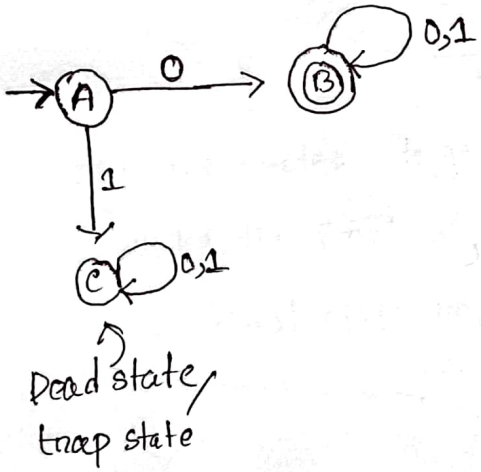
δ

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

DFA Example – 1.

$L1$ = set of all strings that start with '0'.

$= \{0, 00, 01, 000, 010, 011, 0000, - - -\}$



eg. 001 ✓

initial state: A → B → B → B

Final state

eg. 101 ✗

A → C → C → C

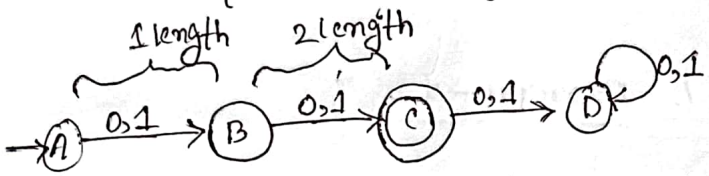
Not final state

Example – 2

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

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

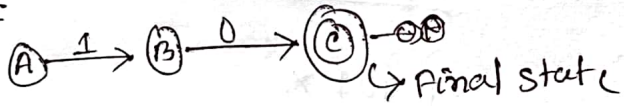
$L = \{00, 01, 10, 11\}$



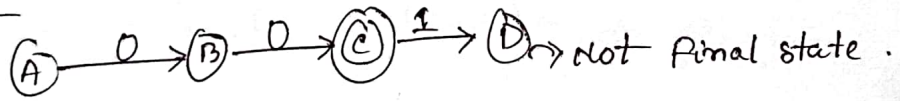
↳ Any string that has length > 2 goes to state D and never comes back (Dead state / trap state).

eg: 00 ✓

eg: 10 ✓



eg: 001 ✗



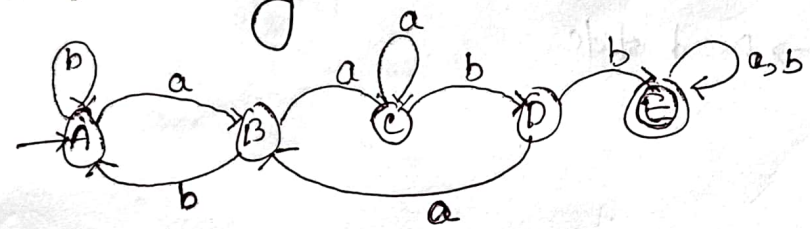
Example—3

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

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

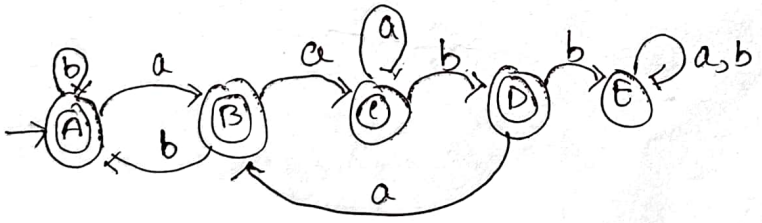
Try to design a simpler problem.

Let us construct a DFA that accepts all strings over {a, b} that contains the string aabb in it.



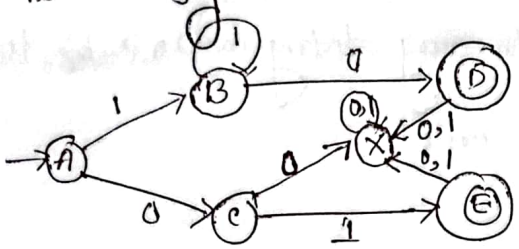
→ Flip the states to get the ~~final~~ main output.

→ Make the final state into non-final state and make the non-final state into final states.



Example-4

* How to figure out what DFA recognizes?



10 ✓
110 ✓
1111110 ✓

L = {accepts the string '01' on a string of at least '1' followed by a '0'.}

eg: 001, 010, 011, 1101, 1100
× × × × ×

(X) → Dead state.

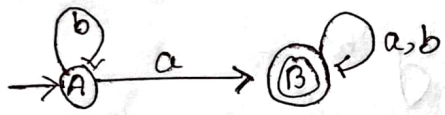
Question solution.

Drawing state diagram of D. finite automata.

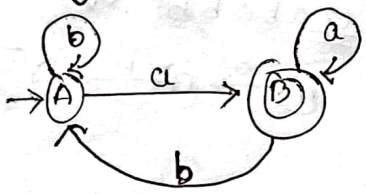
⇒ Some languages over alphabet $\Sigma = \{a, b\}$

1. All string that contains 'a'.

$= \{a, aa, ab, ba, bab, ---\}$

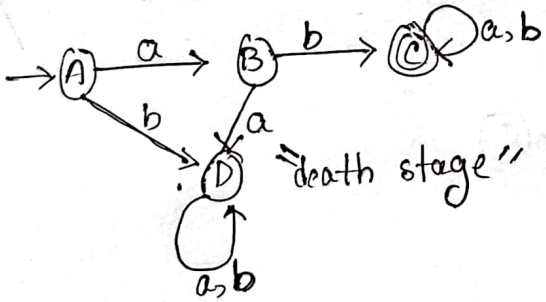


* if ending with 'a'.

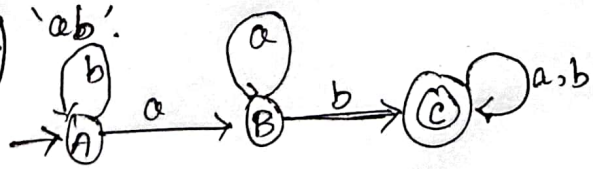


2. l_R = All strings starting with 'ab' over $\{a, b\}$

$= \{ab, aba, abb, ---\}$

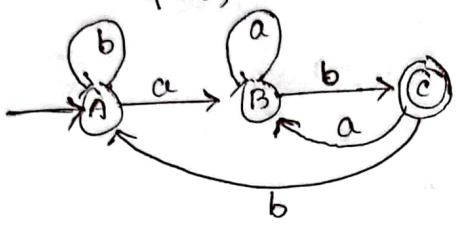


* containing 'ab'.



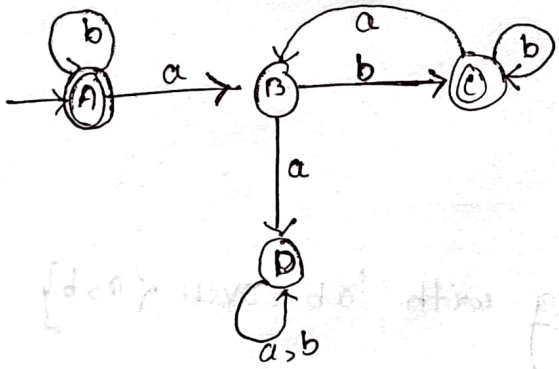
⑤ All string ending with 'ab'

{ab, aab, bab, baab, abab, abbab - - -}

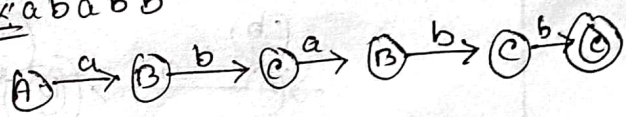


⑥ Every 'a' should be followed by 'b'.

= {ab, bab, abbb, ababb - - -, b, ε, - - -}

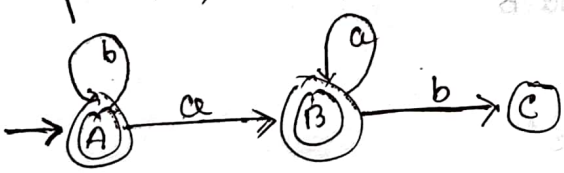


Ex: ababb

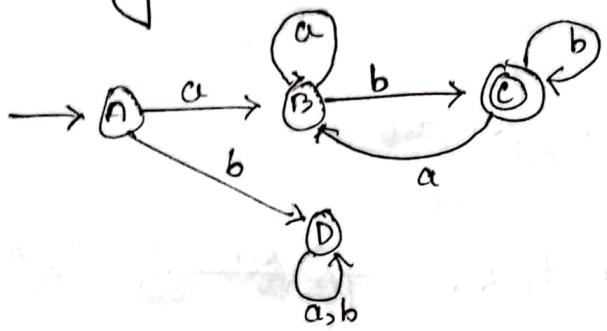


⑦ 'a' must not be followed by 'b'.

= {a a, b, bb, baa, bba, bbbaa, - - -, ε, ε - - }

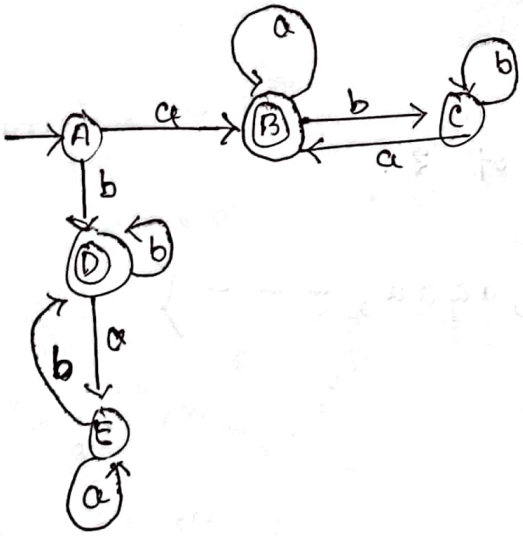


④ starting with 'a' ending with 'b'.



⑤ starting Ending with same character.

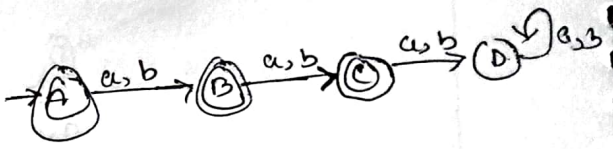
{a, aaa, bab, b, bb, abbba, babb, ---}



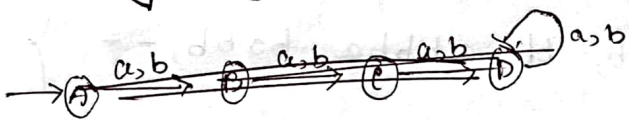
10. string length is exactly 2.

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

11. string length is at most 2.

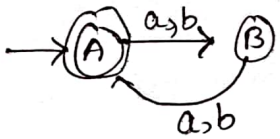


12. string length is at least 2.

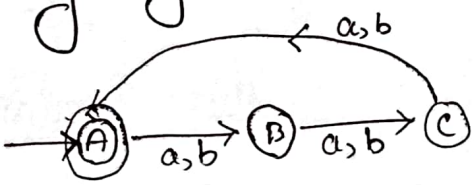


13. string length is multiple of 2.

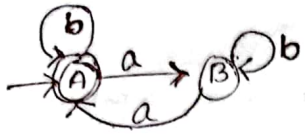
$\{\varepsilon, aa, ab, ba, bb, aaaa, ---\}$



14. string length is multiple of 3.



* $L = w \in \{a, b\}^*$ in which number of 'a' is even.

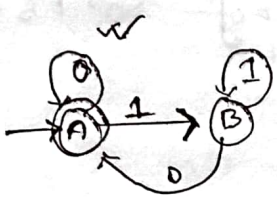


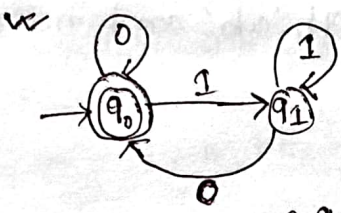
	a	b
q_0	q_1	q_0
q_1	q_0	q_1

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

$L = w \in \{0, 1\}$, is a binary number divisible by 2.

$= \{\varepsilon, 0, 10, 100, 1000, ---\}$ mod 2 → 0 accept q_0, 1 reject q_1





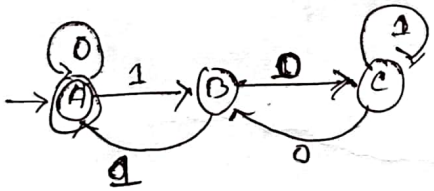
	0	1
q_0	q_0	q_1
q_1	q_0	q_1

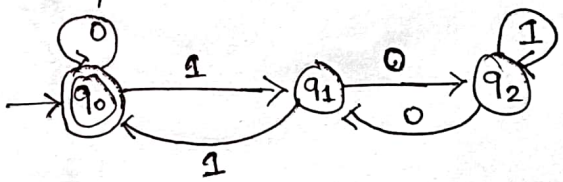
11 → 3 % 2 = 1

* divisible by 3. mod → 3 → 0 q_0, 1 q_1, 2 q_2

	0	1
→ q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

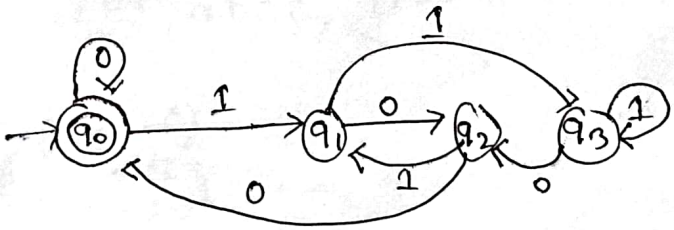
{011, 110, 1001, ---}





* divisible by 4.

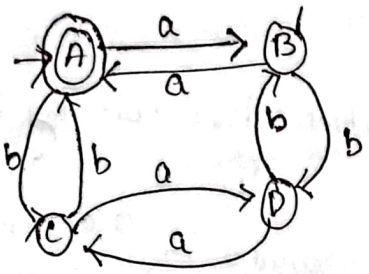
	0	1
q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_0	q_1
q_3	q_2	q_3



16. Number of 'a' and number of 'b' is even.

{~~aa, bb, aabb, bbaa~~ - - ->

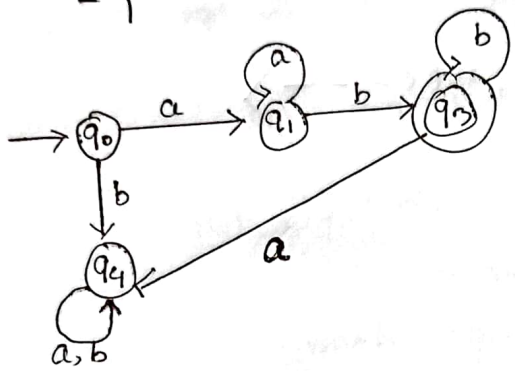
{ ε, aa, bb, abab, aabb, bbaa, baba, abba, baab - - ->



17. string starts with 'ab' and number of 'a' is even.

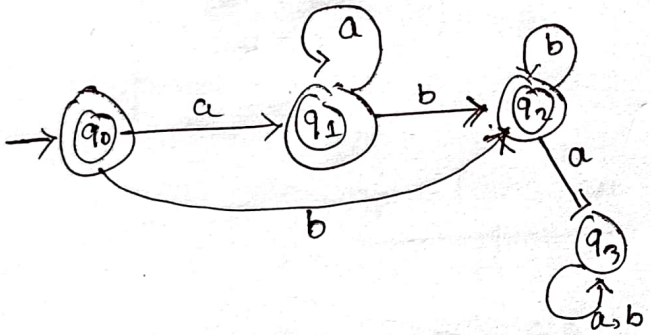
18. $L = \{a^m b^n$ where $m, n >= 1\}$

$= \{ab, aab, abb, ---\}$ sequence break not permitted.



