

06/01/21

Basic Concepts of Automata Theory

There are few concepts that explain the automata theory, they are

- (i) Symbol
- (ii) Alphabet
- (iii) Strings
- (iv) Languages
- (v) Grammar

(i) Symbol :-

A symbol is an abstract entity. Letters and digits are examples of frequently used symbols.

ex:- letters, digits.

(ii) Alphabet :-

Alphabet is a collection of symbols. Alphabet is a finite set of symbols and denoted by Σ .

ex:- $\Sigma = \{1, 2, 3\}$

(iii) Strings :-

A string is a finite sequence of symbols obtained from an alphabet.

ex:- $\Sigma = \{a, b, c\}$

$= \{ab, abc, bc, ac\}$

operations on Strings:-

(i) Length of String :- $|w|$

The length of String is denoted by $|w|$.
It is the no. of Symbols composing the String.

ex:- $w = abcd$

$$|w| = 4$$

(ii) Empty String :-

The empty string is denoted by ϵ (or) λ .
If the string consisting of zero symbols.

$$\text{length of } w = 0$$

(iii) Prefix of the String :-

Any number of leading symbols including the string is known as "Prefix of the string".

ex:- $w = abcd$

then, prefixes are $a, ab, abc, abcd$.

(iv) Suffix of String :-

The trailing symbols of a string are called as "Suffix of a string".

ex:- $w = abcd$

suffixes are $d, cd, bcd, abcd$.

Proper prefix:-

Same as prefix but excluding string itself.

ex:- $w = abcd$

proper prefixes are a, ab, abc

Proper suffix:-

Same as suffix but excluding string itself.

ex:- $w = abcd$

proper suffixes are d, cd, bcd .

④ Language :- (L)

A Language is a set of strings of symbols from an alphabet. A Language is a finite and non-empty set of strings. It is denoted by ' L '. All these strings are formed from an alphabet.

→ Language notations:-

(i) $L(M)$:-

It is a language defined by a machine (M) that accepts set of strings.

(ii) $L(G)$:-

It is a language defined by the Grammar G that recognise set of strings.

(iii) $L(R)$:-

It is a language defined by the Regular expression that represents the set of strings.

* Let, $\Sigma = \{z\}$ then the language of all possible strings is given by.

$$L = \{\epsilon, z, zz, zzz, \dots\}$$

* The language of all possible even length strings over $\Sigma = \{z\}$

(A) $L = \{zz, zzzz, \dots\}$

$$= \{z^{2n} \mid \text{where } n \geq 1\}$$

Operations on Language

(i) Union :-

It is a simple operation on two languages. Let L_1, L_2 be two languages then the union is defined by $L_1 \cup L_2$.

* Mathematically, the Union of two languages is defined by

$$L_1 \cup L_2 = \{x \mid x \in L_1 \text{ or } x \in L_2\}$$

ex:-

$$L_1 = \{0, 01, 011\}, L_2 = \{1, 10, 1011\}$$

$$L_1 \cup L_2 = \{0, 01, 011, 1, 10, 1011\}$$

08/01/20

ii) Intersection

It is a simple operation on two languages. Let L_1 and L_2 be two languages, then their intersection is denoted by $L_1 \cap L_2$.

* Mathematically, $L_1 \cap L_2$ is defined as:

$$\Rightarrow L_1 \cap L_2 = \{x / x \in L_1 \text{ and } x \in L_2\}$$

ex:

$$L_1 = \{a, ab, aab\}$$

$$L_2 = \{a, aba, abb\}$$

$$L_1 \cap L_2 = \{a\}$$

iii) Complementation

It is a simple operation performed on a single language. Let L be a language, then the complement of L is denoted by $\bar{L}$.

$$\bar{L} \text{ (or) } L^c$$

$$\Rightarrow L^c = \Sigma^* - L$$

* Mathematically, the complementation of L is defined as:

$$L^c = \{x / x \in \Sigma^* \text{ and } x \notin L\}$$

ex:-

① If $\Sigma^* = \{a, b\}^*$ and $L = \{a, ab, abb\}$ then

$$L^c = \Sigma^* - L$$

i.e,

$$\Sigma_1^0 = \{\Lambda\} \cup \{\epsilon\}$$

$$\Sigma_1^1 = \{\Lambda, b\}$$

$$\Sigma_1^2 = \{\Lambda, ab, ba\}$$

$$\Sigma_1^3 = \{\Lambda, aab, aba, abb, bba, bab, baa\}$$

$$L^c = \{\Lambda, b, aa, bb, ba, aab, aba, abb, bba, bab, baa\}$$

where

$$\Sigma_1^* = \{\Sigma_1^0, \Sigma_1^1, \Sigma_1^2, \dots\}$$

(iv) Symmetric Difference :-

* It is a simple operation on two languages. Let L_1 & L_2 be two languages then the Symmetric operation on L_1 and L_2 is defined as.

$$L_1 \oplus L_2 = (L_1 \cup L_2) - (L_1 \cap L_2)$$

ex:-

Let L be a language over Σ_1^*
then

$$\begin{aligned} \textcircled{i} L \oplus \emptyset &= (L \cup \emptyset) - (L \cap \emptyset) \\ &= L - \emptyset \\ &= L \end{aligned}$$

$$(ii) L \oplus L = (L \cup L) - (L \cap L)$$

$$= L - L = \emptyset$$

$$(iii) L \oplus \Sigma^* = (L \cup \Sigma^*) - (L \cap \Sigma^*)$$

$$= \Sigma^* - L$$

$$= L^c$$

(v) Concatenation of Languages

It is used to combine two (or) more languages into a single language.

* It is denoted by " $L_1 L_2$ " (or) " $L_1 L_2$ ".

* Mathematically, it is denoted by

$$L_1 L_2 = \{xy / x \in L_1, y \in L_2\}$$

ex:-

$$L_1 = \{a, ab\}$$

$$L_2 = \{b, bb\}$$

$$L_1 L_2 = \{ab, abb, abb, abbb\}$$

(vi) Reversal of a Language

It is similar to reversal of strings. It means reverse all the strings in the language i.e. L^R .

* Mathematically, it is defined as

$$L^R = \{w^R / w \in L\}$$

ex:- $L = \{ab, aab\}$

$$L^R = \{ba, baab\}$$

(vii) Kleen Closure of a language:-

It is a set of strings including null string. It is denoted by " L^* ".

(viii) +ve closure of a language:-

It is a set of strings excluding null string. It is denoted by " L^+ ".

09/01/24

(5) Grammar:-

* It is a finite set of formal rules for generating syntactically correct sentences.

(or) meaningful correct sentences.

* Grammar is basically composed of two elements.

(i) Terminal Symbol

(A symbol which does not derive another symbol & it is final symbol (or) end symbol)

Terminal symbols are those which are the components of sentences generated using a grammar and they are represented by $a, b, c, d, \dots$

(ii) Non-Terminal Symbols :- (A symbol which derives another symbol)

Those symbols which take part in the generation of sentence but are not the component of sentence. These symbols are represented by Capital letters. $A, B, C, D, \dots$

Formal Definition of a Grammar

Any Grammar can be represented by 4 tuples (V, T, P, S) where

V - Variables (or) Non-Terminals.

T - Terminal symbols.

P - Production rules.

S - Starting Variable.

Production rule :

It is a re-write rule specifying a symbol substitution that can be recursively performed to generate new symbol sequences.

* It is of the form, $\alpha \rightarrow \beta$ where

α is Non-Terminal symbol, which can be replaced by β which is a string of Terminal (or) Non-Terminal symbols.

ex:

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

$P = \{ A \rightarrow Aa, A \rightarrow Ab, B \rightarrow b \}$

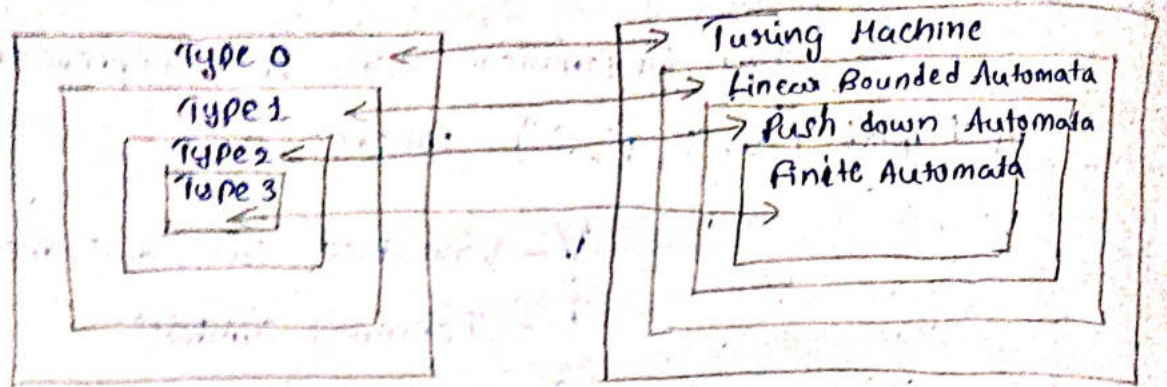
then

$V = \{ A, B \}$

$T = \{ a, b \}$

$S = \{ A \}$

Chomsky hierarchy of Grammar



* According to Chomsky Hierarchy, Grammar is divided into '4' types as follows.

- ① Type 0 is known as "Unrestricted Grammar".
- ② Type 1 is known as "Context Sensitive Grammar".
- ③ Type 2 is known as "Context Free Grammar".
- ④ Type 3 is known as "Regular Grammar".

10/01/24

① Type 0 Grammar:-

- * It is also known as "Unrestricted Grammar".
- * This grammar has no restrictions.
- * They are recognised by "Turing machines".
- * The Languages obtained are known as "Recursively enumerable language".
- * It has productions like $\alpha \rightarrow \beta$

where

$$\alpha \in (V+T)^+$$

$$\beta \in (V+T)^*$$

ex:- $A \rightarrow B, Aa \rightarrow B, A \rightarrow a, A \rightarrow aB,$
 $A \rightarrow \epsilon$

② Type - 1 Grammar

* It is also known as "Context Sensitive Grammar".

* If we apply restriction on type-0 then we get Context Sensitive Grammar (CSG).

* The languages generated are "Context Sensitive Languages (CSL)" by the machine "Linear Bounded Automata".

* The productions are $\alpha \rightarrow \beta$

where,

$$|\alpha| \leq |\beta| \text{ and}$$

$$\alpha \in (V+T)^+$$

$$\beta \in (V+T)^+$$

ex:-

$$A \rightarrow aB, A \rightarrow B, A \rightarrow aB$$

③ Type - 2 Grammar

* It is also known as "Context free Grammar".

* If we apply it generates Context free languages and languages are recognised by "push down automata".

* It should be in Type - 1 Grammar.

* Left side of production can have only one variable & there is no restriction on right hand.

* It is of the form, $A \rightarrow \alpha$
where, $\alpha \in (V+T)^+$

ex:-

$$A \rightarrow aB, A \rightarrow aA, \dots$$

④ Type - 3 Grammar :-

- * It is known as "Regular Grammar".
 - * It ~~gener~~ generates "Regular languages".
 - * It is more restricted form of grammar.
 - * accepted by "finite Automata".
 - * In each rule, left hand side is a non-Terminal and right hand side contains at most ~~one~~ non-Terminal symbol, either right most (or) left most.
- $$\begin{cases} \cdot V \rightarrow VT/T \\ V \rightarrow TV/T \end{cases}$$
- ex:- $A \rightarrow bB, A \rightarrow ab, D \rightarrow Bd$

① Finite Automata :-

- * It is a mathematical model of a system with discrete inputs and outputs.

Defination :-

- * The finite Automata consists of finite set of states and a set of transitions from ~~one~~ state to ^{another} state that occurs on input symbols chosen from an alphabet (Σ).
- * for each input symbol, there is exactly one transition out of each state [possibly back to its state]

* finite automata is 5 tuple form $(Q, \Sigma, \delta, q_0, F)$

where,

Q = finite set of states.

Σ = finite no. of input symbols
(or)
Alphabet.

δ = Transition function.

$q_0 = q_0 \in Q$, is the initial state.

$F = F \subseteq Q$, is the final state.

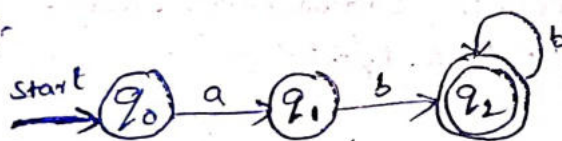
Transition Diagram:-

* It is nothing but a directed graph.

* The vertices of graph corresponding to the states of finite automata. If there is a transition from state P to state Q on input symbol 'a' then there is arc label 'a' from P to Q .

i.e., 

ex:-



Final state is always represented by double circle

here,

$$Q = \{q_0, q_1, q_2\}$$

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

$$\delta = \{(q_0, a) \rightarrow q_1, (q_1, b) \rightarrow q_2, (q_2, b) \rightarrow q_2\}$$

$$q_0 = \{q_0\}$$

$$F = \{q_2\}$$

Transition table :-

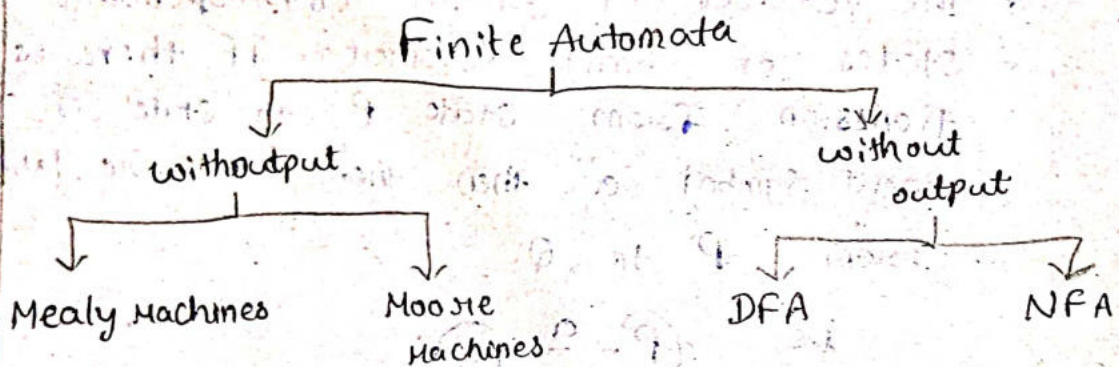
δ	a	b
$\rightarrow q_0$	q_1	Λ
q_1	Λ	q_2
(q_2)	Λ	q_2

→ There are 2 types of Finite Automata

They are

① Deterministic Finite Automata (DFA)

② Non-Deterministic Finite Automata (NFA or NFA)



Finite automata is classified into two types based on the output.

They are

① Finite Automata with output

② Finite Automata without output

Here,

finite automata with output is again divided into 2 types.

① Mealy Machines

② Moore Machines.

and finite automata without output is divided into 2 types. They are

(i) DFA

(ii) NFA

1) Deterministic Finite Automata (DFA) :-

* A DFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$

where,

Q = finite set of states

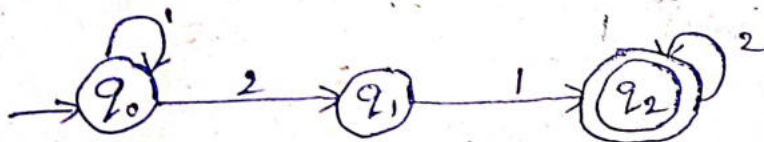
Σ = finite set input alphabet

q_0 = The initial state

$F = F \subseteq Q$, is a set of final states.

δ = It is the transition function which is mapping $Q \times \Sigma \rightarrow Q$ ~~is a state~~

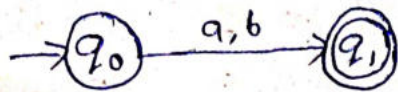
where (q, a) is the state, for each state q on input symbol a to a state q' .



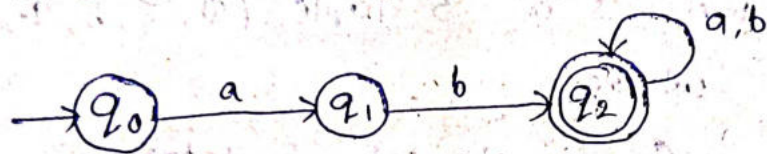
Transition table

δ	0	1	2
$\rightarrow q_0$	q_0	q_1	
q_1	q_2	$\wedge$	
q_2	$\wedge$	q_2	

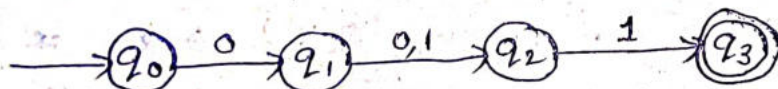
- 1) Design a DFA that accepts the strings $a+b$ over $\Sigma = \{a, b\}$



- 2) Design a DFA to accept strings of a's and b's starting with the string ab.



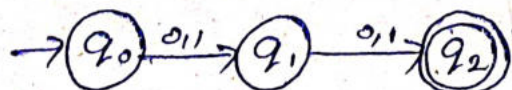
- 3) Design a DFA to accept the language $0(0+1)^1$



- 4) Design a DFA to accept the language $0(0+1)^*1$



- 5) Design a DFA for $(0+1)^1(0+1)$

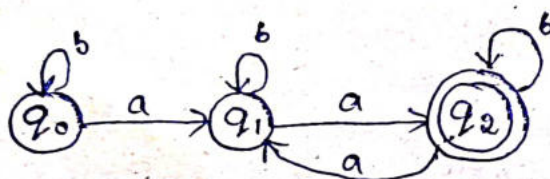


23/01/24

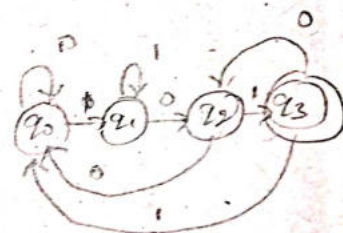
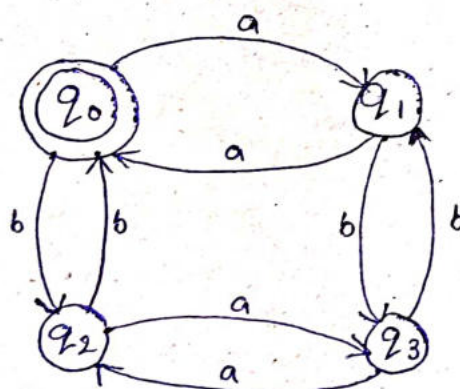
- 6) Design a DFA to accept strings of 0's and 1's over $\Sigma = \{0, 1\}$



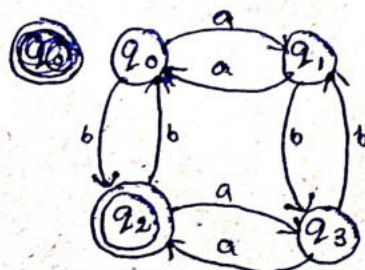
- 7) Design a DFA to accept strings of a's and b's over $\Sigma = \{a, b\}$ where it consists of even length number



- 8) Design a DFA to accept strings of a's and b's where it consists of even number of a's and b's. over $\Sigma = \{a, b\}$

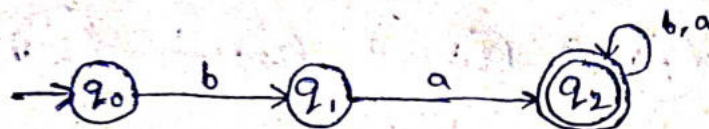


- 9) Design a DFA to accept strings of a's and b's where it consists of even no of a's and odd no of b's over $\Sigma = \{a, b\}$

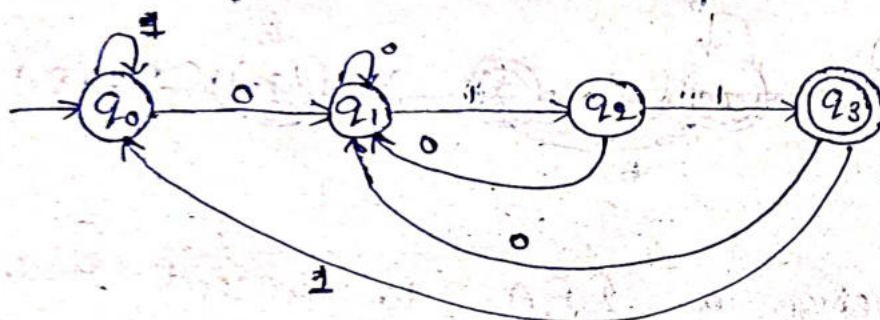


7/10/24

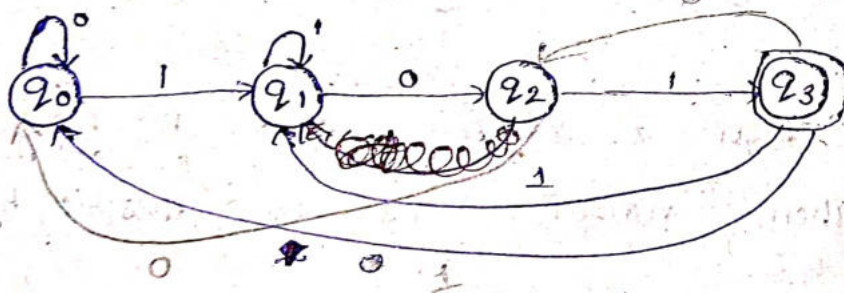
- 10) Design a DFA to accept the strings of a's and b's starting with ba.