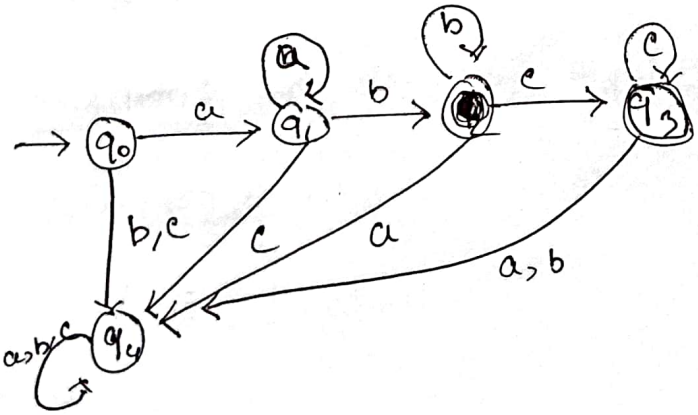
19. $L = \{a^m b^n$ where $m, n >= 0\}$

$= \{a, aa, aaa, ---, b, bb, --- ab, aabb ---\}$



20. $L = \{a^m b^n c^u \mid m, n, u >= 1\}$

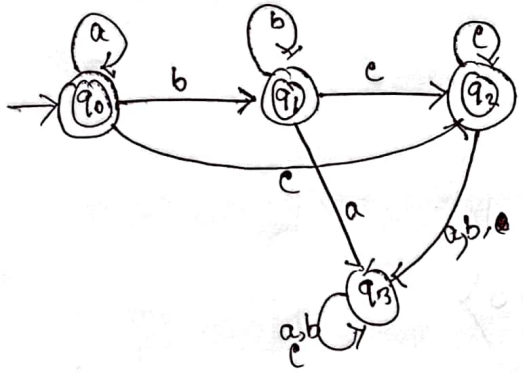
$= \{abc, aabc, abbc, abcc, abcb, ---\}$



21. $L = \{a^m b^n c^k \mid m, n, k >= 0\}$

$= \{\epsilon, a, aa, aaa, ---, b, bb, bbb, -- c, cc, ccc ---,$

$abc, aabc, abbc, abbcc - - -\}$



Regular languages.

* A language is said to be regular languages if and only if some Finite state machine recognizes it.

* Not regular.
  - which are not recognized by any FSM.
  - which requires memory.

eg. abab b ab ab b
Not regular
can't be designed with FSM

$a^N b^N$
a a a b b b → Not recognized by FSM because no memory
Not regular

Operations on regular language.

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

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

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

Theorem 1: The class of Regular languages is closed under UNION

Theorem 2: " " " " " " under

concatenation.

NFA — Non-deterministic Finite Automata.

DPA

Determinism

(i) In DFA given the current state we know the next state will be.

(ii) It has only one unique next state.

(iii) It has no choices or randomness.

(iv) It is simple and easy design.

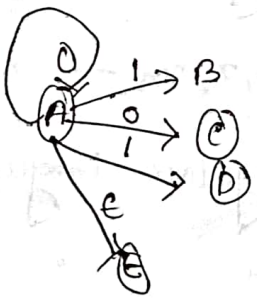
NFA

Non-determinism.

(i) In NFA, given a current state there could be multiple next state.

(ii) The next state may be choosen at random.

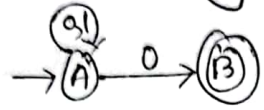
(iii) All the next states may be choosen in parallel.



ϵ → empty string/input.

NFA — Formal Defn

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



~~A×0~~ $\delta = Q \times \Sigma \to 2^Q$

$A \times 0 \to A$

$A \times 0 \to B$

$A \times 1 \to A$

$B \times 0 \to \phi$

$B \times 1 \to \phi$

$A' \to A, B, AB, \phi$

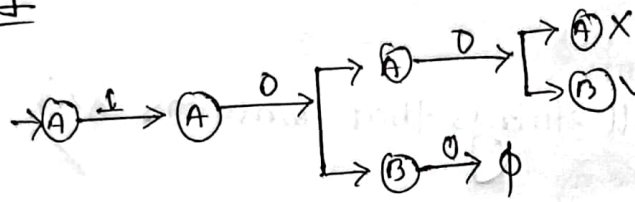
3 states, ABC

$A' \to A, B, C, AB, AC, BC, ABC, \phi$

2^n → possibilities

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

eg: 100



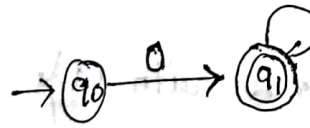
if one string on the accepting state then its accepted.

* If there is any machine that ends in any set of states out of which at least one state is a final state then NFA accepts.

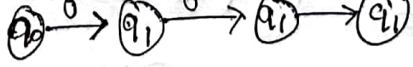
Example – 2

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

= {0, 00, 01, 000, ----}



eg: 001



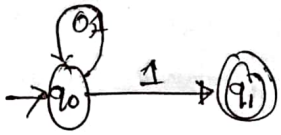
eg: 101

dead configuration

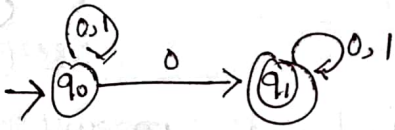
Ex-2 L = {set of all strings with length 2}

= {00, 01, 10, 11}

Ex-3: L = {set of all strings that ends with 1}

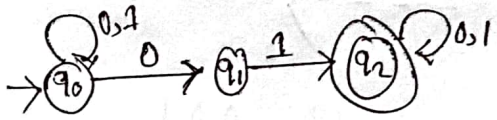


Ex-4: L = {set of all strings that contains '0'}

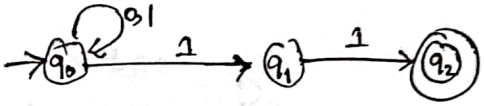


Ex-4: L = {set of all strings that starts with '10'}

Ex-5: L = {set of all string that contains '01'}

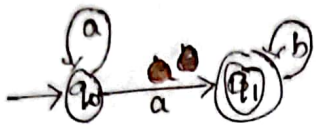


Ex-6: L = {set of all string that ends with 11}

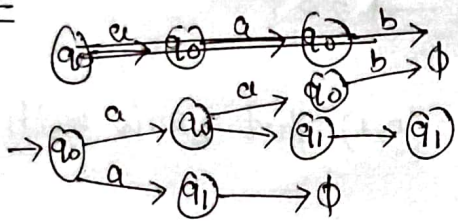


Problems

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

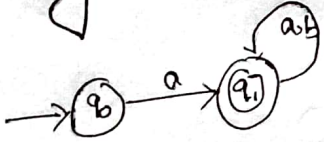


eg: a a b

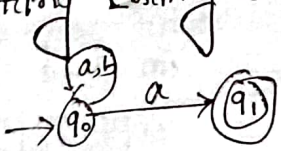


2. $\Sigma = \{a, b\}$

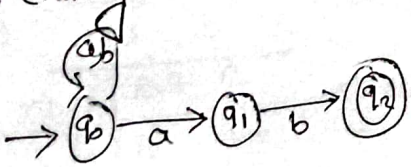
string starts with 'a'.



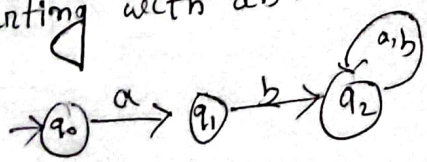
3. string Ending with 'a'.



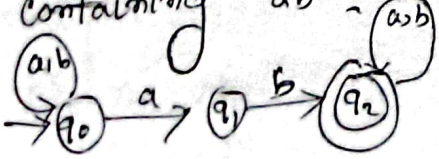
4. Ending with 'ab'



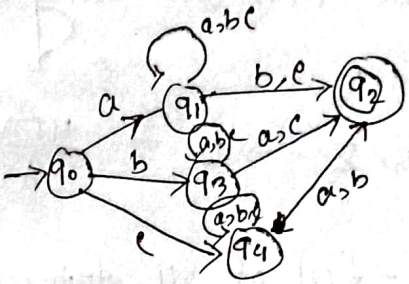
5. starting with 'ab'



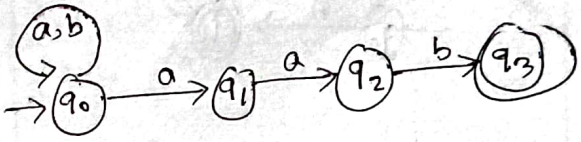
6. containing 'ab'.



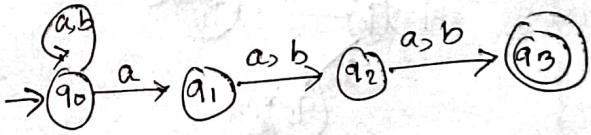
7. starting and Ending with different state.



8. string ending with aab



9. 3rd character is 'a' from right hand side.



Convertion of NFA to DFA

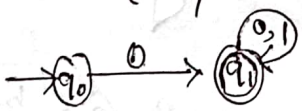
(*) Every DFA is an NFA but not vice versa, But there is an Equivalent DFA for every NFA.

DFA $\delta = Q \times \Sigma \to Q$ $\cong$ NFA $\delta = Q \times \Sigma \to 2^Q$.

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

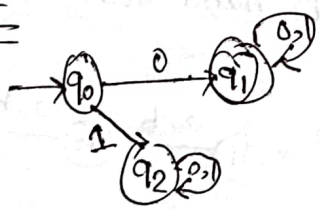
$\Sigma = \{0, 1\}$

NFA



	0	1
q_0	q_1	ϕ
q_1	q_1	q_1

DFA



	0	1
q_0	q_1	q_2
q_1	q_1	q_1
q_2	q_2	q_2

q_2 — dead state

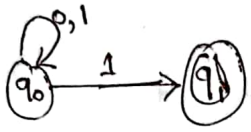
in dead state whatever input you get it stays at that state.

Examples

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

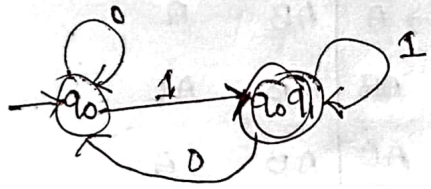
$\Sigma = \{0,1\}$

NFA



	0	1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	ϕ	ϕ

DFA



	0	1
q_0	$\{q_0\}$	$\{q_0q_1\}$
q_0q_1	$\{q_0\}$	$\{q_0q_1\}$

q_0q_1 = single state

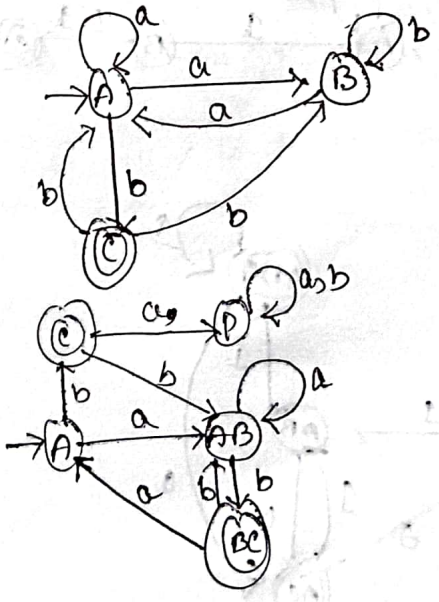
(2) Find the equivalent DFA for NFA given by $M = [\{A,B,C\}, (a,b), \delta, A, \{C\}]$ where δ is given by:

NFA

	a	b
→ A	A,B	C
B	A	B
(C)	–	A,B

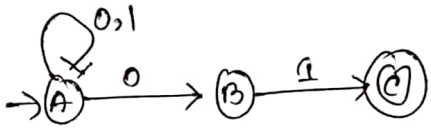
DPA

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



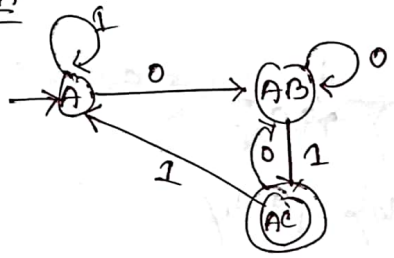
(*) L={set of all strings over (0,1) that ends with '01'}, convert its equivalent DFA.

NFA



	0	1
→ A	A,B	A
B	ϕ	C
(C)	ϕ	ϕ

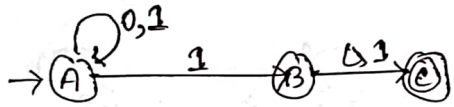
DFA



	0	1
→ A	AB	A
AB	AB	AC
(AC)	AB	A

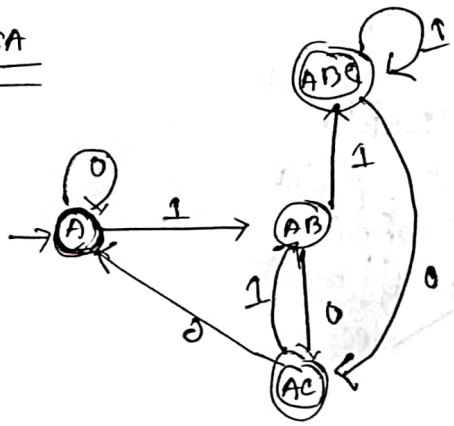
(*) Design an NFA for a language that accepts all string over {0,1} in which the second last symbol is always 1. Convert it to eq. DFA

NFA:



	0	1
→ A	A	{A,B}
B	C	C
*C	ϕ	ϕ

DFA



	0	1
→ A	A	AB
AB	AC	ABC
*AC	A	AB
*ABC	AC	ABC

Minimization of DFA (Partitioning method)

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

(1) combine two states when they are equivalent.

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

$\delta(A,x) \longrightarrow F$ and $\delta(B,x) \longrightarrow P$ OR $\delta(A,x) \longrightarrow F$ and $\delta(B,x) \longrightarrow P$

where 'x' is any input string.

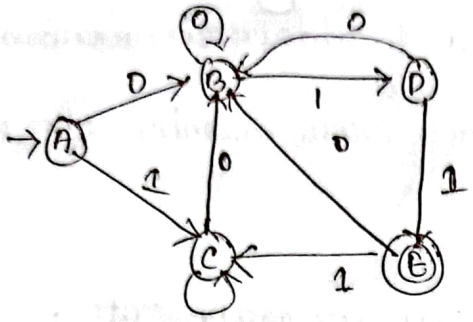
If $|x| = 0$ (length), then A and B are said to be 0 equivalent.

If $|x| = 1$, " A " B " " " " 1 "

If $|x| = 2$, " A " B " " " " 2 "

If $|x| = n$, " A " B " " " " n "

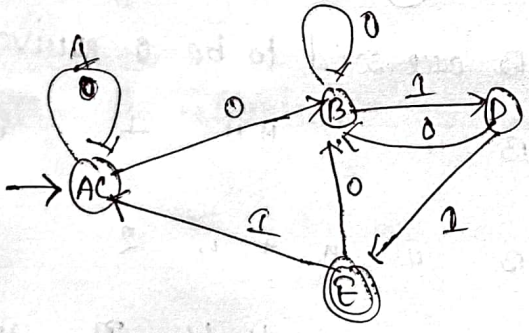
Examples - 1.



(A) State transition table.