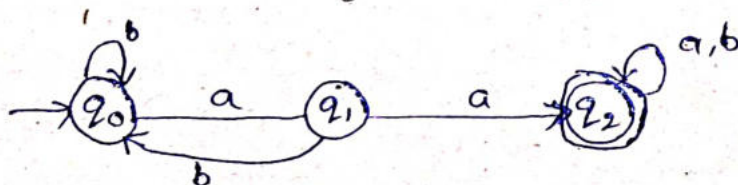
- 11) Design a DFA to accept the string of 0's and 1's ending with string 011.



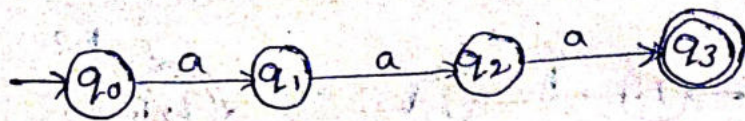
- 12) Design a DFA to accept the string of 0's and 1's ending with 101.



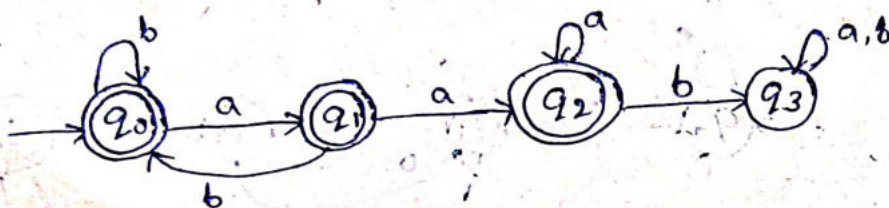
- 13) Design a DFA to accept the string of a's and b's having substring aa.



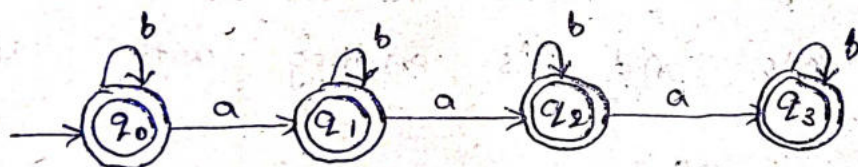
- 1) Design a DFA which consists of exactly $aaaa$.



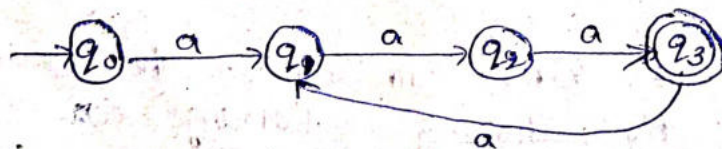
- 5) Design a DFA to accept the strings of a's and b's except those containing the substring acb .



- 16) Design a DFA to accept strings of a's and b's having not more than aaa 3 a's.



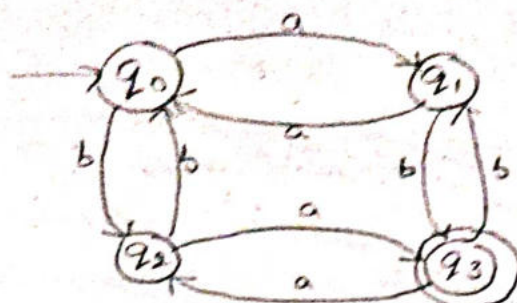
- 17) Design a DFA over $\Sigma = \{a\}$ where then no. of a's is divisible by 3.



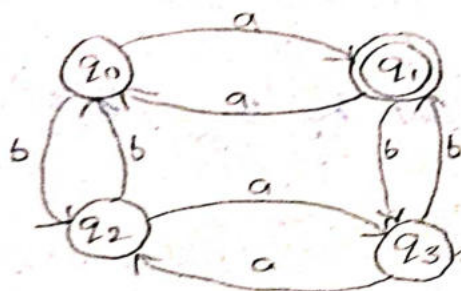
29/01/23

18)

Design a DFA to accept the strings of a's and b's where it consists of
 (i) odd a's and odd b's.



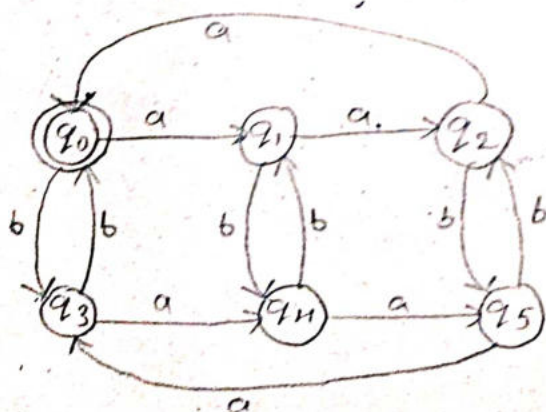
(ii) odd a's and even b's.



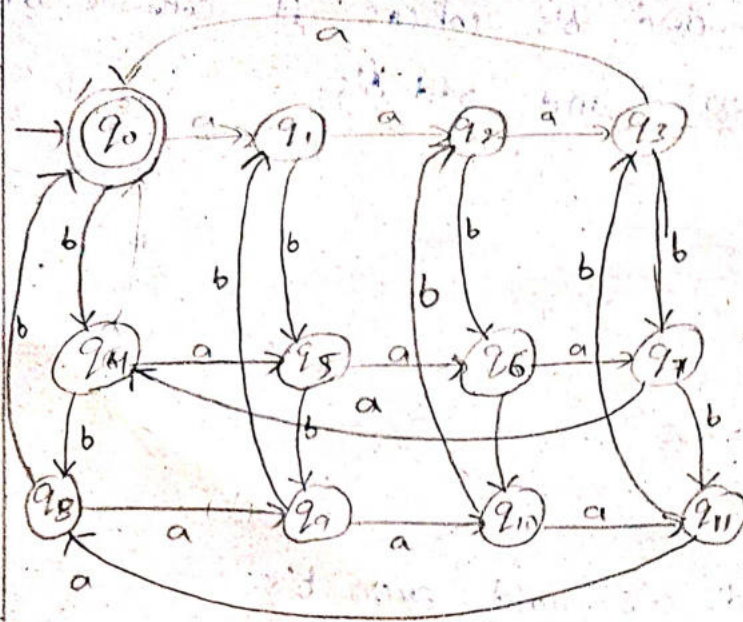
19) Design a DFA over $\Sigma = \{a, b\}$ such that no. of a's are divisible by 3 and no. of b's are divisible by 2.

Note:-

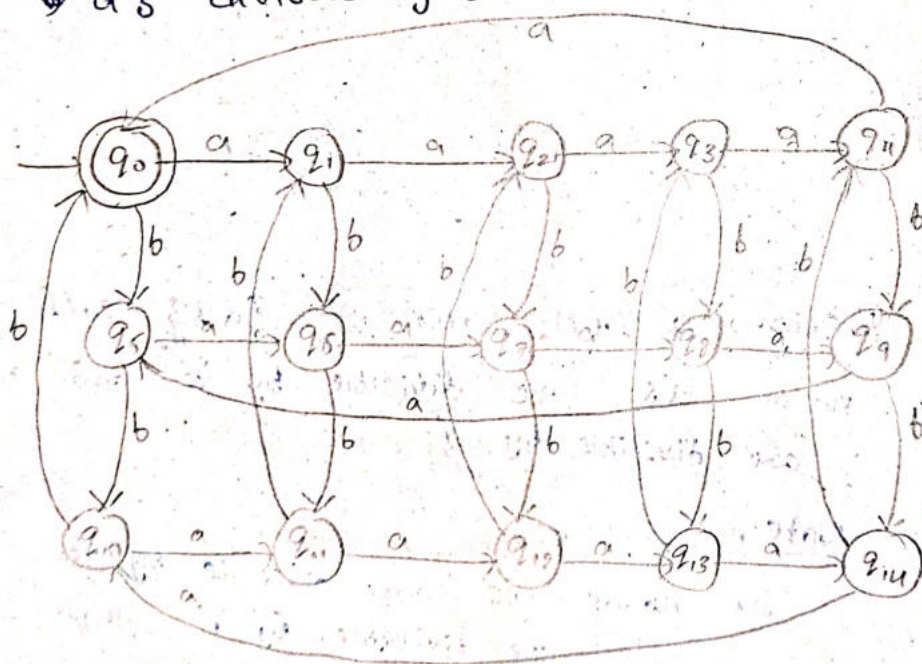
If no. of a's are divisible by 'm' and no. of b's are divisible by 'n' then there will be total of $m \times n$ states



20) a's divisible by 4 b's divisible by 3



21) a's divisible by 5 b's divisible by 3

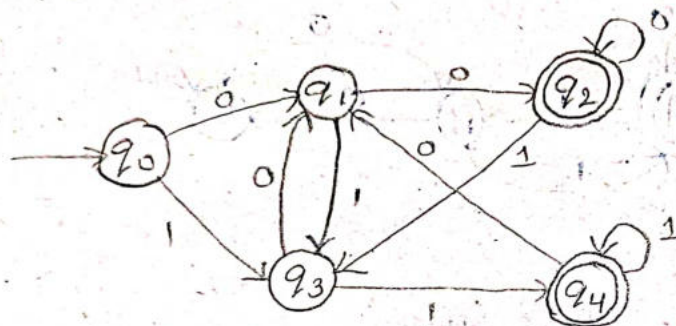


22) Design a DFA to accept all strings

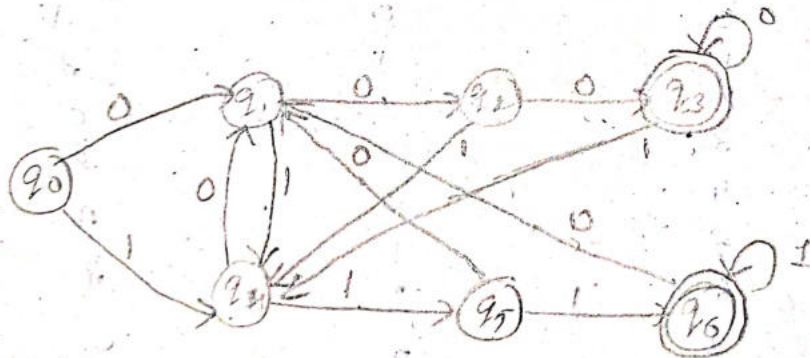
$\Sigma = \{0, 1\}$ such that the last two symbols must be same.