	0	1
→A	B	C
B	B	D
C	B	C
D	B	E
(E)	B	C

	Non Final	Final
0 Equivalence,	{A, B, C, D}	{E}
1 Equivalence	~~{A, B}~~ {A, B, C}	{D} {E}
2 Equivalence	{A, C}	{B} {D} {E}
3 Equivalence	{A, C}	{B} {D} {E}



Example—2

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

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

1 – " – $\{q_0, q_4, q_6\}$

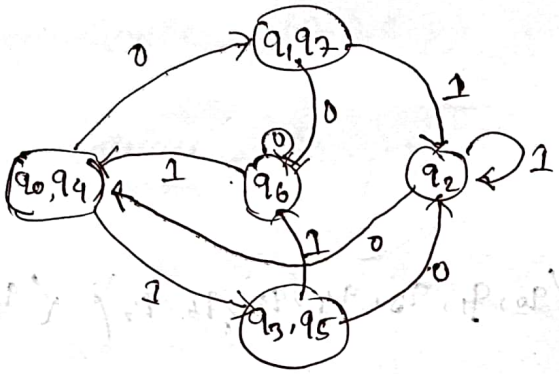
$\{q_1, q_7\}$

$\{q_3, q_5\}$ $\{q_2\}$

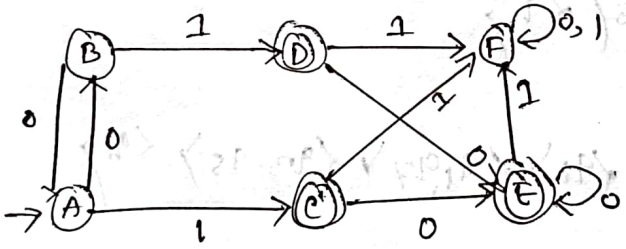
2 – Eq. – $\{q_0, q_4\}$ $\{q_6\}$ $\{q_1, q_7\}$ $\{q_3, q_5\}$ $\{q_2\}$

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

	0	1
→ $\langle q_0, q_4 \rangle$	$\langle q_1, q_7 \rangle$	$\langle q_3, q_5 \rangle$
$\langle q_6 \rangle$	$\langle q_6 \rangle$	$\langle q_0, q_4 \rangle$
$\langle q_1, q_7 \rangle$	$\langle q_6 \rangle$	$\langle q_2 \rangle$
$\langle q_3, q_5 \rangle$	$\langle q_2 \rangle$	$\langle q_6 \rangle$
* $\langle q_2 \rangle$	$\langle q_0, q_4 \rangle$	$\langle q_2 \rangle$



Minimization of DPA with more final states



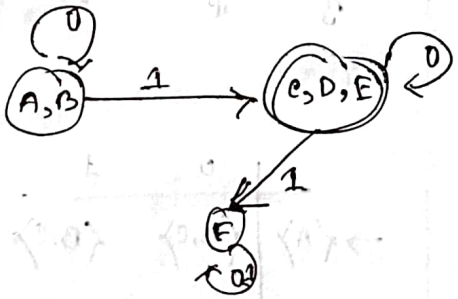
	0	1
→ A	B	C
B	A	D
(C)	E	F
(D)	E	F
(E)	E	F
F	F	F

0-Equivalence — {A, B, F}, {C, D, E}

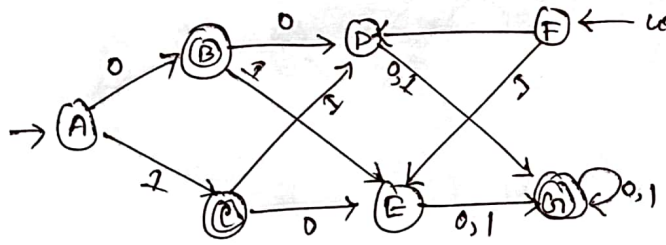
1 » — {A, B} {F} {C, D, E}

2 » — {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}

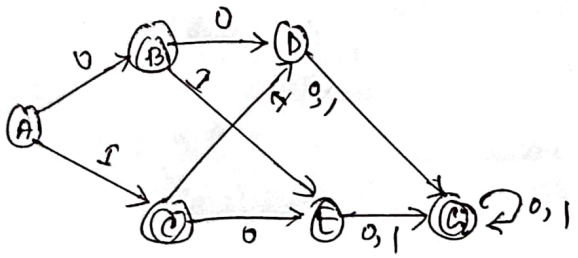


(A) when there are unreachable states involved



← unreachable state : A state is called unreachable if there is no way it can be reached from initial state.

remove the unreachable state —



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

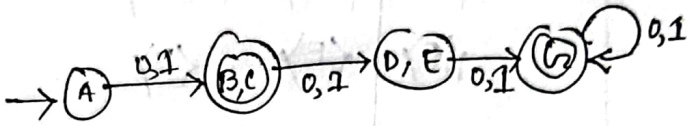
0 – Equivalence: {A, D, E} {B, C, G}

1 – " : {A, D, E} {B, C} {G}

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

3 " : {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}

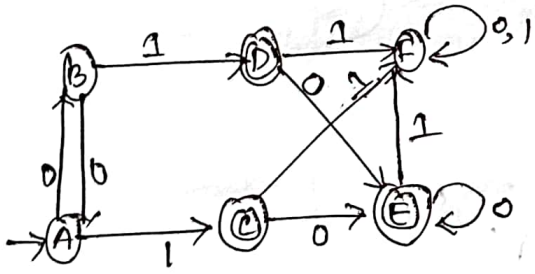


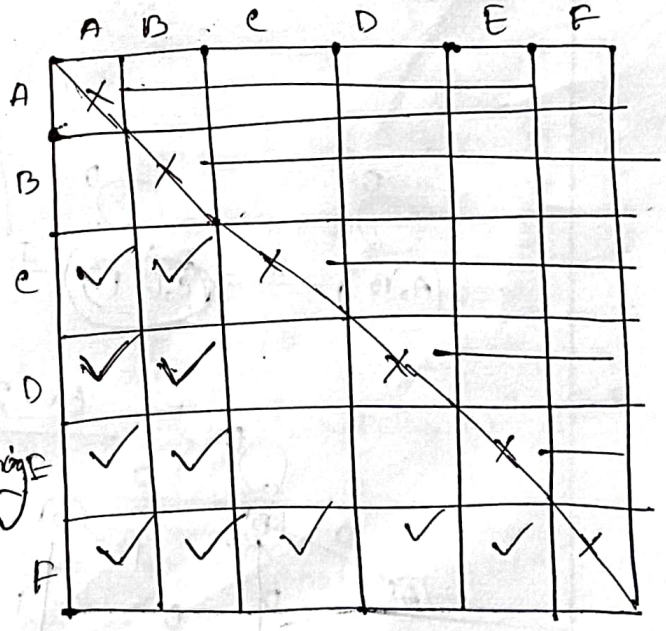
Minimization of DFA — Table filling Method

"Myhill–Nerode theorem"

Steps:

1. Draw a table for all pairs of states (P, Q).
2. Mark all pairs where $P \in F$ and $Q \notin F$.
3. If there are any unmarked pairs (P, Q) such that $[\delta(P,x), \delta(Q,x)]$ is marked then mark (P, Q), (where, x is input symbol). Do this until No more marking can be made.
4. Combine all the unmarked pairs and make them a single state in the minimized DFA.





$(BA) \rightarrow \delta(B,0) = A$, $\delta(A,0) = B$ } not marked in table, do nothing

$\delta(B,1) = D$, $\delta(A,1) = C$ } not marked

$(DC) \rightarrow \delta(D,0) = E$, $\delta(C,0) = E$ } $\delta(D,1) = F$, $\delta(C,1) = F$ } ignore because not present in the table.

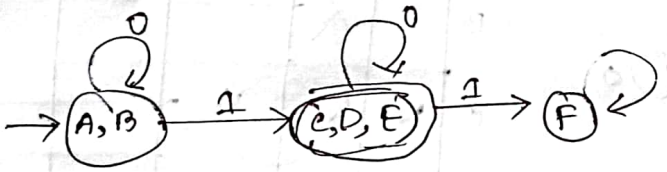
$(E,C) \rightarrow \quad \delta(E,0) = E$, $\delta(C,0) = E$; $\delta(E,1) = F$, $\delta(C,1) = F$ } ignore

$(E,D) = \quad \delta(E,0) = E$, $\delta(D,0) = E$; $\delta(E,1) = F$, $\delta(D,1) = F$ } ignore

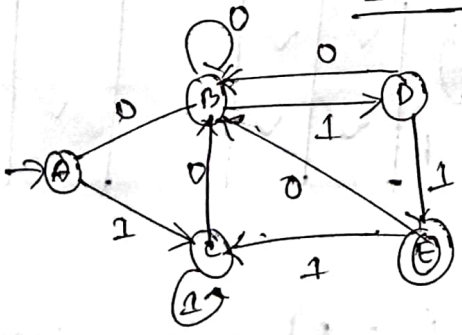
$(E,A) = \quad \delta(E,0) = E$, $\delta(A,0) = B$; $\delta(E,1) = F$, $\delta(A,1) = C$

$(E,B) = \quad \delta(E,0) = E$, $\delta(B,0) = A$ } marked ; $\delta(E,1) =$, $\delta(B,1) =$

(A,B) (D,C) (E,C) (E,D) → combined to single state (unmarked pair)



Ex-2



practice

Finite Automata with outputs

Mealy machine

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

Q = Finite set of states.

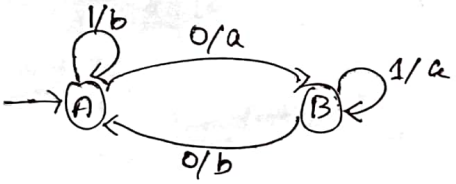
Σ = Finite non-empty set of input alphabet.

Δ = The set of output Alphabet

δ = Transition function $Q \times \Sigma \to Q$

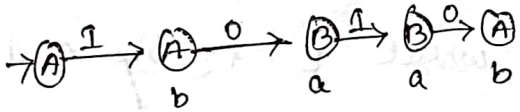
λ = output function $\Sigma \times Q \to \Delta$ (X) (dif).

q_0 = Initial states / start state.



output depends on current state and input.

Eg: 1 0 1 0



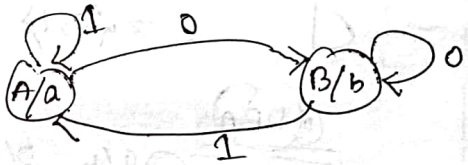
input = output (length)

$n = n$

Moore machine

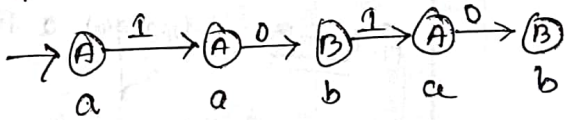
$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

η = output function $Q \to \Delta$



output depends upon on only the current state.

Eg: 1 0 1 0



output ≠ input

$n+1 \leftarrow n$.

Before getting input we have output on initial state.

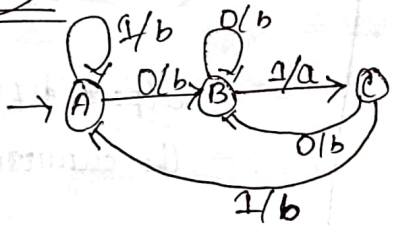
Construction of Mealy machine.

Ex-1: Construct a Mealy machine that produces the 1's complement of any binary string.

Ex-2: construct a Mealy machine that prints 'a' whenever the sequence (01) is encountered in any input binary string.

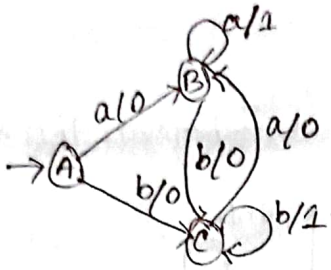
$\Sigma = \{0, 1\}$ $\Delta = \{a, b\}$

DFA



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

aa = 1
bb = 1



Eg: abb
baa

(*) Ex-9: construct a Mealy machine that gives 2's complement of any binary input. (Assume that the last carry bit is neglected)

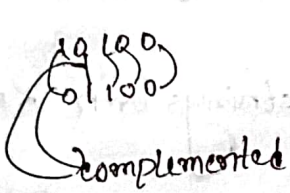
Eg: msb ← lsb
10100

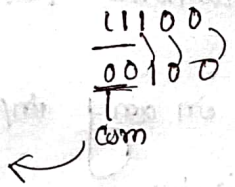
1's 01011
+1
2's 01100

11100
1's 00011
+1
2's 00100

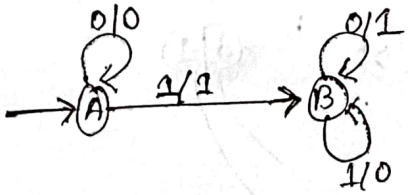
1111
0000
+1
0001

1111
0001





That means the first 1 from starting at LSB will remain the same but after that all bit will be complemented.



~~101~~

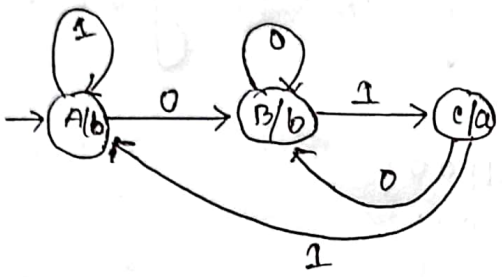
1 0 1 0 0
B B B A A
0 1 1 0 0

Construction of moore machine

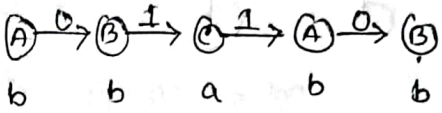
(1) Construct a Moore Machine that prints 'a' whenever the sequence 01 is encountered in any input binary string.

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

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



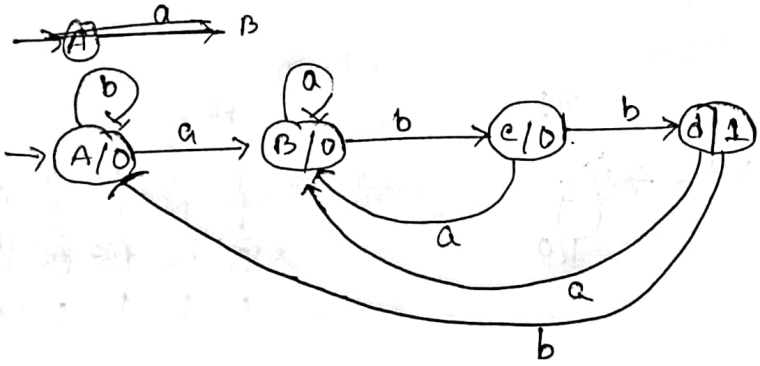
0110



(2) Construct a Moore machine that counts the occurrences of the sequence abb in any input strings over $\{a, b\}$

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

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

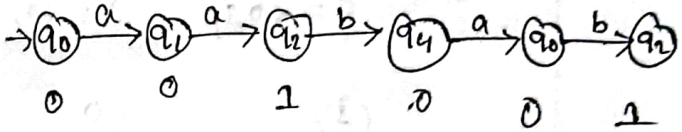


(3) For the following moore machine the input alphabet is $\Sigma = \{a, b\}$ and the output alphabet is $\Delta = \{0, 1\}$. Run the following input sequence and find the respective outputs.

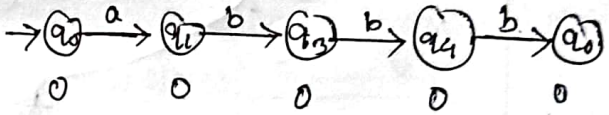
(i) aabbab (ii) abbb (iii) ababb.

states	a	b	outputs
$\to q_0$	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

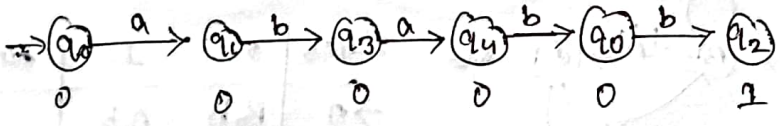
(i) aabab



(ii) abbb



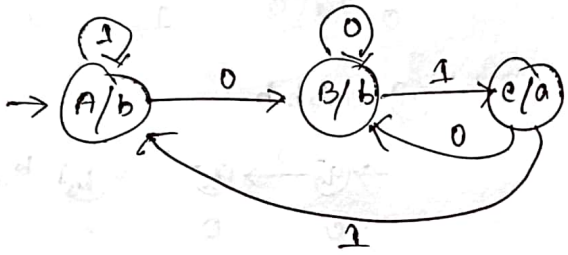
(iii) ababb



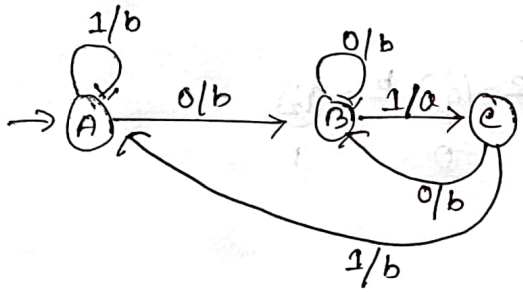
Convertion of Moore Machine to mealy machine

Ex-1: construct a Moore machine that prints a whenever the sequence '01' is encountered in any input binary string and then convert it to its equivalent mealy machine.

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



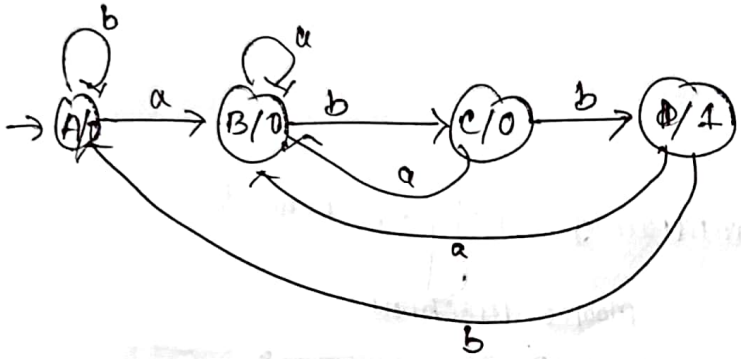
state	0	1	output
→ A	B	A	b
B	B	C	b
C	B	A	a

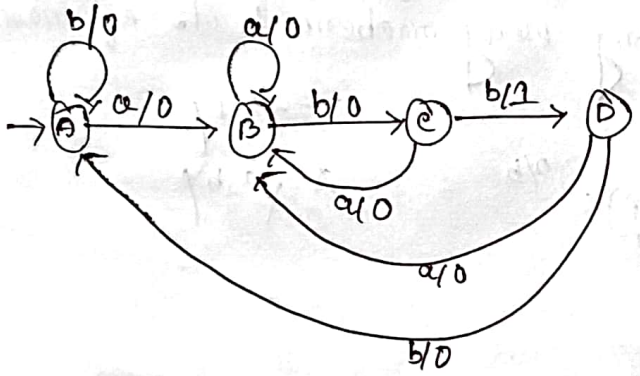


states	0	1	outputs
→ A	B, b	A, b	
B	B, b	C, a	
C	B, b	A, b	

(2) The given moore machine counts the occurrences of sequence 'abb' in any input string over {a, b}. convert it to Eq. mealy machine.