Note :-

If we have n consecutive 0's and n consecutive 1's at the end of string then minimum no. of states is $2n+1$.



23) 3 consecutive symbols are same.

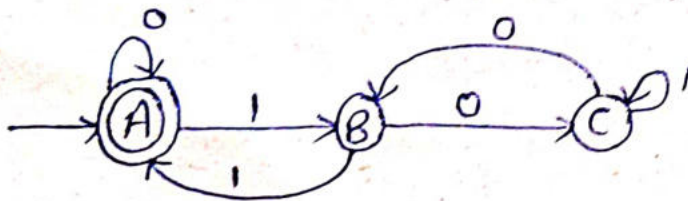


30/01/24

24) Design a DFA using minimum no. of states that recognizes the following languages.

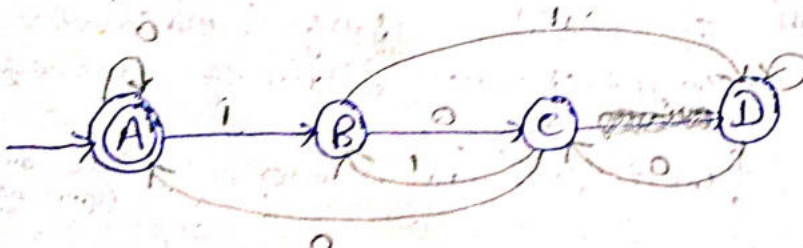
i) $L = \{w \in \{0, 1\}^* / w \text{ is interpreted as binary number (ignoring leading 0's) is divisible by } 3\}$

<u>Input</u>	<u>number</u>	<u>Remainder</u>	<u>states</u>
000	0	0	A
001	1	1	B
010	2	2	C
011	3	0	A
100	4	1	B
101	5	2	C
110	6	0	A
111	7	1	B

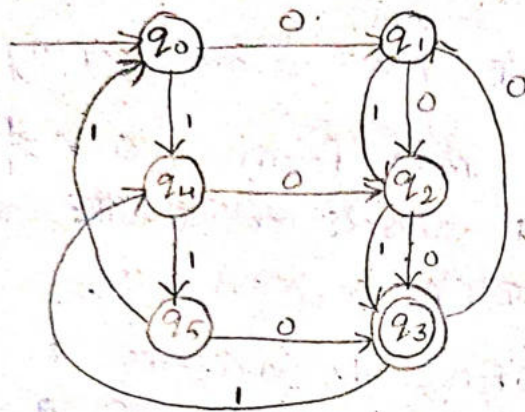


ii) Divisible by 4

<u>number</u>	<u>Remainder</u>	<u>States</u>
0	0	A
1	1	B
2	2	C
3	3	D
4	0	A
5	1	B
6	2	C
7	3	D

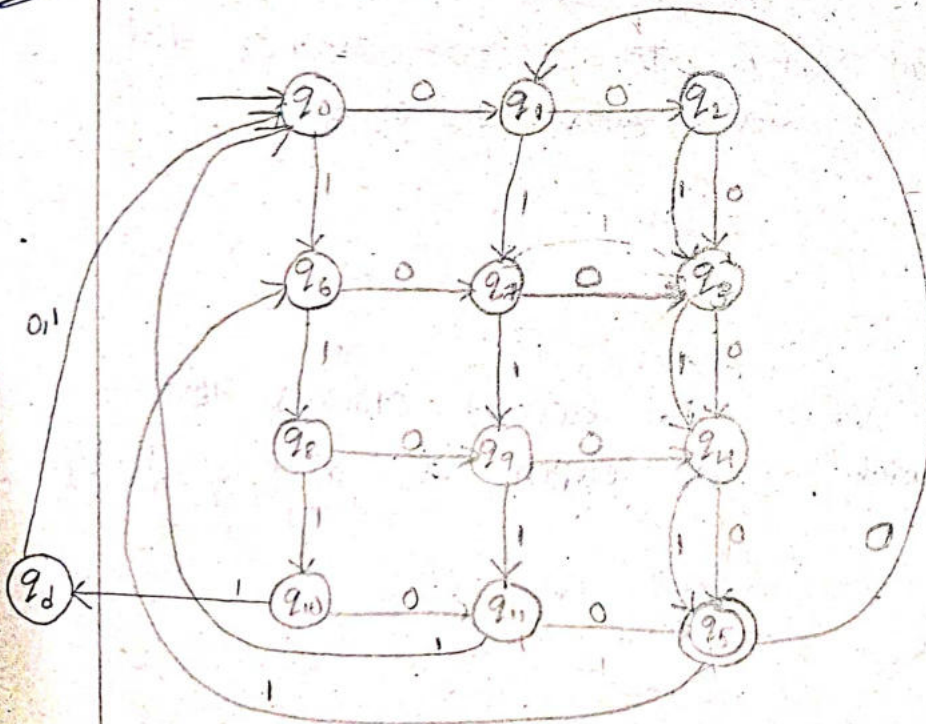


- 25) Design a DFA such that every block of 3 consecutive symbols contain at least one 0 over $\Sigma = \{0, 1\}$



000
001
010
011
100
101
110
111 X

- 31/01/23 (i) for 5 consecutive symbols with at least two 0's.



00000
00001
00010
00011
00100
00101
00110
00111
01000
01001
01010
01011
01100
01101
01110
01111
10000
10001
10010
10011
10100
10101
10110
10111
11000
11001
11010
11011
11100
11101
11110
11111

(q_d = dead state)

ii) Non-Deterministic Finite Automata :-

* NFA is a 5 tuple $(Q, \Sigma, \delta, q_0, F)$

where,

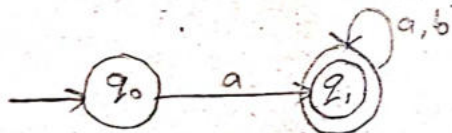
Q, Σ, q_0, F have same meaning as for DFA.

δ is a map from $Q \times \Sigma \rightarrow 2^Q$ set of all subsets of Q . The only difference b/w DFA & NFA is,

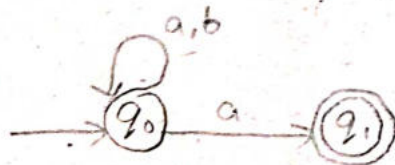
→ In DFA, the outcome is a state.
i.e. an element of Q .

→ for NFA, the outcome is a subset of Q .

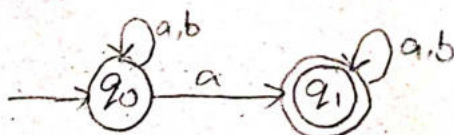
- 1) Design an NFA to accept strings that starts with 'a' over $\Sigma = \{a, b\}$



- 2) Design an NFA to accept strings that ~~starts with~~ ends with 'a' over $\Sigma = \{a, b\}$

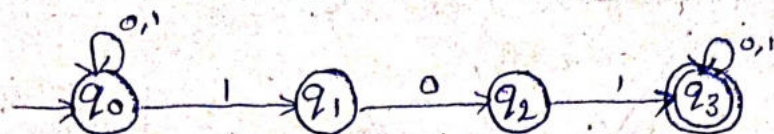


- 3) Design an NFA to accept strings that contains 'a' over $\Sigma = \{a, b\}$.

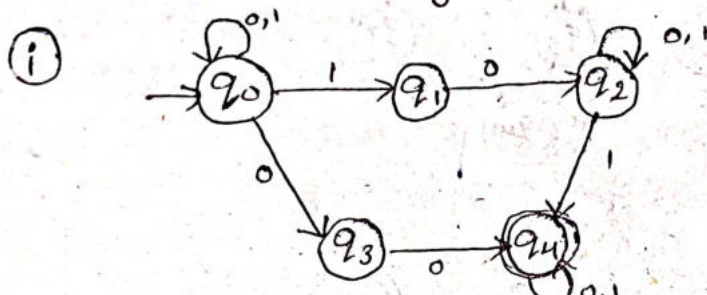


01/02/24

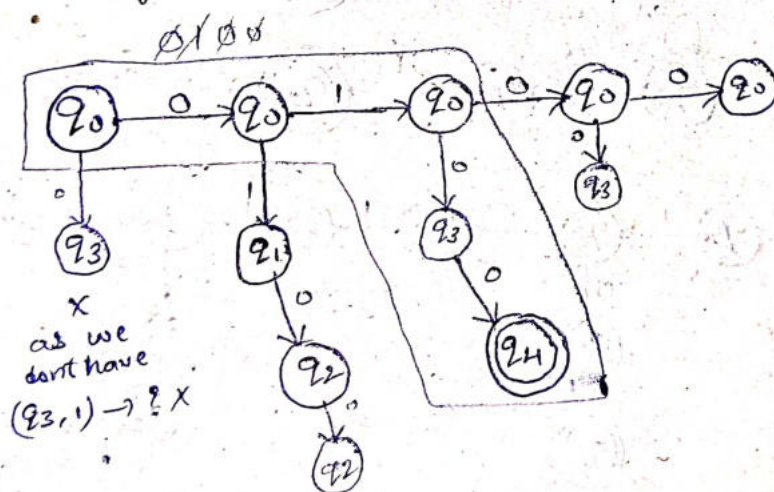
- 4) Design an NFA containing the string 101 as sub-string?