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





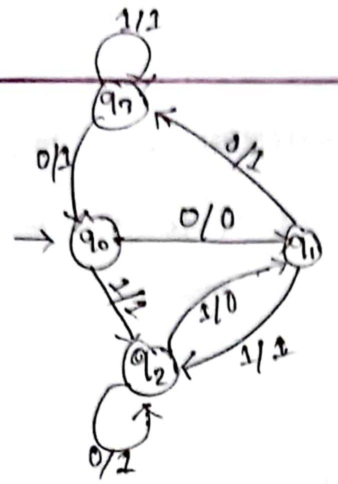
states	a	b
→A	B, 0	A, 0
B	B, 0	C, 0
C	B, 0	D, 1
D	B, 0	A, 0

(3) Convert the moore machine to its equivalent mealy machine.

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

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

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, 1	q_3, 1

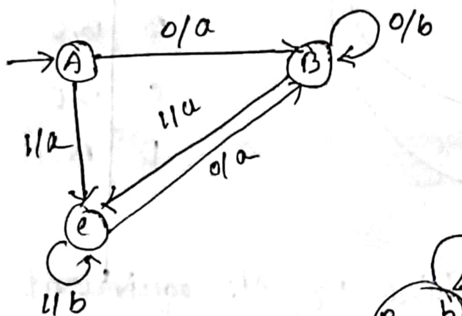


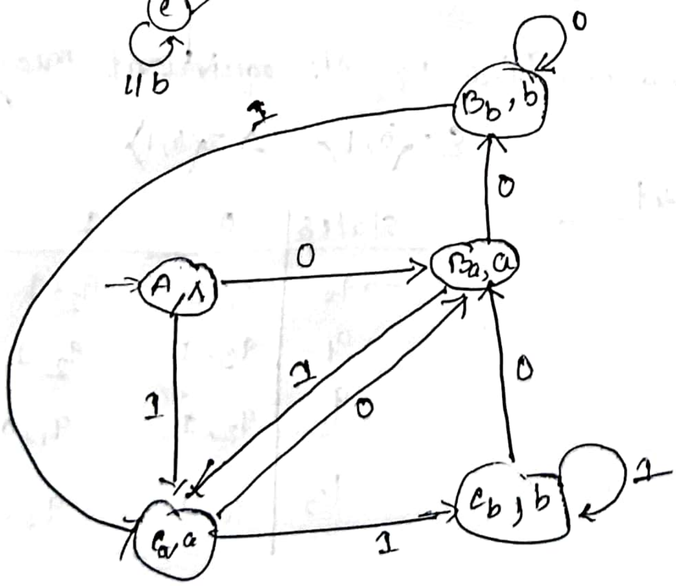
Convertion of Mealy machine to Moore Machine

(*) Convert the following mealy machine to its eq. moore machine.

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

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

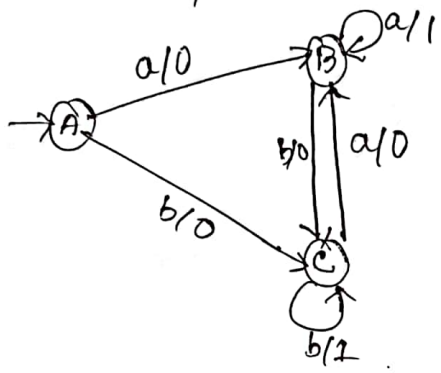




(*) moore → mealy
Number of states were same

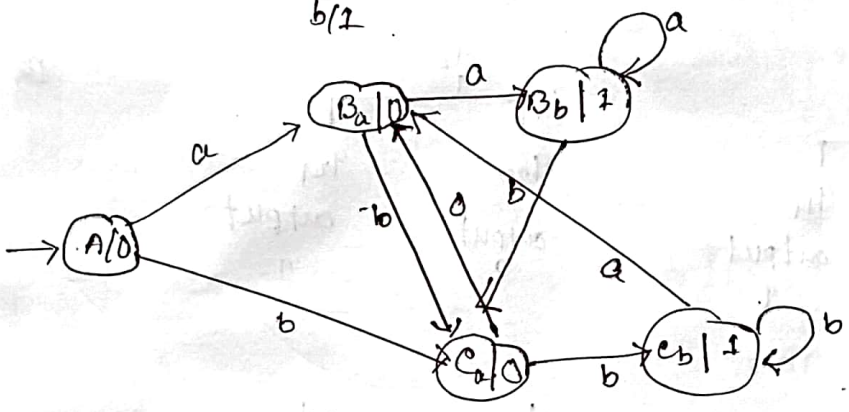
(*) mealy – moore
states increased.
α and Δ y
states output } (α × y) no. of states at max.

Ex-2: Given bellow a mealy machine that prints '1' whenever the sequences 'aa' 'bb' is encountered in any binary string from Σ^* where $\Sigma = \{a, b\}$. Design the eq. Moore machine for it.

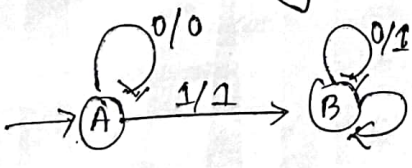


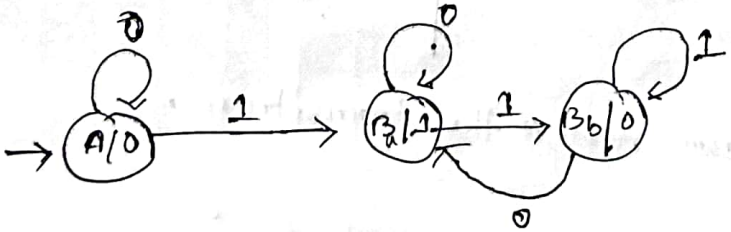
$\Sigma = \{a, b\}$

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



Ex-3 Convert the given mealy machine that give the 2's compliment of any binary input to its eq. moore machine.

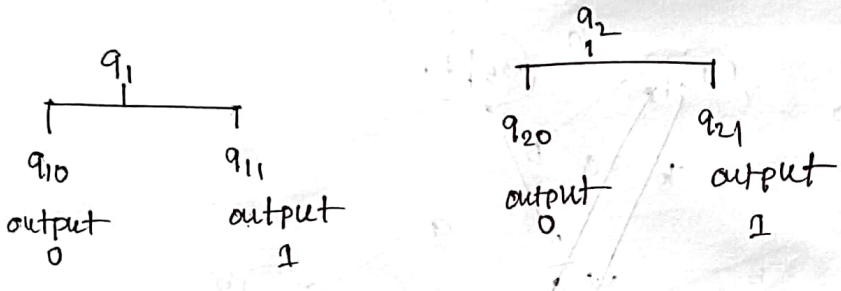




using Transition Table

(M) convert the given mealy machine to its eq. Moore machine.

states	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	$q_1, 0$	$q_0, 1$



states	a	b	output
→ q_0	$q_3, 0$	$q_{11}, 1$	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

(M) now we can remove the transition.

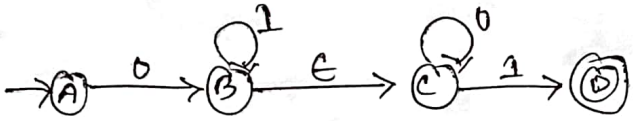
Epsilon (ϵ) – NFA

ϵ – NFA
└ empty symbols.

ϵ-NFA $\{Q, \Sigma, q_0, \delta, F\}$

regular – NFA $\delta = Q \times \Sigma \longrightarrow 2^Q$

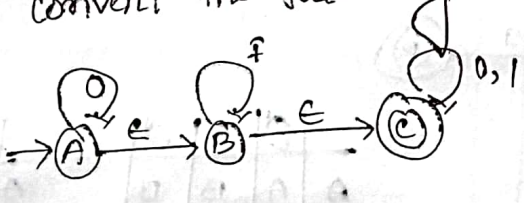
ϵ-NFA $\delta = Q \times (\Sigma \cup \epsilon) \longrightarrow 2^Q$ (δ getting any of the input can go to 2^Q states on getting nothing can go to 2^Q state)



Every state on epsilon (ϵ) goes to itself.

Convertion of ϵ-NFA to NFA

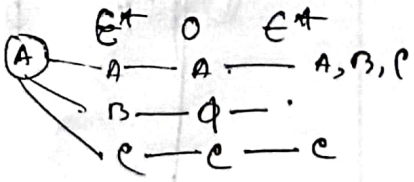
Ex-1: convert the following ϵ-NFA to its eq. NFA.



	0	1
→A	{A,B,C}	{B,C}
B	{C}	{B,C}
C	{C}	{C}

state → ϵ^* → input → ϵ^*

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

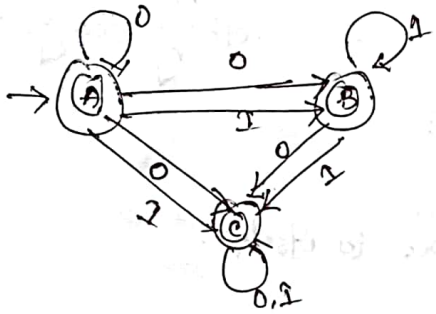


	ϵ^*	1	ϵ^*
A	A	ϕ	
B	B	B	B, C
C	C	C	C

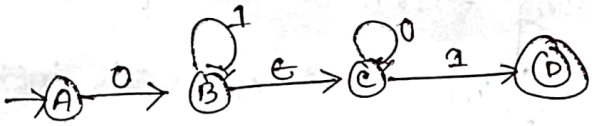
	ϵ^*	0	ϵ^*
B	B	ϕ	
C	C	C	C

	ϵ^*	1	ϵ^*
B	B	B	B, C
C	C	C	C

Final state will be the states which can go to final states only getting the epsilon input.



Ex2: convert the following NFA to its equivalent NFA.

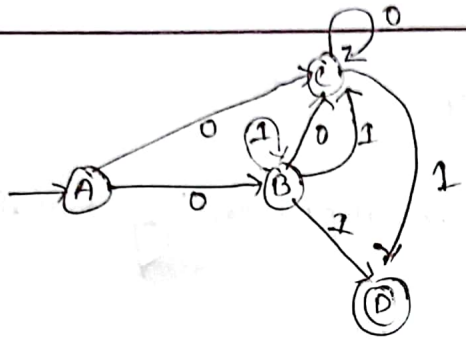


NFA

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

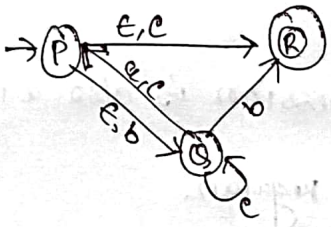
	ϵ^*	0	ϵ^*
A	A	B	B, C
B	B, C	ϕ	ϕ
C	C	C	C
D	D	ϕ	ϕ

	ϵ^*	1	ϵ^*
A	A	ϕ	ϕ
B	B, C	B	B, C
C	C	D	D
D	D	ϕ	ϕ



NFA

Ex:8: convert the following ε-NFA to its eq. NFA.



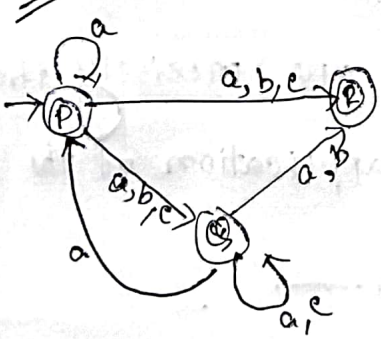
	a	b	c
→P	{P,Q,R}	{Q,R}	{Q,R}
Q	{P,Q,R}	{R}	{Q}
R	ϕ	ϕ	ϕ

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

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

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

NFA



Regular Expression

Regular Expression are used for representing certain set of string in an algebraic fashion.

(i) Any terminal symbol i.e symbols ϵ, Σ including $\wedge$ and ϕ are regular expression $\rightarrow a, b, c$ $\wedge \phi$

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

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

(iv) The iteration (or closure) of regular expression is also ceregular expression. $R \rightarrow (R^*)$ – Regular $a^* = \epsilon, a, aa, aaa, \ldots$

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

Ex1: Describe the following sets as regular expression —

① {0, 1, 2}

$R = 0+1+2$

② $\{\epsilon, ab\}$

$R = \epsilon + ab$

③ {abb, a, b, bba}

$R = abb + a + b + bba$

④ $\{\lambda, 0, 00, 000, \dots\}$ closure of 0

$R = 0^*$

⑤ {1, 11, 111, 1111, ---}

$R = 1^+$

Identities of Regular Expression.

① $\emptyset + R = R$

② $\emptyset R + R\emptyset = \emptyset$

③ $\epsilon R = R\epsilon = R$

④ $\epsilon^* = \epsilon$ and $\emptyset^* = \epsilon$

⑤ $R + R = R$

⑥ $R^*R^* = R^*$

⑦ $RR^* = R^*R$

⑧ $(R^*)^* = R^*$ closure without ϵ.

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

⑩ $(PQ)^*P = P(QP)^*$

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

⑫ $(P+Q)R = PR + QR$ and

$R(P+Q) = RP + RQ$.

Arden's theorem

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

$R = Q + RP$ ——— (1)

$= Q + QP^* P$

$= Q(\epsilon + P^* P)$ $\quad [\epsilon + R^* R = R^*]$

$= QP^*$

proved

unique solution.

$R = Q + RP$

$= Q + [Q + RP]P$

$= Q + QP + RP^2$

$= Q + QP + [Q + RP]P^2$

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

⋮

$= Q + QP + QP^2 + \cdots + QP^m + RP^{m+1}$ $\quad [R = QP^*]$

$= Q + QP + QP^2 + \cdots + QP^m + QP^* P^{m+1}$

$= Q[\epsilon + P + P^2 + \cdots + P^m + P^* P^{m+1}]$

$= QP^*$

An Example proof using Identities of regular expression

prove that $(1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1)$ is equal to $0^*1(0+10^*1)^*$.

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

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

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

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

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

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

Designing of Regular Expression

Ex 1: Design Regular expression for the following languages over $\{a, b\}$.

(1) Language accepting string of length exactly 2.

(2) " " " of length at least 2.

(3) " " " " atmost 2.

Soln: (i) $L = \{aa, ab, ba, bb\}$

$R = aa + ab + ba + bb$

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

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

for 3 → $(a+b)(a+b)(a+b)$.

(ii) $L = \{aa, ab, ba, bb, aaa, \dots\}$

$R = (a+b)(a+b)\boxed{(a+b)^*}$ → anything that will be more than 2.

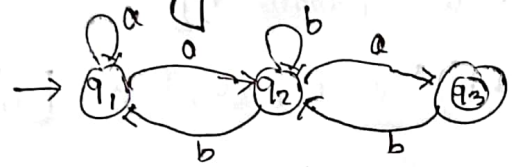
(iii) $L = \{\epsilon, a, b, aa, ab, ba, bb\}$

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

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

NFA to Regular Expression

(i) Find the regular expression for the following NFA —.



$q_3 = q_2 a$ ——→ (i)

$q_2 = q_1 a + q_2 b + q_3 b$ —— (ii)

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

(i) $q_3 = q_2 a$

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

$= q_1 aa + q_2 ba + q_3 ba$ —— (iv)

(ii) $q_2 = q_1 a + q_2 b + q_3 b$ putting value of q_3 from 1.

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

$= q_1 a + q_2 b + q_2 ab$

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

$R \quad Q \quad R \quad P$

$R = Q + RP$
$R = QP^*$ } by ardens theorem.

$\Rightarrow q_2 = q_1 a (b + ab)^*$ —— (v)

(iii) $q_1 = \epsilon + q_1 a + q_2 b$ put q_2 from (v)

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

$q_1 = \epsilon + q_1 \dfrac{(a + a(b+ab)^*) b}{P}$

$R = Q + RP$
$R = QP^*$
$\epsilon R = R$

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

$q_1 = ((a + a(b+ab)^*) b)^*$ —— (vi).

Final state —

$q_3 = q_2 a$

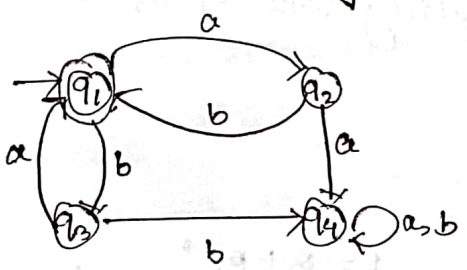
$= q_1 a (b + ab)^* a$ [from V]

$q_3 = (a + a(b+ab)^* b)^* a (b+ab)^* a$ [q_1 from (VI)]

∴ required regular expression for NFA —.

DFA to regular Expression

(*) Find the following expression for the following DFA.



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

$q_2 = q_1 a$ —— (II)

$q_3 = q_1 b$ —— (III)

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

$q_1 = \epsilon + q_2 b + q_3 a$ [put values q_2 and q_3 from (II), (III)]

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

$q_1 = \epsilon + q_1 (ab + ba)$ $\quad R = Q + RP$

$R \quad Q \quad R \quad P$ $\quad R = QP^*$ —

$q_1 = \epsilon (ab + ba)^*$ —

$q_1 = (ab + ba)^*$ —

Required regular expression for DFA —.

Ex: when there are multiple final state.....

$q_1 = \epsilon + q_1 0^*$ —— (i)

$q_2 = q_1 1 + q_2 1$ —— (ii)

$q_3 = q_2 0 + q_3 0 + q_3 1$ —— (iii)

final state $q_1 = \epsilon + q_1 0$

$q_1 = \epsilon 0^*$

$q_1 = 0^*$ —— (iv)

$R = Q + RP$

$R = QP^*$

$\epsilon R = R$.

$q_2 = q_1 1 + q_2 1$

$q_2 = 0^* 1 + q_2 1$

$q_2 = 0^* 1 (1)^*$ —— (v)

R = union of both the final state

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

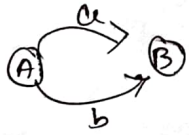
$= 0^* (\epsilon + 1 (1)^*)$ $\quad \epsilon + RR^* = R^*$

$R = 0^* 1^*$

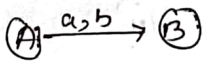
required regular expression for DFA.

Convention of Regular Expression to Finite Automata

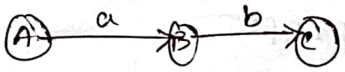
(i) a+b



or



(ii) (ab)



(iii) a*

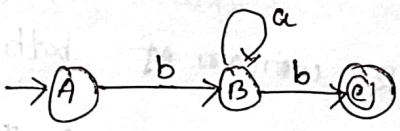
Ex: convert the regular expression to eq. Finite Automata —

(1) ba*b

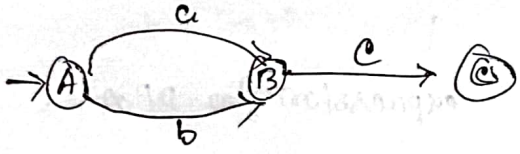
(2) (a+b)c

(3) a(bc)*

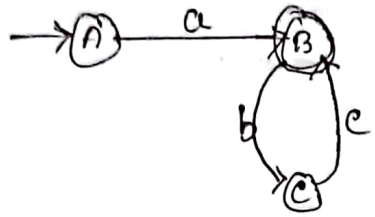
1) ba*b bb, bab, baab, ---



(2) (a+b)c , ac, bc



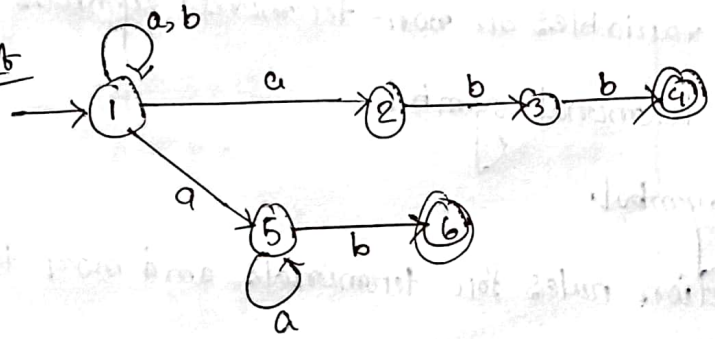
③ $a(bc)^*$ $\{a, abc, abcbc, abcbcbc---\}$



Ex: convert R.E to F.A.

$(a|b)^*(abb|a^*b)$ Here $|, +$ are same.

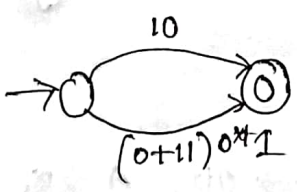
NFA

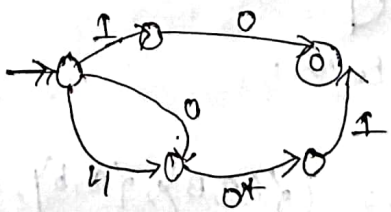


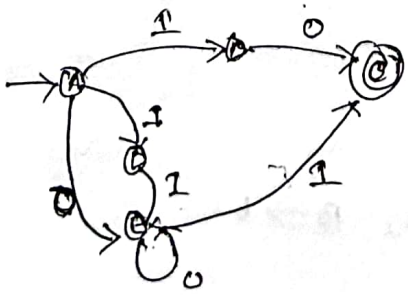
$a+ = \{a, aa, aaa, ---\}$

$a^* = \{\epsilon, a, aa, aaa, ---\}$

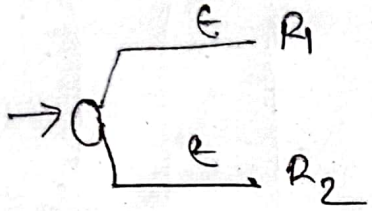
Ex: $10+(0+11)0^*1$





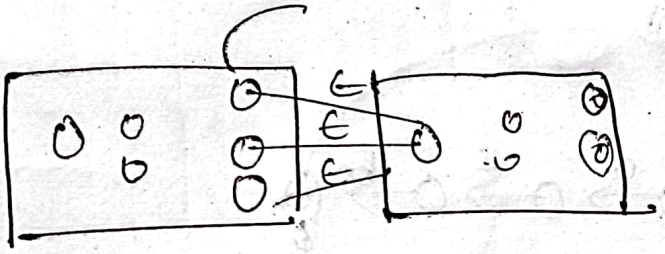


$N = R_1 \cup R_2 = R_1 | R_2$

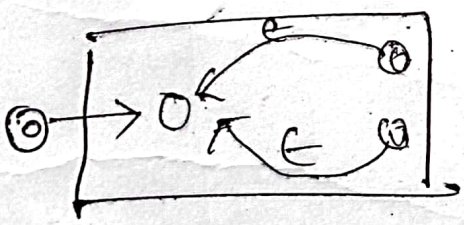


N2 $R_1 \circ R_2$

or $R_1 R_2$



N2 R^+



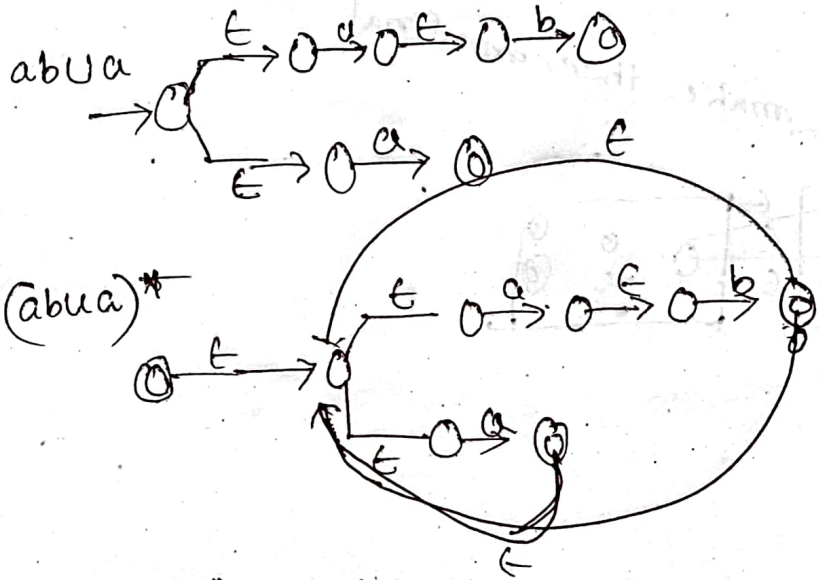
Regular Expression to NFA:

$(ab \cup a)^*$.

✓ a → 0 —a→ (0)

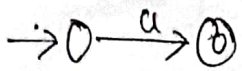
✓ b → 0 —b→ (0)

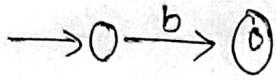
ab ⇒ 0 —a→ 0 —ϵ→ 0 —b→ (0)

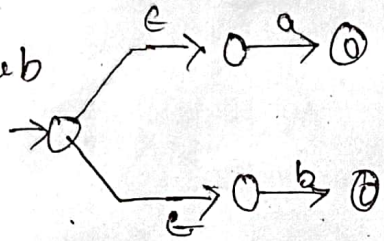


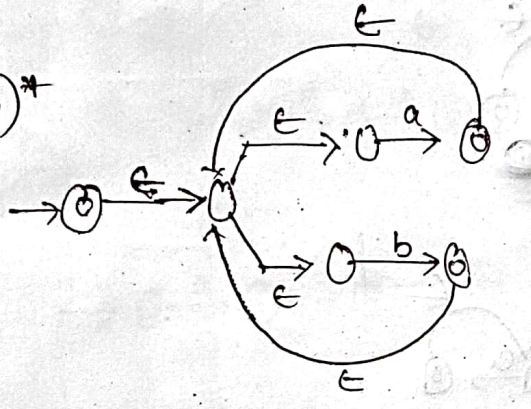
$(abua)^*$ = {ϵ, ab, a, ---}

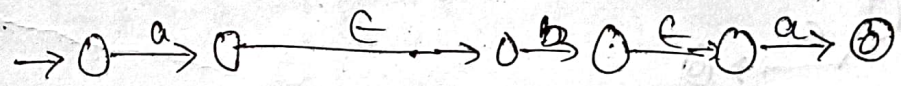
Ex $(a \cup b)^* aba$

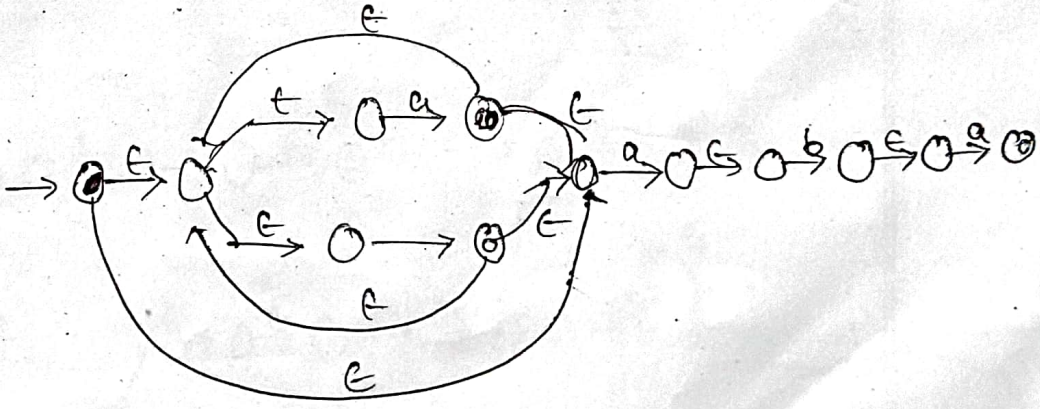
a 

b 

$a \cup b$ 

$(a \cup b)^*$ 

aba 

$(a \cup b)^* aba$ 

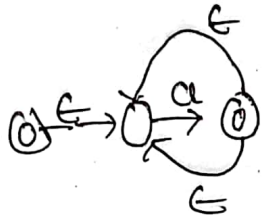
Ex:3 $ba^* + \epsilon$

$\epsilon \rightarrow$ (0)

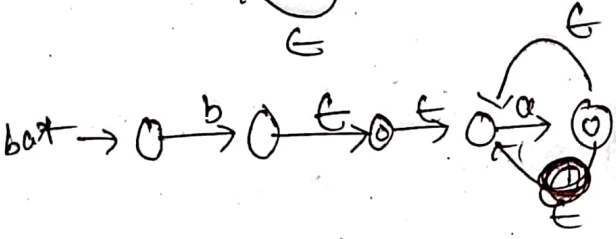
$a \rightarrow$ (0) —a→ (0)

$b \rightarrow$ (0) —b→ (0)

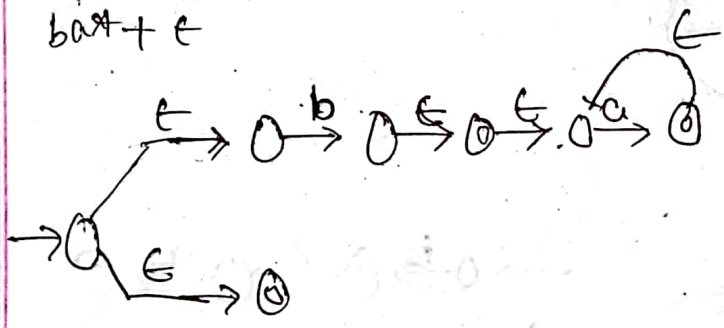
a^*



ba^*

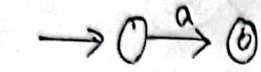


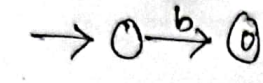
$ba^* + \epsilon$

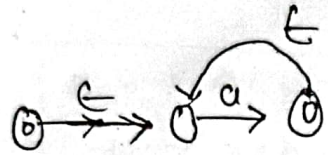


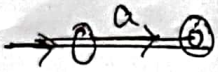
Ex: $a^+ \cup (ab)^+$

$= aa^* \cup ab(ab)^*$

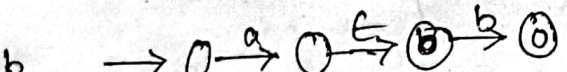
(a) 

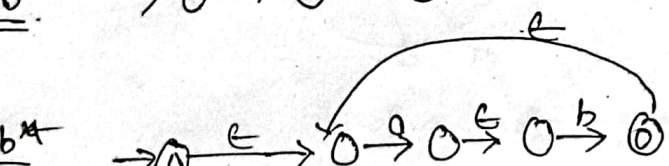
b 

a^* 

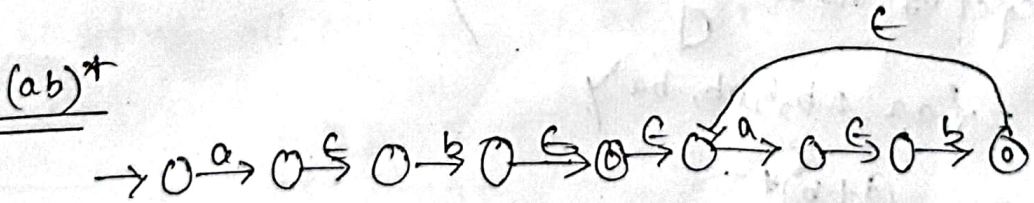


aa^*

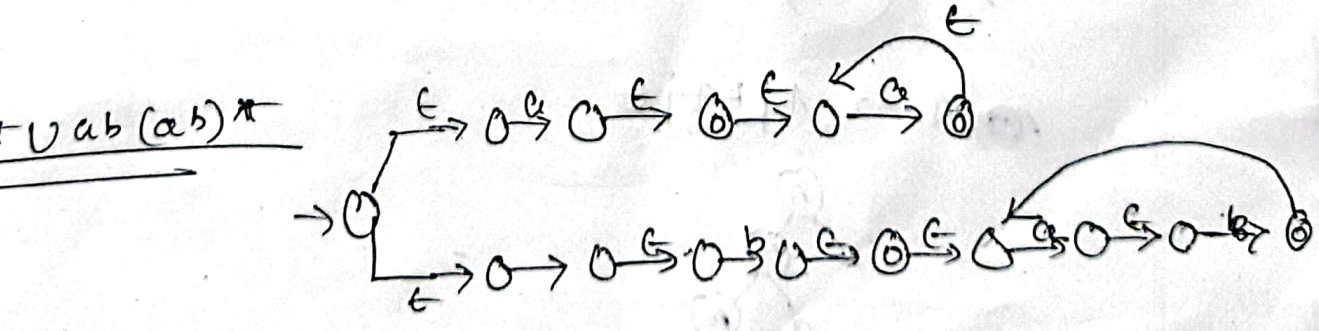
ab 

ab^* 

$ab(ab)^*$



$aa^* \cup ab(ab)^*$



Equivalence of Two finite Automata

Pumping lemma — simple paragraph

Regular Grammar

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

V = set of variables or non-terminal symbols

T = set of terminal symbol.