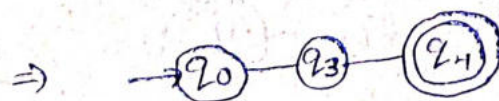
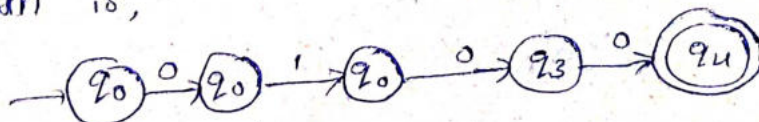
- 5) Consider an example of NFA, whose transition diagram is given as,

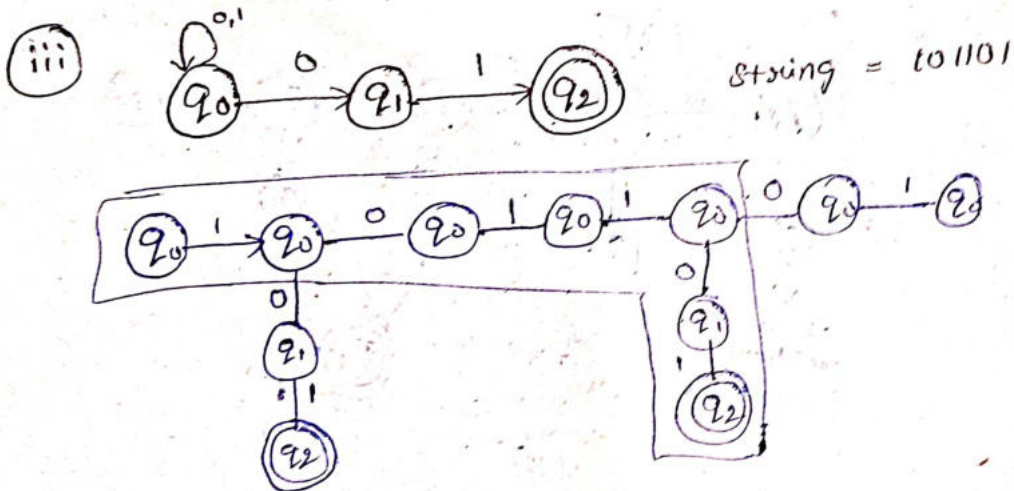
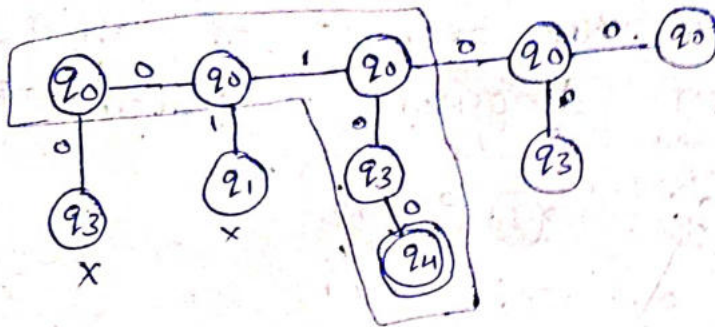
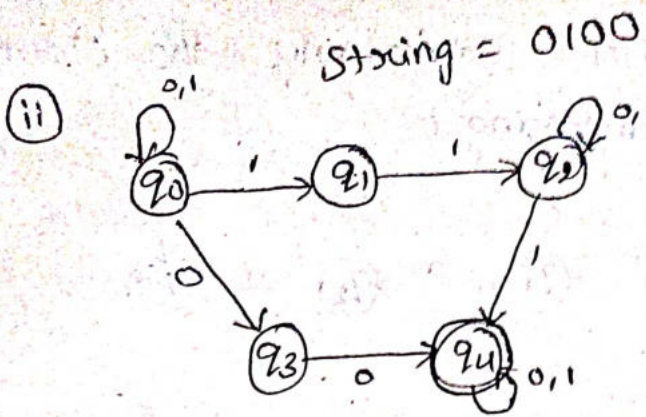


The sequence of states for the input string 0100 is.



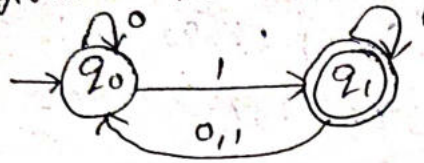
Path is,





Equivalence of deterministic & non-deterministic automata

1) Construct a DFA which is equivalent to given NFA.



A)

Transition table for DFA :-

δ	0	1
$\rightarrow q_0$	q_0	q_1
q_1	q_0	q_{01}
q_{01}	q_0	q_{01}

$$(q_{01}, 0) = (q_0, 0) \cup (q_1, 0)$$

$$= q_0 \cup q_0$$

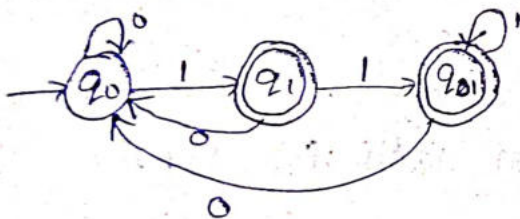
$$= q_0$$

$$(q_{01}, 1) = (q_0, 1) \cup (q_1, 1)$$

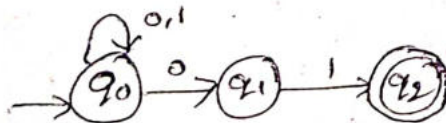
$$= q_1 \cup q_{01}$$

$$= q_{01}$$

Transition diagram for DFA



Construct a DFA ~~was~~ for given NFA



Transition table for DFA

δ	0	1
$\rightarrow q_0$	q_{01}	q_0
q_{01}	q_{01}	q_{02}
q_{02}	q_{01}	q_0

$$(q_{01}, 0) = (q_0, 0) \cup (q_1, 0)$$

$$= q_{01} \cup \Lambda$$

$$= q_{01}$$

$$(q_{01}, 1) = (q_0, 1) \cup (q_1, 1)$$

$$= q_0 \cup q_2$$

$$= q_{02}$$

$$(q_{02}, 0) = (q_0, 0) \cup (q_2, 0)$$

$$= q_{01} \cup \Lambda$$

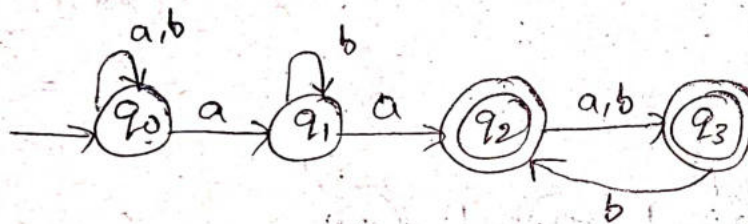
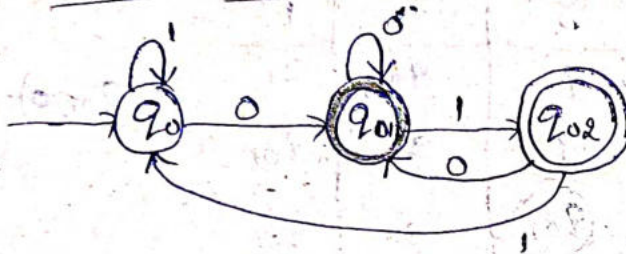
$$= q_{01}$$

$$(q_{02}, 1) = (q_0, 1) \cup (q_2, 1)$$

$$= q_0 \cup \Lambda$$

$$= q_0$$

Transition diagram for DFA:



Transition table for DFA

δ	a	b
$\rightarrow q_0$	q_{01}	q_0
q_{01}	q_{012}	q_{01}
q_{012}	q_{0123}	q_{013}
q_{0123}	q_{0123}	q_{0123}
q_{013}	q_{012}	q_{012}

$$(q_{01}, a) = (q_0, a) \cup (q_1, a)$$

$$= q_{01} \cup q_2$$

$$= q_{012}$$

$$(q_{01}, b) = (q_0, b) \cup (q_1, b)$$

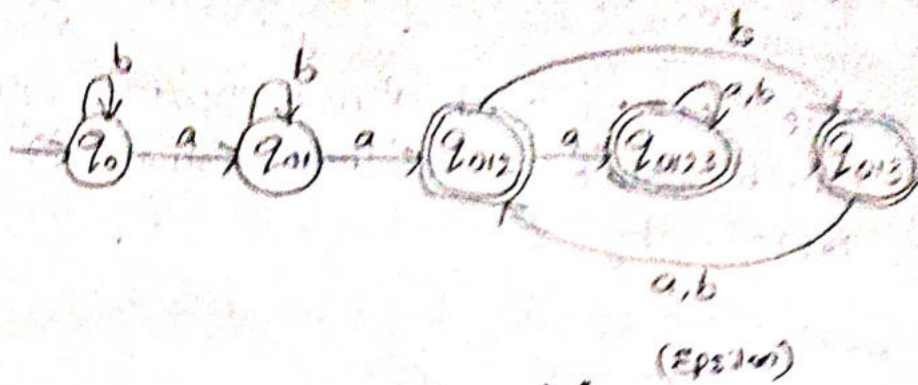
$$= q_0 \cup q_1$$

$$= q_{01}$$

$$(q_{012}, a) = (q_0, a) \cup (q_1, a) \cup (q_2, a)$$

$$= q_{01} \cup q_2 \cup q_3$$

$$= q_{0123}$$



Finite Automata with ϵ transitions.

* If a finite Automata is modified to permit transition without input symbol then we get a NFA with ϵ -transition.

* NFA with ϵ -transition is also a 5-tuple form:

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

where,

$$\delta = Q \times (\Sigma \cup \epsilon) \rightarrow 2^Q$$

Acceptance of String w by NFA with ϵ -transition.

* A String " w " in Σ^* will be accepted by NFA with ϵ -transition, if there exists atleast one path corresponding " w " which starts in an initial state and ends in one of the final states. Since this path may be formed by ϵ -transition as well as non- ϵ -transition.