S = start symbol.

P = Production rules for terminals and non terminals.

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

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

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

$T = \{a, b\}$

$S = S$

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

Eg:

$S \rightarrow AB$

$\rightarrow aB$

$\rightarrow ab$

Regular grammer can be divided into two types:

Right linear grammer

A grammer is said to be right linear if all production are of form

$A \rightarrow xB$

$A \rightarrow x$

where, $A, B \in V$

$x \in T$.

Left linear Grammer

A Grammer is said to be left linear if all production are of the form —

$A \rightarrow Bx$

$A \rightarrow x$

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

Eg: $S \rightarrow abS \mid b$ (Non terminal) ——→ Right linear.

$S \rightarrow Sbb \mid b$ —— Left ».

(#) Derivation from a grammer: The set of all strings that can be derived from Grammer is said to be the language generated by from that grammer

Ex 1: consider the grammer $G_1 = (\{S, A\}, \{a, b\}, S, \{S \to aAb, aA \to aaAb, A \to \epsilon\})$

$S \to \underline{aA}b$ [by $S \to aAb$]

$\to aa\underline{Ab}b$ [by $aA \to aaAb$

$\to aaa\underline{Ab}bb$ ["]

$\to aaabbb$ [$A \to \epsilon$]

Ex 2: $G_2 = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to a, B \to b\})$

$S \to AB$

$\to ab$

$L(G_2) = \{ab\}$

Ex-3: $G_3 = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to aA \mid a, B \to bB \mid b\})$

$S \to AB$

$\to ab$

$S \to AB$

$\to aAbB$

$\to aabb$

$S \to AB$

$\to aAb$

$\to aab$

$S \to AB$

$\to abB$

$\to abb$

$L(G_3) = \{ab, a^2b^2, a^2b, ab^2, ---\}$

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

context Free Languages.

In formal language theory, a context free language generated by some context free Grammer.

The set of all CFL is identical to the set of languages accepted by push down acetomata.

context free grammer is defined by 4 tuples as $G = (v, \Sigma, s, P)$

where, v = set of variables or non-terminal symbol.

Σ = set of terminal symbol.

S = start symbol.

P = production rules.

context free grammer has the production rules of the form

$$A \rightarrow \alpha$$

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

Ex: For generating a language that generates equal numbers of a's and b's in the form $a^n b^n$, the CFG will be defined as,

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

$S \rightarrow aAb$

$\rightarrow aaAbb$

$\rightarrow aabb$

$= a^n b^n$

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

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

② Replace the variables with appropriate production. Repeat until the string is generated or until no production are left.

Ex: Verify whether the Grammar $S \to 0B \mid 1A$,
$A \to 0 \mid 0S \mid 1AA \mid \wedge$
$B \to 1 \mid 1S \mid 0BB$
generates the string 00110101

$S \to 0B$ [S → 0B]
$\to 00BB$ [B → 0BB]
$\to 001B$ [B → 1]
$\to 0011S$ [B → 1S]
$\to 00110B$ [S → 0B]
$\to 001101S$ [B → 1S]
$\to 0011010B$ [S → 0B]
$\to 00110101$ [B → 1]

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.

Ex: For the Grammar $G = \{V, T, P, S\}$ where,

$S \to 0B$

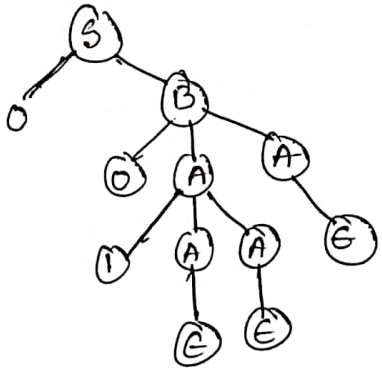
$A \to 1AA \mid \in$

$B \to 0AA$

Root vertex: must be labelled by the start symbol.

vertex: labelled by Non-terminal symbol.

Leaves: labelled " Terminal " or $\in$.



Left Derivation tree:

A left derivation tree is obtained by applying production rules to the leftmost variable in each step

Right derivation tree

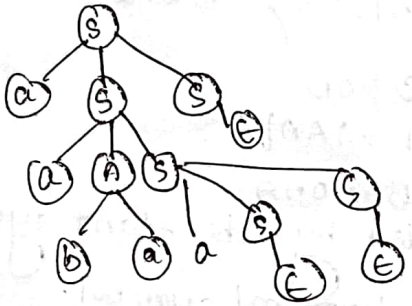
- rightmost variable in each step.

Ex: For generating the string aabaa from the grammar

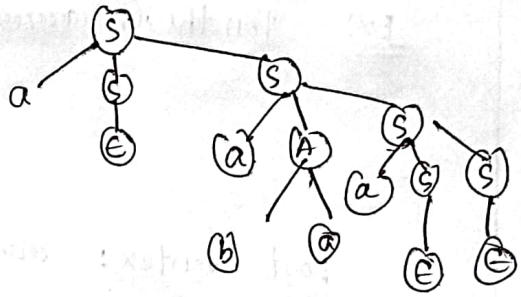
$S \rightarrow aAS \mid aSS \mid \epsilon,$

$A \rightarrow SbA \mid ba.$

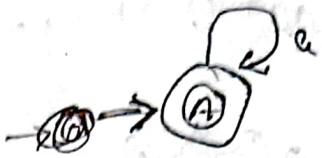
Left



Right



Construction of grammer



Ex:

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

$= \{\epsilon, a, aa, aaa, \dots\}$

a থাকলে A তে আসবে।

(1) $A \Rightarrow aA \mid \epsilon$ $[a^*]$

(2) $A \Rightarrow bA \mid \epsilon$ $[b^*]$

$A \Rightarrow cA \mid \epsilon$ $[c^*]$

Ex: 2 $L = \{a^n \mid n \geq 1\}$

$= \{a, aa, aaa, \dots\}$

$= a^+$

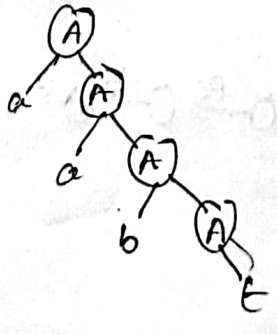
$A \Rightarrow aA \mid a$

Ex 3: Set of all string over a, b

$= \{a, aa, ab, b, bb, ba\}$

$= (a+b)^*$

(1) $A \rightarrow aA \mid bA \mid \epsilon$



Ex-4

CFG for set of all strings which length at least 2

$L = \{aa, ab, ba, bb, aaa, ---\}$

$= \underbrace{(a+b)}_{A}\underbrace{(a+b)}_{A}\underbrace{(a+b)^*}_{B}$

a^*

$A \to aA \mid \epsilon$

$(a+b)^*$

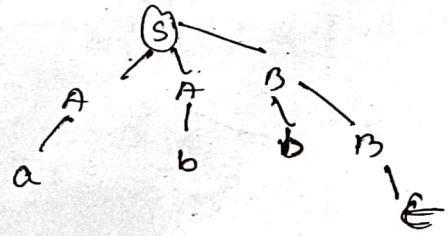
$A \to aA \mid bA \mid \epsilon$.

$S \to AAB$

$A \to a \mid b$

$B \to aB \mid bB \mid \epsilon$

abb



Ex:5: set of all strings of at least 3 0's.

$E0E0E0E$

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

10000

$S \to E0E0E0E$

$E \to 0E \mid 1E \mid \epsilon$

Ex: set of all strings which length at ~~most~~ most 2.

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

$$= \underbrace{(a+b+\epsilon)}_{A}(a+b+\epsilon)$$

$S \rightarrow AA$

$A \rightarrow a \mid b \mid \epsilon$

~~aaa~~

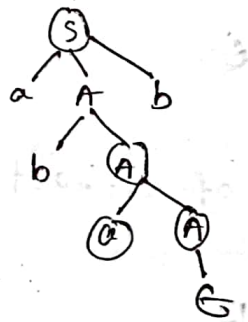
Ex-7: starts with a and Ends with b

$$a\underbrace{(a+b)^*}_{A}b$$

$S \rightarrow aAb$

$A \rightarrow aA \mid bA \mid \epsilon$

abab



Ex: starts and Ends with different symbol.

$$a\,(a+b)^*\,b \;\Big|\; b\,(a+b)^*\,a$$

$$= \underbrace{a(a+b)^*b}_{A} + \underbrace{b(a+b)^*a}_{A}$$

$$S \to aAb \mid bAa$$
$$A \to aA \mid bA \mid \epsilon$$

Ex: starts and Ends with same symbol.

$$a\,(a+b)^*\,a \mid b\,(a+b)^*\,b \mid \epsilon \mid a \mid b$$

$$= \underbrace{a(a+b)^*a}_{A} + \underbrace{b(a+b)^*b}_{A} + \epsilon + a + b$$

$$S \to aAa \mid bAb \mid \epsilon \mid a \mid b$$
$$A \to aA \mid bA \mid \epsilon$$

Ex: Even length string: $\underbrace{(a+b)(a+b)}_{A}$ $\{(a+b)(a+b)\}^*$

$$S \to BS \mid \epsilon$$
$$B \to AA$$
$$A \to a \mid b$$

Pushdown Automata.

A pushdown Automata (PDA) is a way to implement a context free grammer in a similar way we design Finite Automata for regular grammer.

→ it is powerful than FSM.

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

→ PDA = Finite state machine + A stack.

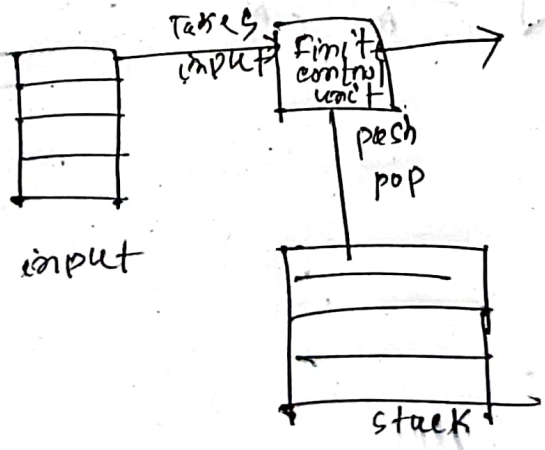
stack . push pop

(*) A pushdown Automata has 3 components —

(i) An input tape.

(ii) A Finite control unit

(3) A stack with infinite size.



(#) A push down Automata is formally defined by 7 tuples.

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

Q = A finite set of states

Σ = A finite set of input symbols,

Γ = A finite stack alphabet.

δ = The Transition function

q_0 = The start state.

z_0 = The start stack symbol.

F = The final state.

input, stack

$\delta(q, a, X) \rightarrow (q_1, \gamma)$

state, new state, pushed alphabet.

δ takes as argument a triple $\delta(q, a, X)$ where —

(i) q is a state in Q

(ii) a is either an input symbol in Σ or $a = \epsilon$.

(iii) X is stack symbol that is a member of Γ.

The output of δ is finite set of pairs ~~(p, γ)~~ (p, γ) where

p is a new state

γ is a string of stack symbol that replaces X at the top of the stack.

Eg. if $Yz\epsilon$ then stack is popped.

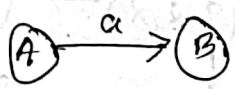
$Yz\,X$ " " " is changed

$Yz\,Yx$ then x is replaced by z and y is replaced pushed onto the stack.

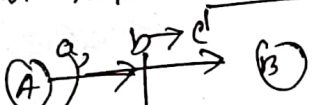
· push down automata

Graphical Notation

Finite state Machine



push down Automata



This is pushed onto the stack.

ϵ means nothing is pushed.

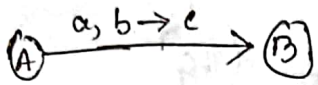
input

(also can be ϵ)

symbol on the top of the stack. This symbol is popped.

ϵ means this symbol neither read nor popped

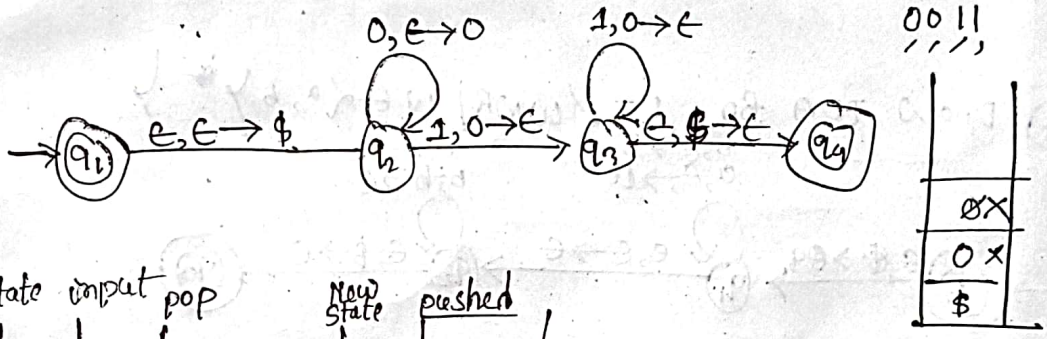
$(q, a, x) \rightarrow (q, \gamma)$



$(A, a, b) \rightarrow (B, c)$

construction

(*) construct a PDA that accepts, $L = \{0^m 1^m \mid m \geq 0\}$

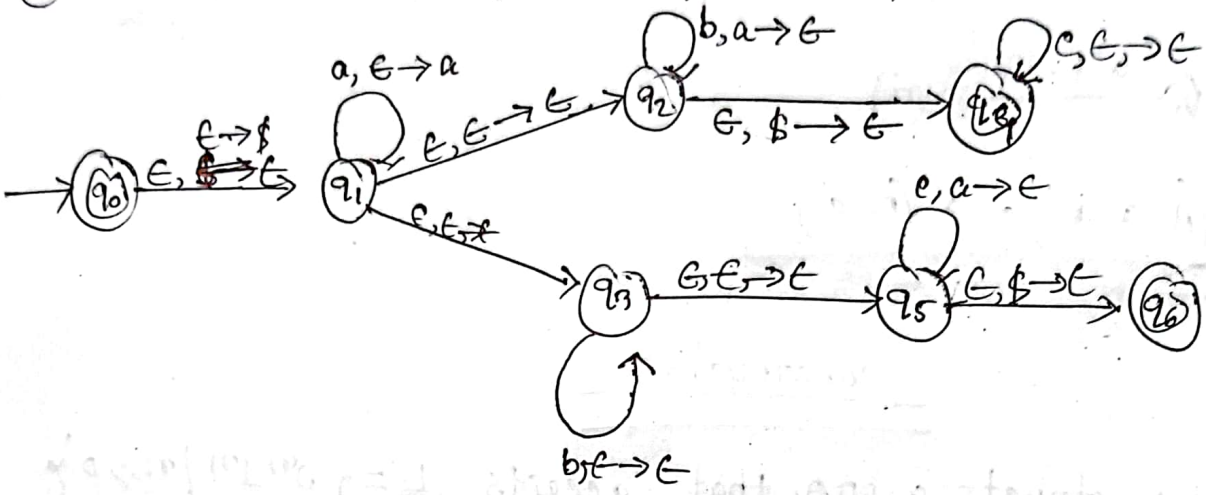


state	input	pop	New state	pushed
q_1	∈	∈	q_2	$
q_2	0	∈	q_2	0
q_2	1	0	q_3	∈
q_3	1	0	q_3	∈
q_3	∈	$	q_4	∈

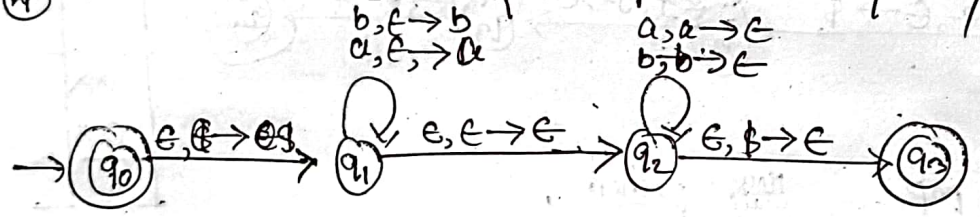
Transition Function

$a = c$
$a = b$

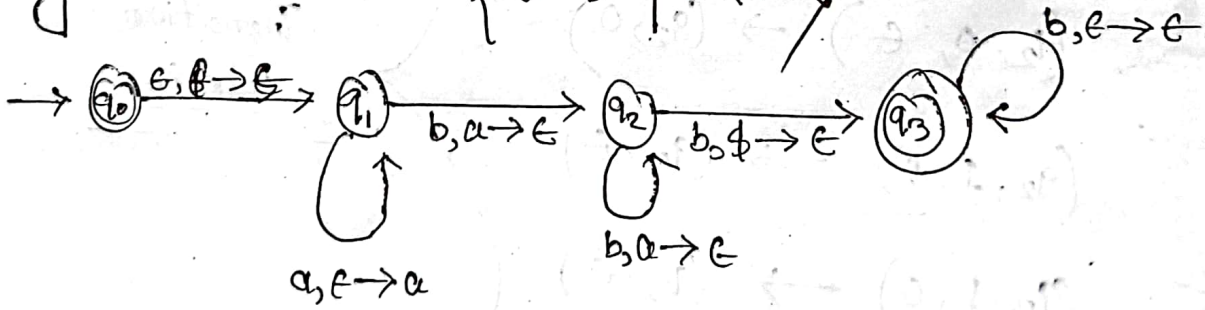
(M) Draw PDA for $L = \{ a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k \}$



(M) Draw PDA for $L = \{ ww^R \mid w \in (a, b)^* \}$

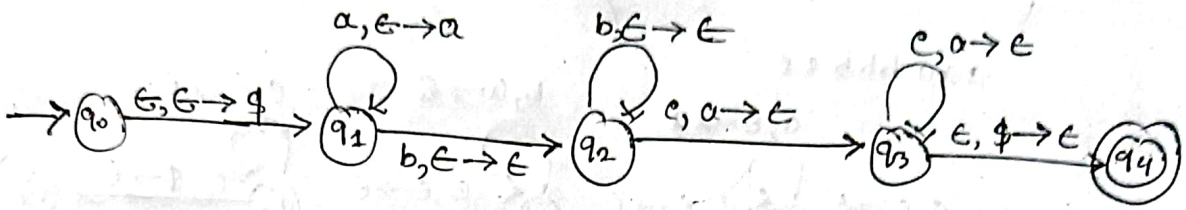


(M) Design PDA for $L = \{ a^m b^n \mid m < n \}$

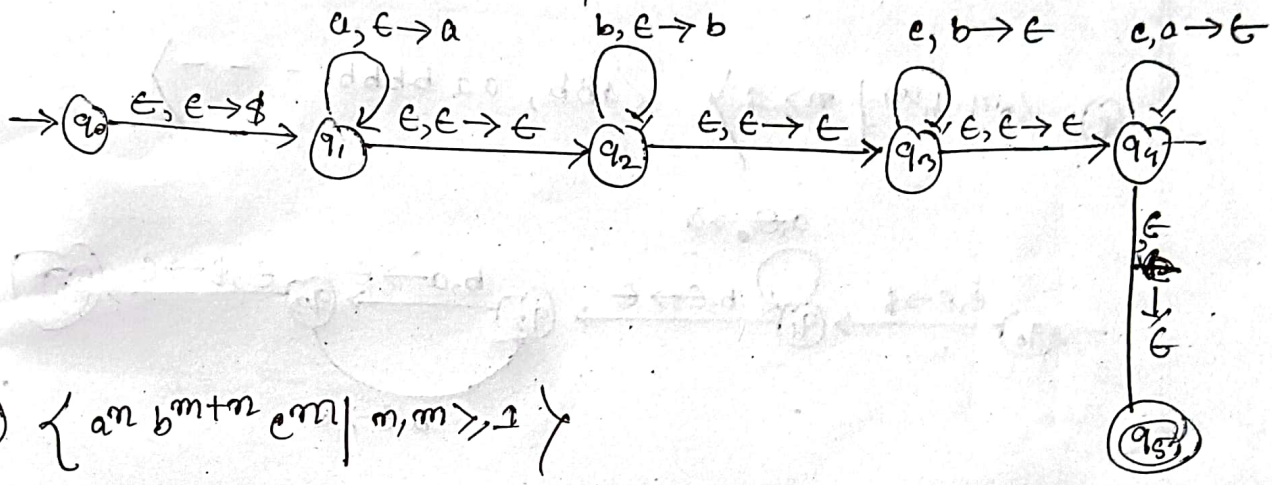


aabbbcc $a^2b^3c^{2+3}$
$a^2b^3c^5$

(*) $\{a^m b^n c^m \mid m,n \geq 1\}$



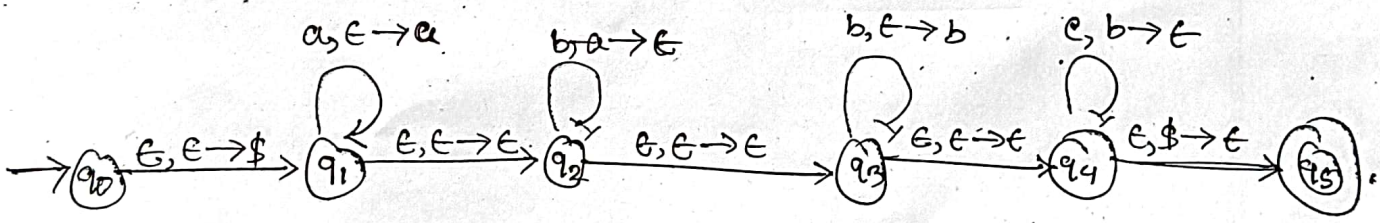
(*) $\{a^m, b^n, c^{m+n} \mid m,n \geq 1\}$



(*) $\{a^m b^{m+n} c^m \mid m,n \geq 1\}$

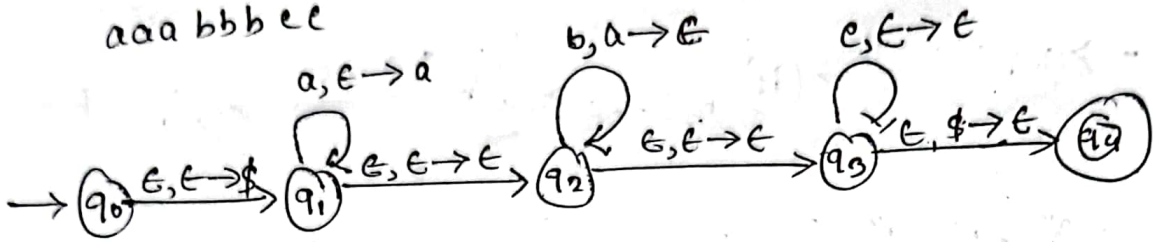
$a^m b^m b^n c^m$
$= a^m b^n b^m c^m$

push pop push pop
a a, b b, b b b, c c c

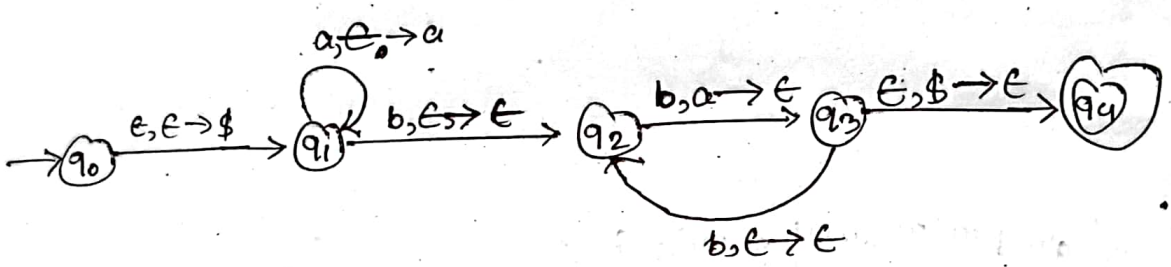


(*) $\{a^m b^m c^m \mid m, m \geq 1\}$

aaa bbb cc



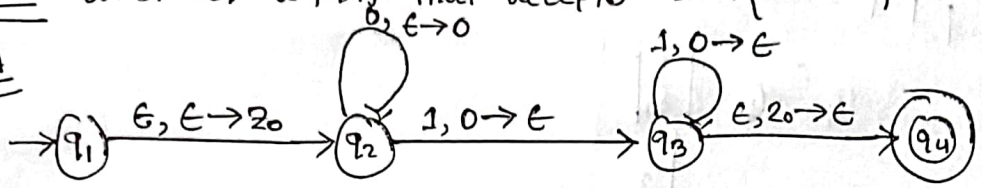
(*) $\{a^m b^{2m} \mid m \geq 1\}$ $\{abb, aabbbb, ---\}$



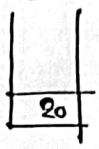
PDA continues.

Ex: construct a PDA that accepts $L = \{0^n 1^n \mid n \geq 0\}$

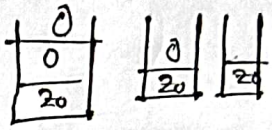
NPDA



0011



There are two cases when a string will be accepted —



(i) reach final state.

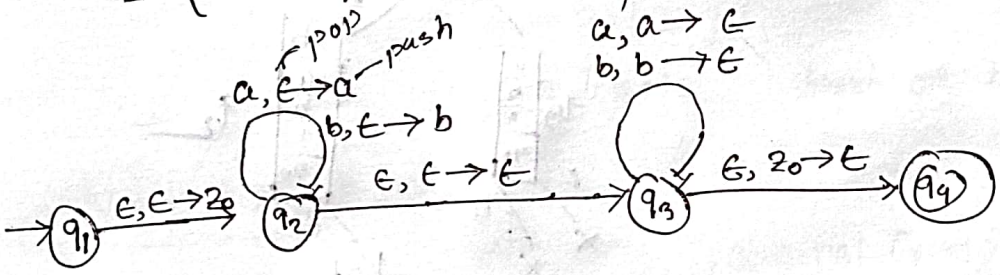
(ii) stack is empty.

Ex1: (Even palindrome)

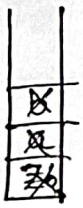
construct a PDA that accepts Even palindrome of the form —

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

∈a∈b∈b∈a∈

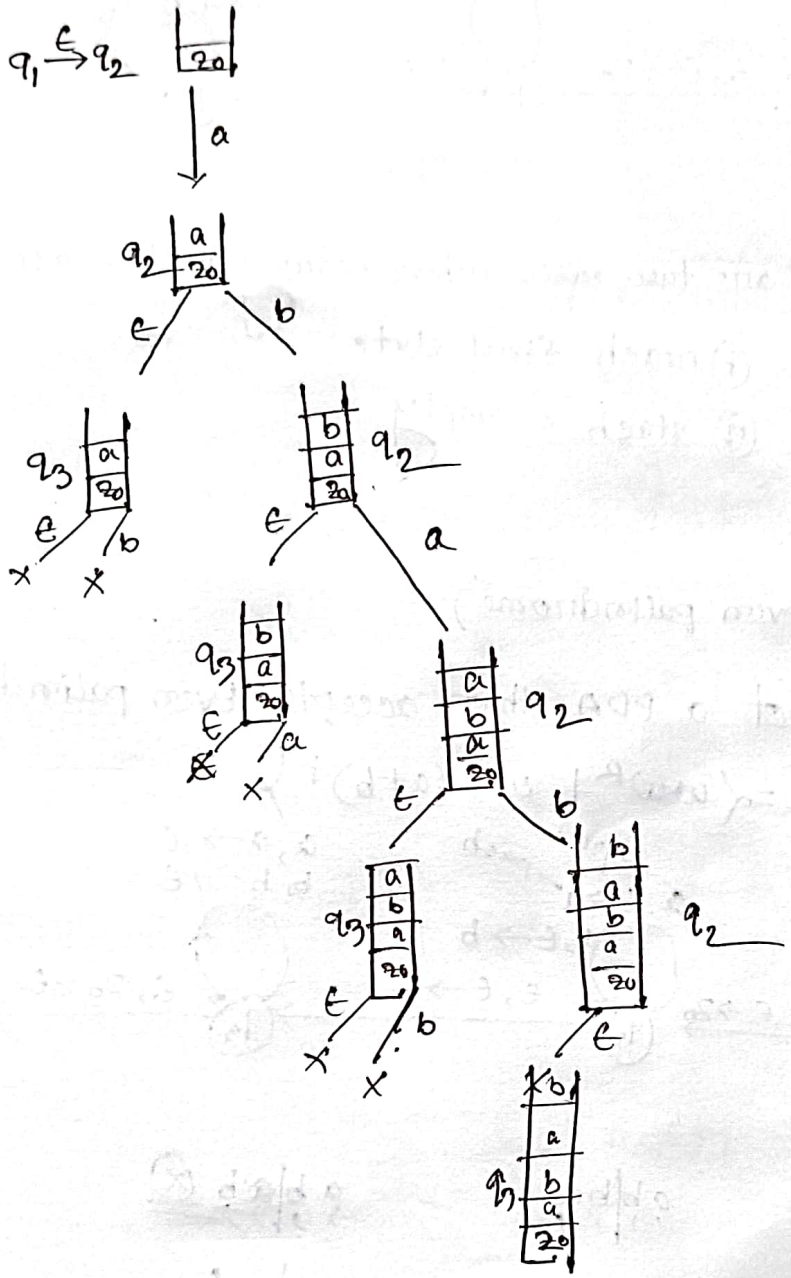


a b | b a



a b | a b ✗

(*) ∈ a ∈ b ∈ a ∈ b ∈


q1 → q2
Z0
a
q2
a
Z0
∈
b
q3
a
Z0
∈
b
X
X
b
a
Z0
q2
∈
a
q3
b
a
Z0
∈
a
X
a
b
a
Z0
q2
∈
b
a
q3
b
a
Z0
∈
b
X
X
b
b
a
b
a
Z0
q2
∈
b
a
q3
b
a
Z0

Equivalence of CFG and PDA

Theorem: A language is context free iff some push down automata recognizes it.

part 1: Given a CFG, show how to construct a PDA that recognize it.

part 2: Given a PDA, show how to construct a CFG that recognizes it.

Ex: Given Grammer,

$S \rightarrow BS \mid A$

$A \rightarrow 0A \mid \epsilon$

$B \rightarrow BB1 \mid 2$ find or build PDA.

left most derivation:

$\rightarrow S$

$\rightarrow BS$

$\rightarrow BB1S$

$\rightarrow 2B1S$

$\rightarrow 221S$

$\rightarrow 221A$

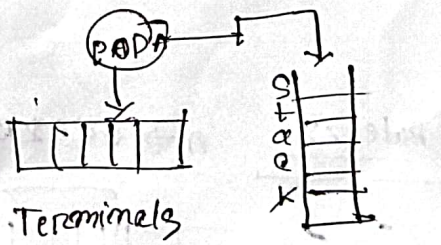
$\rightarrow 221\epsilon$

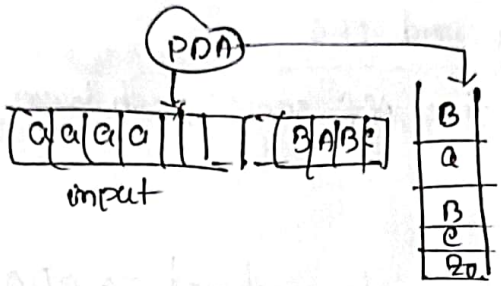
$\rightarrow 221$

General form

aaaa BaBc

Terminals Rest





Eg: Rule $B \rightarrow ASA \times BA$ $\longrightarrow$ a a a a A S A X B A a B C

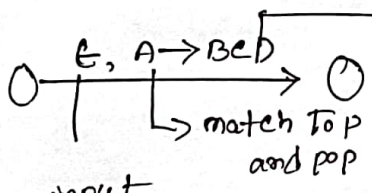
→ match stack top to a rule.

→ pop stack.

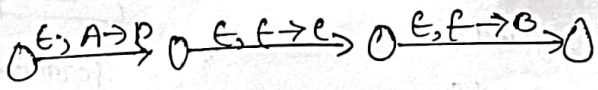
→ push Right-Hand side of Rule onto stack.

(A) Rule → $A \rightarrow BCD$ add this to PDA.

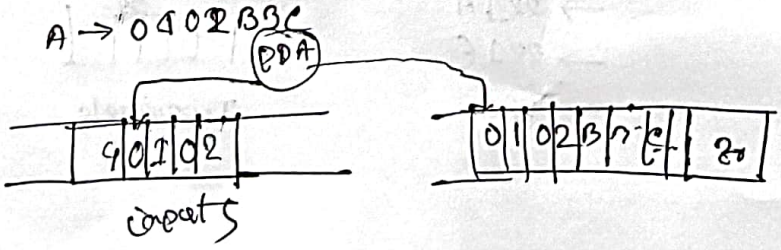
→ Right hand side pushed onto the stack.