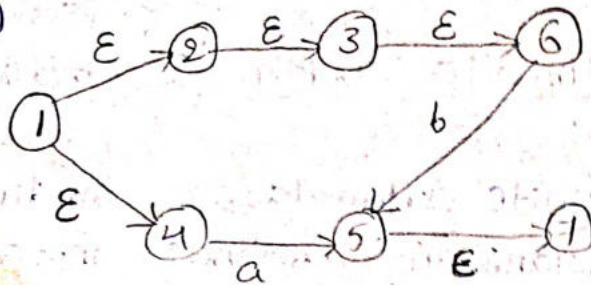
* To find out whether " w " is accepted or not by the NFA with ϵ moves we require to define a function ϵ .

i.e., ϵ -closure(q).

ϵ -closure :-

* The set of all those states of automata which can be reached from 'Q' on a path labeled by ϵ .

ex:- ①



$$\epsilon\text{-closure}(1) = \{1, 2, 3, 4, 6\}$$

$$\epsilon\text{-closure}(2) = \{2, 3, 6\}$$

$$\epsilon\text{-closure}(3) = \{3, 6\}$$

$$\epsilon\text{-closure}(4) = \{4\}$$

$$\epsilon\text{-closure}(5) = \{5, 7\}$$

$$\epsilon\text{-closure}(6) = \{6\}$$

$$\epsilon\text{-closure}(7) = \{7\}$$

Eliminating ϵ -transitions :-

i) Short Cut method :-

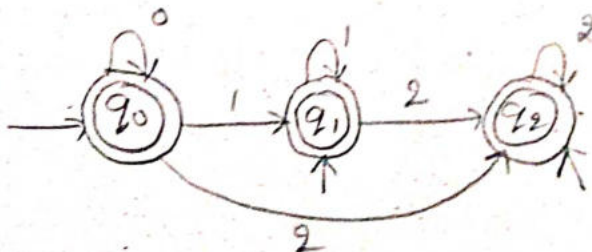
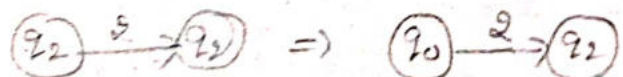
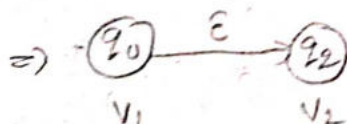
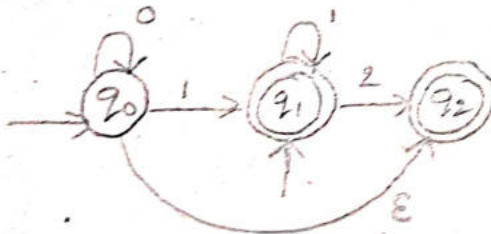
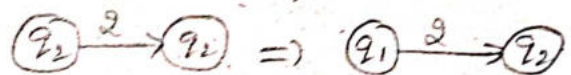
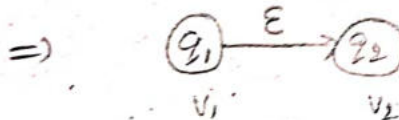
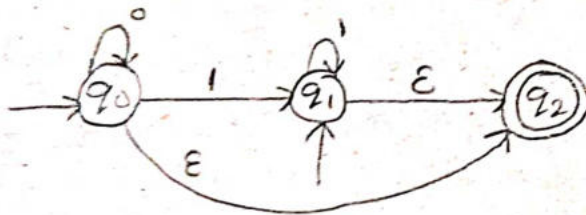
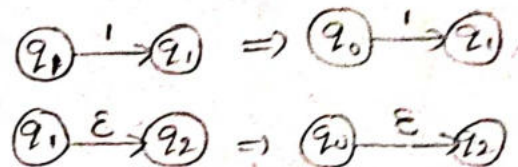
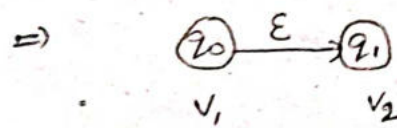
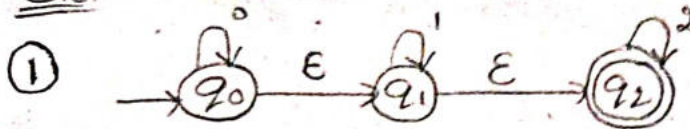
Suppose we want to replace a null move from vertex v_1 to v_2 then proceed as follows:-

- ① find all edges starting from v_2 .
- ② Duplicate these edges starting from v_1 without changing the edge labels (inputs).

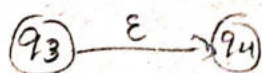
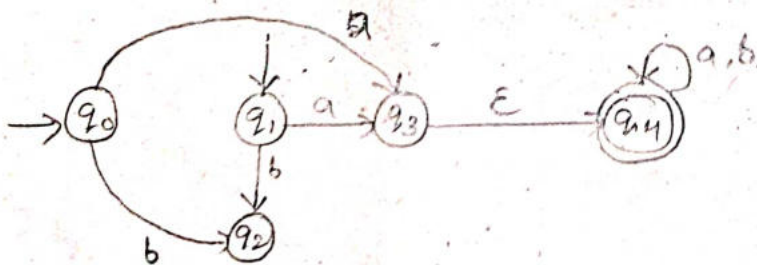
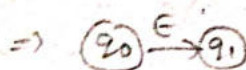
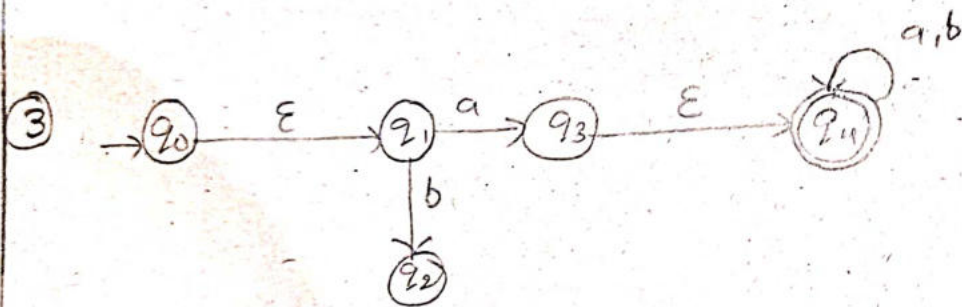
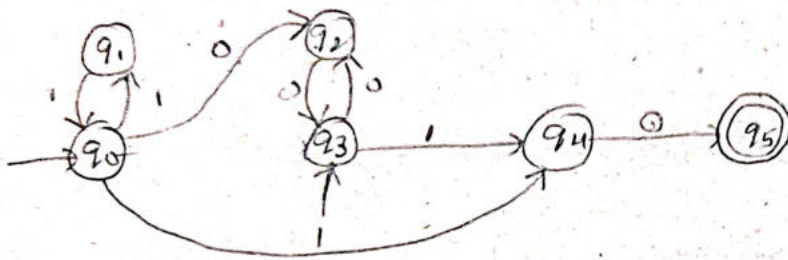
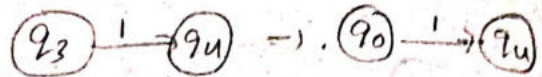
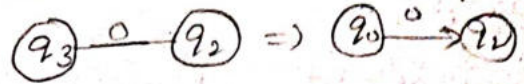
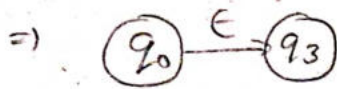
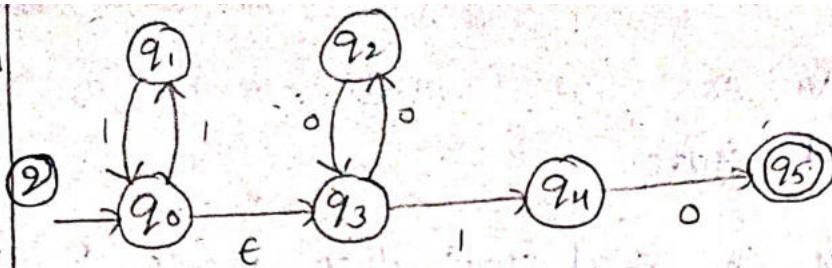
(iii) IF V_1 is initial state & make V_2 also initial state.

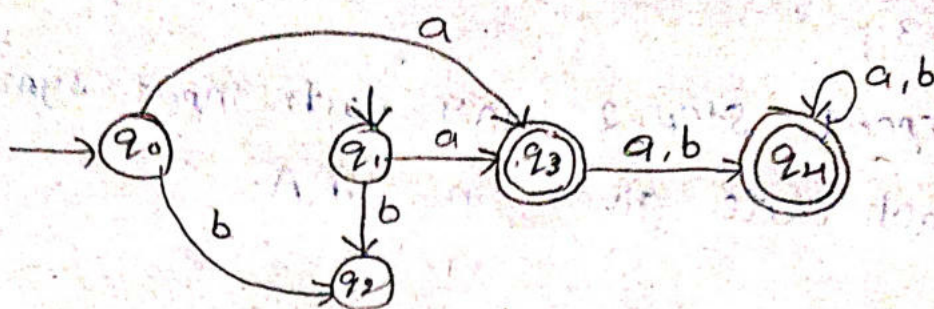
(iv) IF V_2 is final state then make V_1 also final state.

ex:-



02/24





Conversion of NFA with ϵ moves
to NFA without ϵ moves [ϵ -closure method]

* In this method, we remove ϵ -transitions from the given NFA & obtain NFA without ϵ -moves.

Algorithm :-

Step-1 :-

find out all ϵ -transitions from each state in Q that will be called as ϵ -closure ϵ -closure(q_i) or $\epsilon^*(q_i)$ where $q_i \in Q$.

Step-2 :-

δ' transitions can be obtained as

$$\textcircled{a} \quad \delta'(q, \epsilon) = \epsilon\text{-closure}(q)$$

$$\textcircled{b} \quad \delta'(q, a) = \epsilon\text{-closure}[\delta(\epsilon\text{-closure}(q, A))]$$

Step-3:-

repeat step-2 for each input symbol and each state given in NFA.

Step-4:-

finally the resultant states ^{added} NFA without ϵ -moves is obtained.