Actually

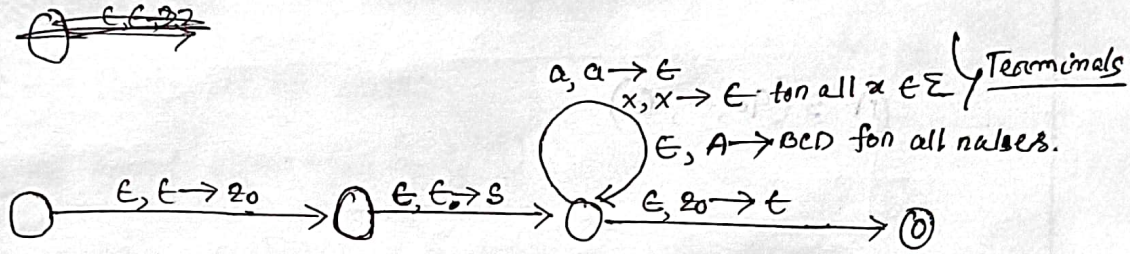


(A) Rule→2 $A \rightarrow 0\,A\,0\,Z\,B\,B\,C$



$0,0 \to \epsilon$
$1,1 \to \epsilon$
$2,2 \to \epsilon$
$x,x \to \epsilon$ for all $x \in \Sigma$.

Final PDA



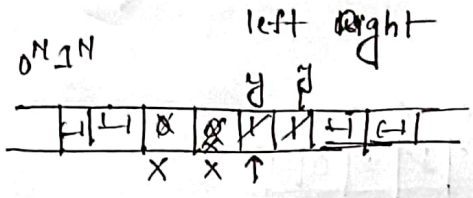
part-2: Given PDA, show how to construct CFG.

step 1: simplify the PDA.

step 2: Build the CFG.

Turing Machine

Formal Definition: Read write possible.



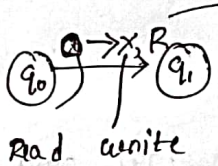
7 tuples $(Q, \Sigma, \Gamma, \delta, q_0, q_{accep}, q_{reject})$

(i) Q — finite set of states.

(ii) Σ — input alphabets ($\sqcup$ / B)

(3) Γ — Tape Alphabet ($\Sigma \in \Gamma$, $\sqcup \in \Gamma$, $xy \in \Gamma$)

(4) $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \{L, R\}$



$(q_0, 0) \rightarrow (q_1, X, R)$

কোন state থেকে Tape symbol read করে কোন state এ যাচ্ছি এবং কি write করছি এবং কোন দিকে move.

(5) q_0 — starting state.

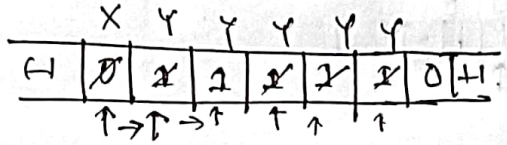
(6) q_{accep} — Accept state

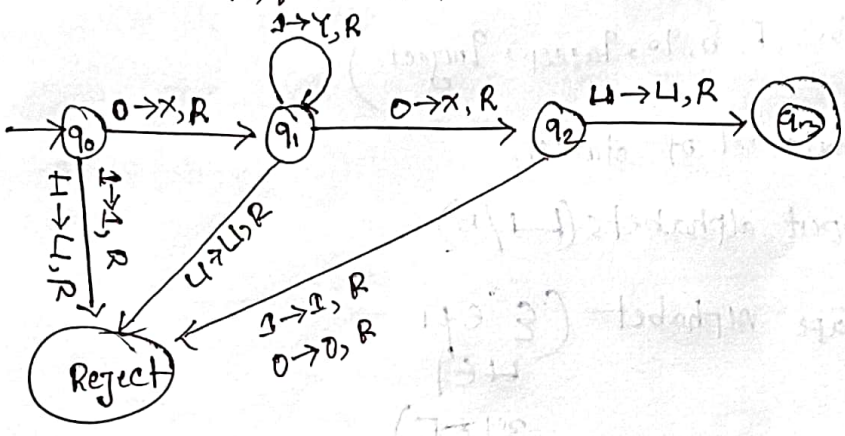
(7) q_{reject} — Reject state.

Construct

(*) Design a Turing machine which recognize the Language

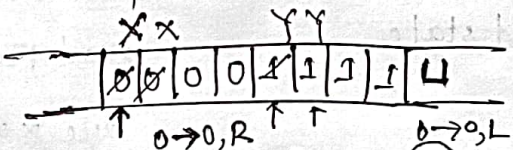
$L = 0 1^* 0$

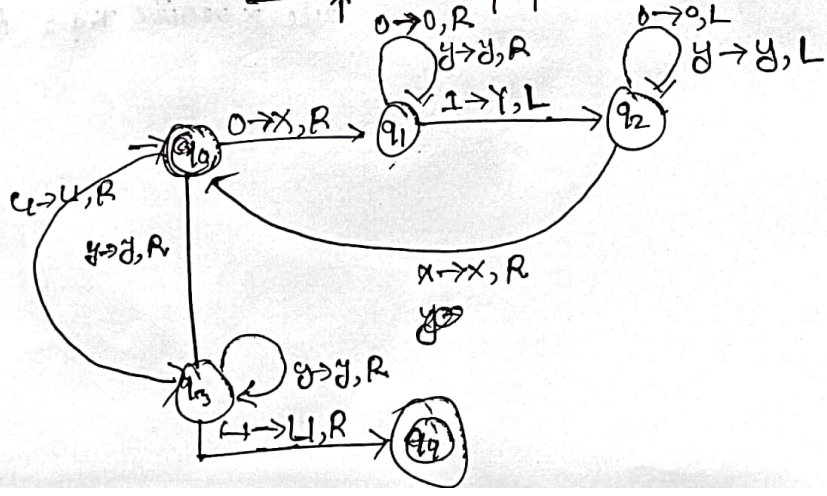




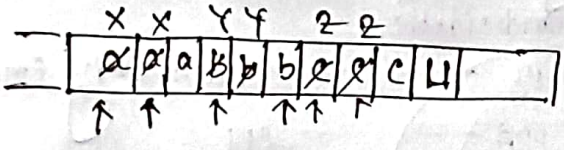
(*) Design and recognize Turing machine which recognize

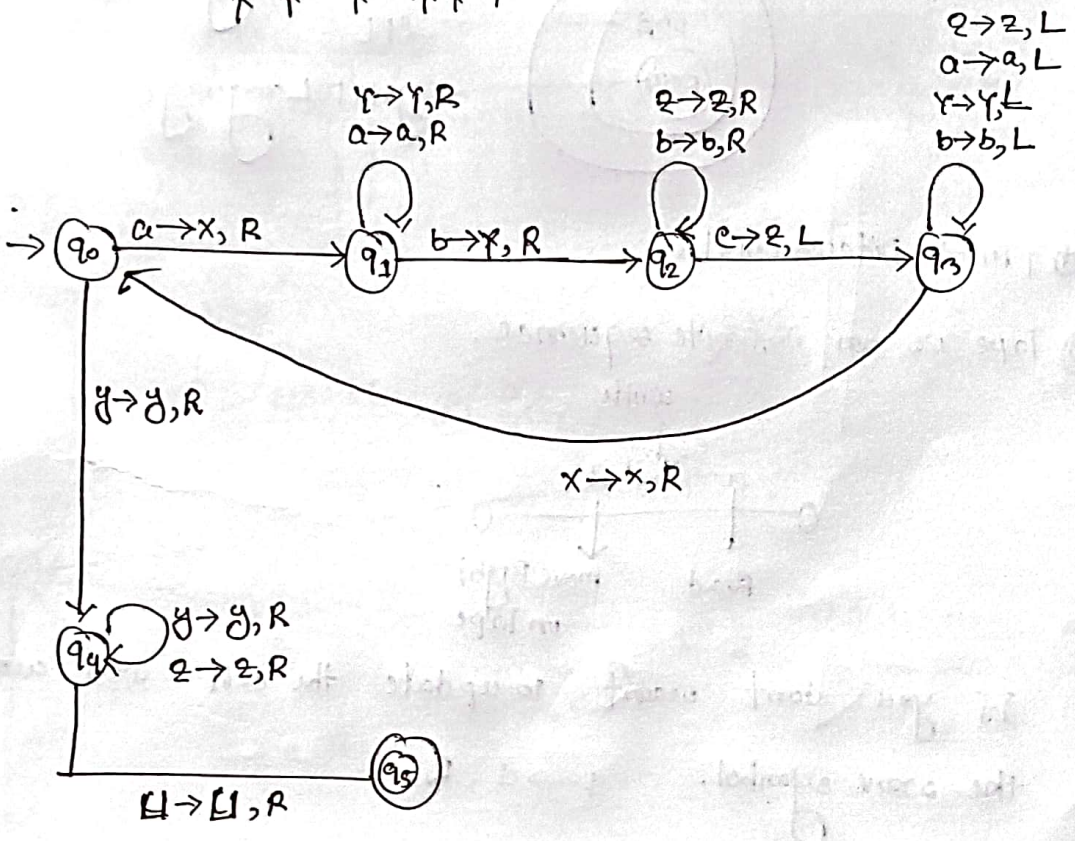
The language $L = \{0^N 1^N ; N \geq 0\}$





Ex: $L = \{a^m b^m c^m \mid m \geq 1\}$

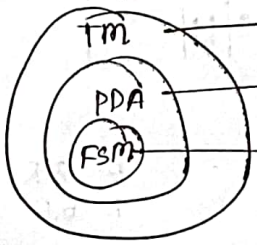




Neso Academy

Turing Machine

Undecidable



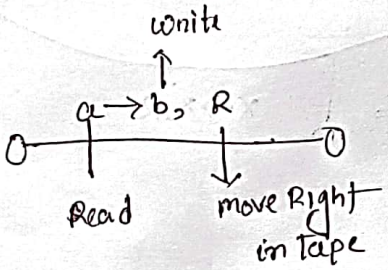
Recursively Enumerable Language.

CFL

Regular Language

(A) TM → Deterministic.

(B) Tape is an infinite sequence.



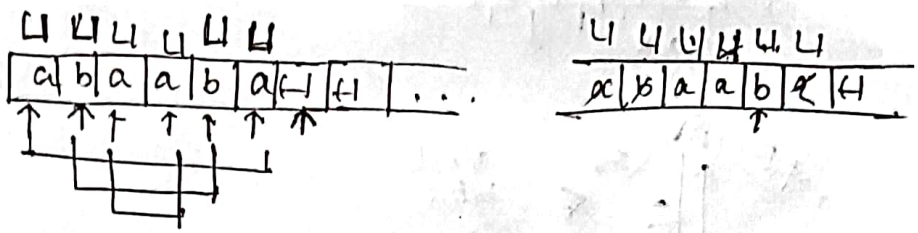
If you dont want to update the cell just write the same symbol. 1 → 1, R.

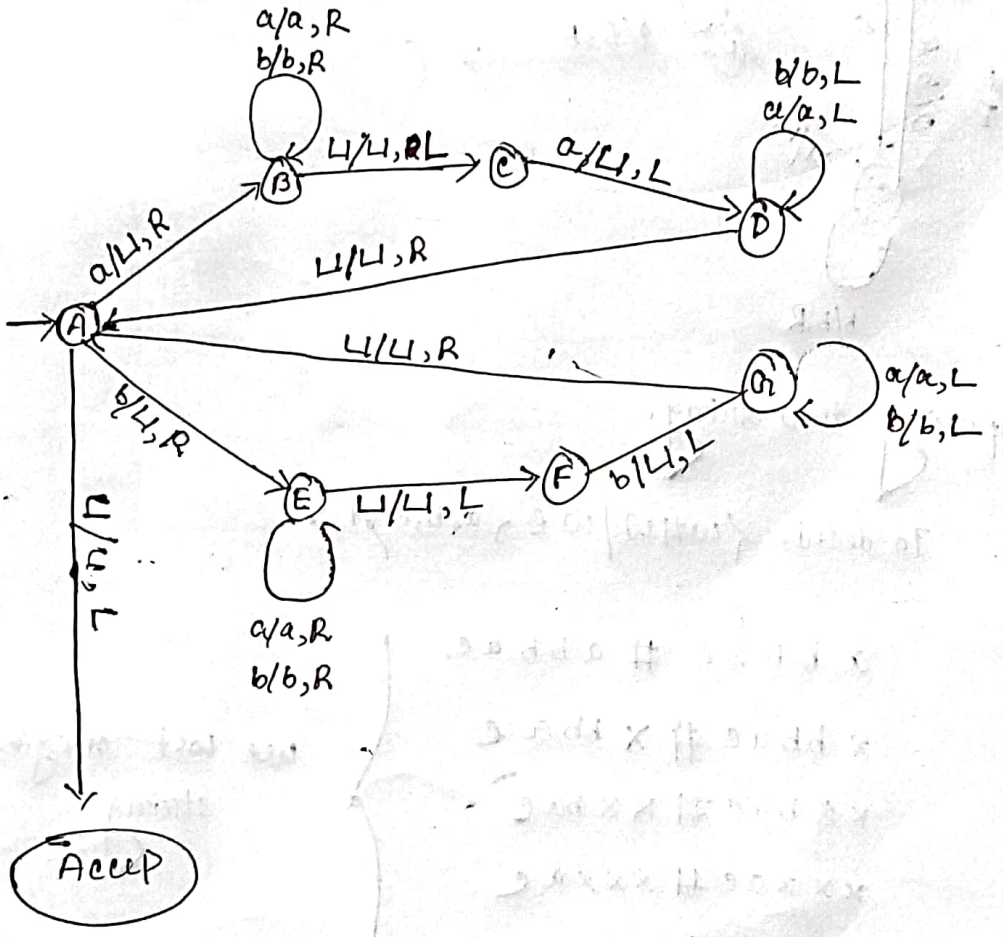
$$\delta(q_0, a) \rightarrow (q_1, 1, R)$$

A Language L and Σ is said to be Recursively Enumerable if there exist a turing machine that accepts it.

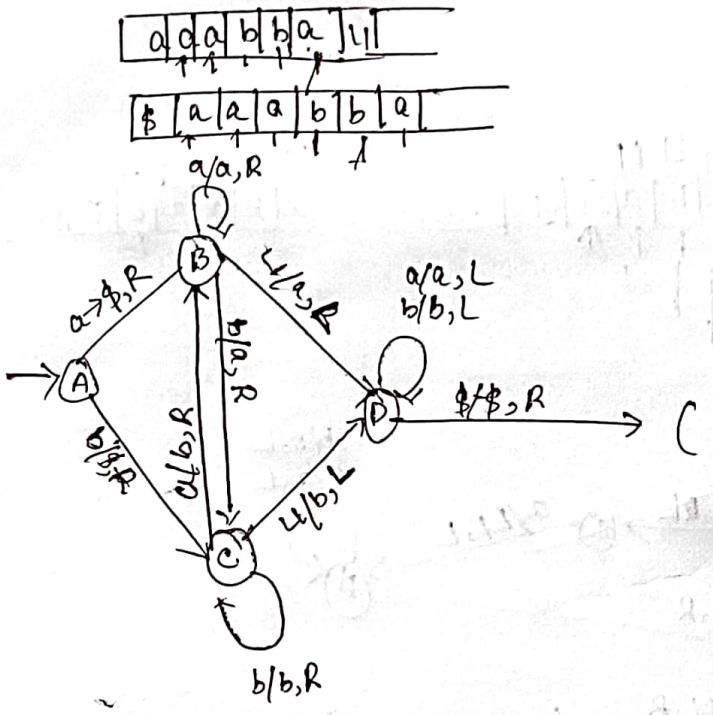
④ Design a Turing machine That accepts even palindromes over the alphabet $\Sigma = \{a, b\}$

Eg: abaaba





④ putting $ symbol at the starting turing machine. —



⑤ comparing two strings.

To decide $\{w\#w \mid w \in \{a,b,c\}^*\}$

a b b a c # a b b a c
x b b a c # x b b a c
x x b a c # x x b a c
x x x a c # x x x a c
x x x x c # x x x x c
x x x x x # x x x x x

} we lost original string.

To not loose the string replace with unique symbol instead of replacing with some symbol.

a → P
b → q
c → ∩

a b b a c # a b b a c
P q q P ∩ # P q q P ∩