5/02/24

1) Convert the given Epsilon NFA into DFA.



Let M be a given epsilon NFA like

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

where,

$$Q = \{q_0, q_1, q_2\}$$

$$\Sigma = \{0, 1, 2\}$$

$$\delta = \{ (q_0, 0) \rightarrow q_0, (q_0, \epsilon) \rightarrow q_1,$$

$$(q_1, 1) \rightarrow q_1, (q_1, \epsilon) \rightarrow q_2,$$

$$(q_2, 2) \rightarrow q_2 \}$$

$$q_0 = \{q_0\}$$

$$F = \{q_2\}$$

Now,

ϵ -closure for all the states in the given NFA are

$$\epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\}$$

$$\epsilon\text{-closure}(q_1) = \{q_1, q_2\}$$

$$\epsilon\text{-closure}(q_2) = \{q_2\}$$

Compute δ' transitions as

$$\begin{aligned}\delta'(\epsilon\text{-closure}(q_0, 0)) &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0), 0)) \\ &= \epsilon\text{-closure}(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)) \\ &= \epsilon\text{-closure}(q_0 \cup \Lambda \cup \Lambda) \\ &= \epsilon\text{-closure}(q_0) \\ &= \{q_0, q_1, q_2\}\end{aligned}$$

$$\begin{aligned}\delta'(\epsilon\text{-closure}(q_0, 1)) &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0), 1)) \\ &= \epsilon\text{-closure}(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)) \\ &= \epsilon\text{-closure}(q_1 \cup \Lambda \cup \Lambda) \\ &= \epsilon\text{-closure}(q_1) \\ &= \{q_1, q_2\}\end{aligned}$$

$$\begin{aligned}\delta'(\epsilon\text{-closure}(q_0, 2)) &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0), 2)) \\ &= \epsilon\text{-closure}(\delta(q_0, 2) \cup \delta(q_1, 2) \cup \delta(q_2, 2)) \\ &= \epsilon\text{-closure}(q_2 \cup \Lambda \cup \Lambda) \\ &= \epsilon\text{-closure}(q_2) \\ &= \{q_2\}\end{aligned}$$

$$\delta'(\epsilon\text{-closure}(q_1, 0)) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1, 0)))$$

$$= \epsilon\text{-closure}(\delta(q_1, q_2), 0)$$

$$= \epsilon\text{-closure}(\delta(q_1, 0) \cup \delta(q_2, 0))$$

$$= \epsilon\text{-closure}(\Lambda \cup \Lambda)$$

$$= \Lambda$$

$$\delta''(\epsilon\text{-closure}(q_1, 1)) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1, 1)))$$

$$= \epsilon\text{-closure}(\delta(q_1, q_2), 1)$$

$$= \epsilon\text{-closure}(\delta(q_1, 1) \cup \delta(q_2, 1))$$

$$= \epsilon\text{-closure}(q_1 \cup \Lambda)$$

$$= \epsilon\text{-closure}(q_1)$$

$$= \{q_1, q_2\}$$

$$\delta'(\epsilon\text{-closure}(q_1, 2)) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1, 2)))$$

$$= \epsilon\text{-closure}(\delta(q_1, 2) \cup \delta(q_2, 2))$$

$$= \epsilon\text{-closure}(\Lambda \cup q_2)$$

$$= \epsilon\text{-closure}(q_2)$$

$$= \{q_2\}$$

$$\delta'(\epsilon\text{-closure}(q_2, 0)) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2, 0)))$$

$$= \epsilon\text{-closure}(\delta(q_2, 0))$$

$$= \epsilon\text{-closure}(\Lambda)$$

$$= \Lambda$$

$$\delta'(\epsilon\text{-closure}(q_2, 1)) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2, 1)))$$

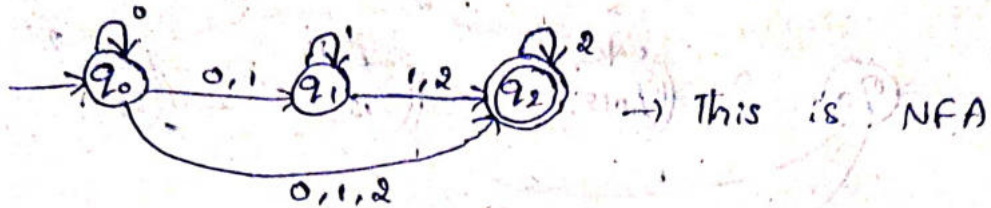
$$= \epsilon\text{-closure}(\delta(q_2, 1))$$

$$= \epsilon\text{-closure}$$

$$= \epsilon\text{-closure}(\Lambda)$$

$$= \Lambda$$

$$\begin{aligned}
 \delta'(\epsilon\text{-closure}(q_1, 2)) &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1, 2), 2)) \\
 &= \epsilon\text{-closure}(\delta(q_2, 2)) \\
 &= \epsilon\text{-closure}(q_2) \\
 &= \{q_2\}
 \end{aligned}$$



Now Convert NFA to DFA.

δ	0	1	2
→ q_0	q_{012}	q_{12}	q_2
q_{012}	q_{012}	q_{12}	q_2
q_{12}	ϵ	q_{12}	q_2
q_2	ϵ	ϵ	q_2

$$\begin{aligned}
 (q_{012}, 0) &= (q_0, 0) \cup (q_1, 0) \cup (q_2, 0) \\
 &= q_0 \cup q_1 \cup q_2 \\
 &= q_{012}
 \end{aligned}$$

$$\begin{aligned}
 (q_{012}, 1) &= (q_0, 1) \cup (q_1, 1) \cup (q_2, 1) \\
 &= \epsilon \cup q_{12} \cup \epsilon \\
 &= q_{12}
 \end{aligned}$$

$$\begin{aligned}
 (q_{12}, 0) &= (q_1, 0) \cup (q_2, 0) \\
 &= \epsilon \cup \epsilon \\
 &= \epsilon
 \end{aligned}$$

$$\begin{aligned}
 (q_{12}, 1) &= (q_1, 1) \cup (q_2, 1) \\
 &= q_{12} \cup \epsilon \\
 &= q_{12}
 \end{aligned}$$

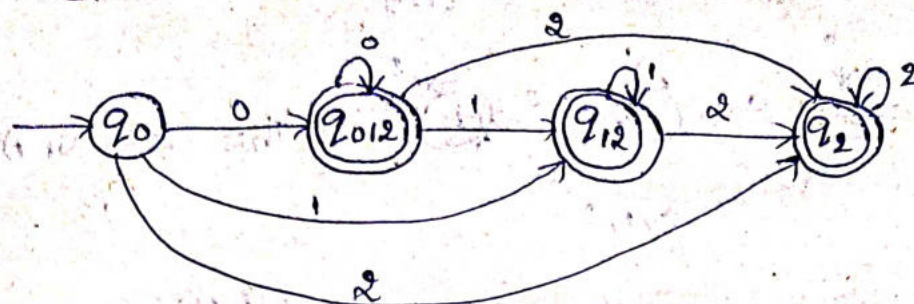
$$\begin{aligned}
 (q_{12}, 2) &= \epsilon \\
 &= (q_1, 2) \cup (q_2, 2) \\
 &= \epsilon \cup q_2 \\
 &= q_2
 \end{aligned}$$

q_2

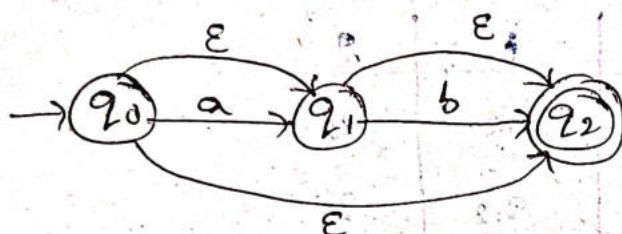
$$(q_2, 0) = \Lambda$$

$$(q_2, 1) = \Lambda$$

$$(q_2, 2) = q_2$$



2)



Let M be the given epsilon NFA then,

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

where,

$$Q = \{q_0, q_1, q_2\}$$

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

$$\delta = \left\{ \begin{array}{l} (q_0, a) \rightarrow q_1, (q_0, \epsilon) \rightarrow q_1, \\ (q_0, \epsilon) \rightarrow q_2, (q_1, b) \rightarrow q_2, \\ (q_1, \epsilon) \rightarrow q_2 \end{array} \right\}$$

Now,

ϵ -closure for all states in the given NFA are

$$\epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\}$$

$$\epsilon\text{-closure}(q_1) = \{q_1, q_2\}$$

$$\epsilon\text{-closure}(q_2) = \{q_2\}$$

Compute δ' transitions are

$$\begin{aligned}
 \delta'(\epsilon\text{-closure}(q_0, a)) &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0), a)) \\
 &= \epsilon\text{-closure}(\delta(q_0, q_1, q_2), a) \\
 &= \epsilon\text{-closure}(\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a)) \\
 &= \epsilon\text{-closure}(q_1 \cup \Lambda \cup \Lambda) \\
 &= \{q_1, q_2\}
 \end{aligned}$$

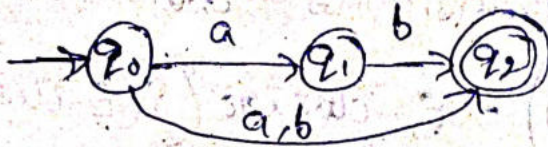
$$\begin{aligned}
 \delta'(\epsilon\text{-closure}(q_0, b)) &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0), b)) \\
 &= \epsilon\text{-closure}(\delta(q_0, q_1, q_2), b) \\
 &= \epsilon\text{-closure}(\delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b)) \\
 &= \epsilon\text{-closure}(\Lambda \cup q_2 \cup \Lambda) \\
 &= \{q_2\}
 \end{aligned}$$

$$\begin{aligned}
 \delta'(\epsilon\text{-closure}(q_1, a)) &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1), a)) \\
 &= \epsilon\text{-closure}(\delta(q_1, q_2), a) \\
 &= \epsilon\text{-closure}(\delta(q_1, a) \cup \delta(q_2, a)) \\
 &= \epsilon\text{-closure}(\Lambda \cup \Lambda) \\
 &= \epsilon\text{-closure}(\Lambda) \\
 &= \Lambda
 \end{aligned}$$

$$\begin{aligned}
 \delta'(\epsilon\text{-closure}(q_1, b)) &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1), b)) \\
 &= \epsilon\text{-closure}(\delta(q_1, b) \cup \delta(q_2, b)) \\
 &= \epsilon\text{-closure}(q_2 \cup \Lambda) \\
 &= \{q_2\}
 \end{aligned}$$

$$\begin{aligned}
 \delta'(\epsilon\text{-closure}(q_2, a)) &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2), a)) \\
 &= \epsilon\text{-closure}(\delta(q_2, a)) \\
 &= \Lambda
 \end{aligned}$$

$$\delta'(\epsilon\text{-closure}(q_2, b)) = \Lambda$$



Now Convert NFA to DFA

δ	a	b
$\rightarrow q_0$	q_{12}	q_2
q_{12}	Λ	q_2
q_2	Λ	Λ

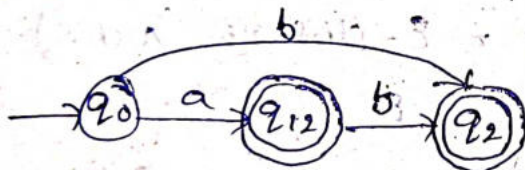
$$(q_{12}, a) = (q_1, a) \cup (q_2, a) \\ = \Lambda \cup \Lambda$$

$$(q_{12}, b) = (q_1, b) \cup (q_2, b)$$

$$= q_2 \cup \Lambda$$

$$= q_2$$

$$(q_2, a) = \Lambda, \quad (q_2, b) = \Lambda$$



06/02/21

Equivalence of finite state machines

* Two finite Automata are equivalent, if they accept the same set of strings over 'L' otherwise they are not equivalent.

Steps:-

Step-1:-

1) We can check the equivalence of two finite state machines by using comparison table.

07/02/21

Comparison table:-

- * It is similar to transition table & it contains $n+1$ columns. Here, 1st column represents state and n columns represents input symbols.
- * Entries are pairs of states, here pairs of states are both are final states (or) both are non-final states.

Construction of comparison table:-

Comparison table starts with initial states of M and M' in the first column and consider the entry as (q_0, q'_0) where,

where,

$$\delta(q_0, a) = q_0$$

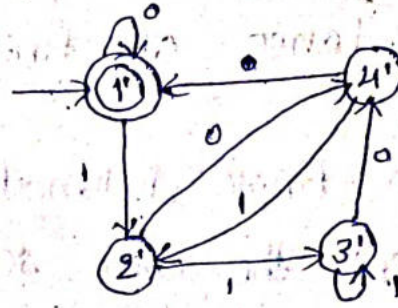
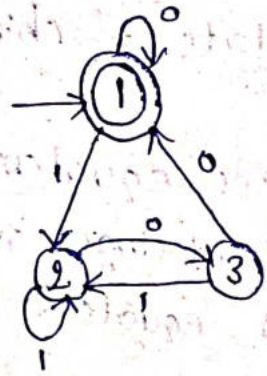
$$\delta(q'_0, a) = q'_0$$

Step-2:-

Repeat step-1 for the input symbol on every pair of states of M & M' until no new pair appears.

Step-3:-

If we get any pair (q_0, q'_0) such that q_0 is the final state & q'_0 is the non-final state (or) vice versa, then we terminate the process & conclude that both M & M' are not equivalent.



Pair of States	0	1
(1, 1')	(1, 1')	(2, 2')
(2, 2')	(3, 4')	(2, 3')
(3, 4')	(1, 1')	(2, 2')
(2, 3')	(3, 4')	(2, 3')

∴ Given two finite automata are equivalent.

2)



Pair of States	0	1
(q0, p0)	(q0, p0)	(q1, p1)
(q1, p1)	(q1, p1)	(q0, p2)
(q0, p2)	(q0, p2)	(q1, p1)

∴ Given two finite automata are equivalent.

DFA minimisation (or) Optimisation of finite Automata

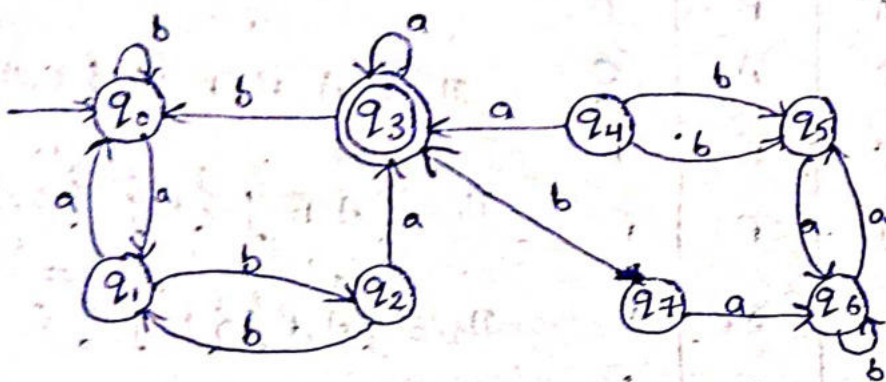
For minimisation, there are 3 methods

① Minimisation of DFA using Equivalence theorem.

② Minimisation of DFA using Myhill Nerd. method

or Table filling method.

1) Construct the state minimising the finite automata for the following transition diagram.



δ	a	b
q0	q1	q0
q1	q0	q2
q2	q3	q1
q3	q3	q0
q4	q3	q5
q5	q6	q4
q6	q5	q6
q7	q6	q3

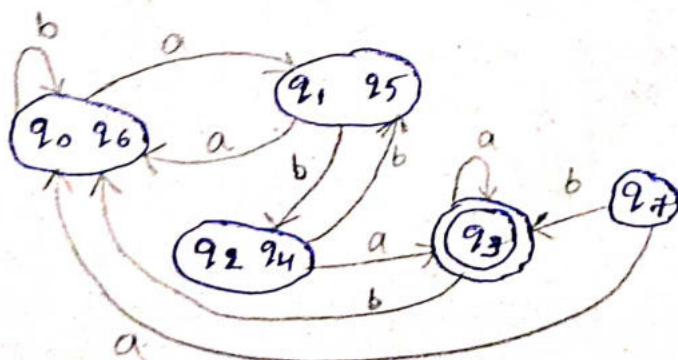
$$\Pi_0 = \{q_0, q_1, q_2, q_4, q_5, q_6, q_7\} \{q_3\}$$

$$\Pi_1 = \{q_0, q_1, q_5, q_6\} \{q_2, q_4\} \{q_7\} \{q_3\}$$

$$\Pi_2 = \{q_0, q_6\} \{q_1, q_5\} \{q_2, q_4\} \{q_7\} \{q_3\}$$

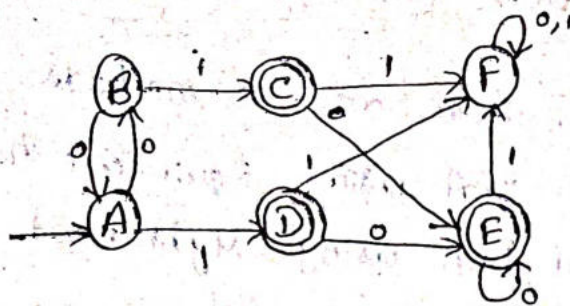
$$\Pi_3 = \{q_0, q_6\} \{q_1, q_5\} \{q_2, q_4\} \{q_7\} \{q_3\}$$

A B C D E



08/02/24

2)



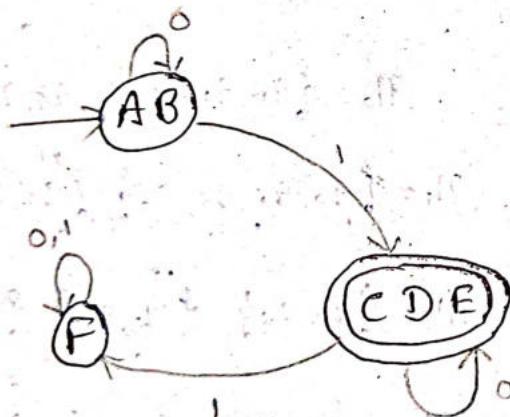
Transition Table

δ	0	1
$\rightarrow A$	B	D
B	A	C
C	E	F
D	E	F
E	E	F
F	F	F

$$\pi_0 = \{A, B, F\}, \{C, D, E\}$$

$$\pi_1 = \{A, B\}, \{F\}, \{C, D, E\}$$

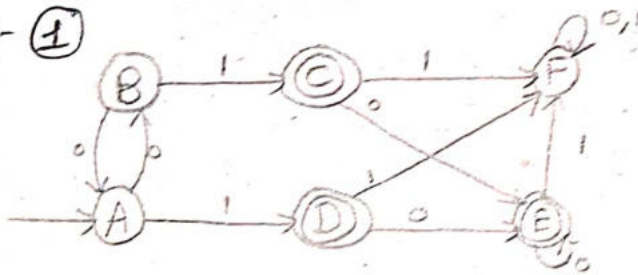
$$\pi_2 = \{A, B\}, \{F\}, \{C, D, E\}$$



Myhill nexrad method (or) Table-filling Method

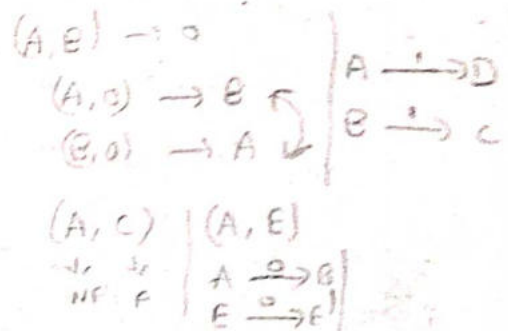
- 1) Construct a lower triangular matrix to represent relationship b/w one state with another state for given DFA, for all states (P, Q)
- 2) Mark all pairs (P, Q) with 'x' where $P \in F$, and $Q \notin F$
- 3) If there are unmarked pairs (P, Q) such that $\delta(P, a)$ and $\delta(Q, a)$ is marked then mark the pair (P, Q) where 'a' is input symbol and repeat this step until no more markings can be made.
- 4) Combine all the unmarked pairs and then a single state in the minimised DFA.

ex:- ①

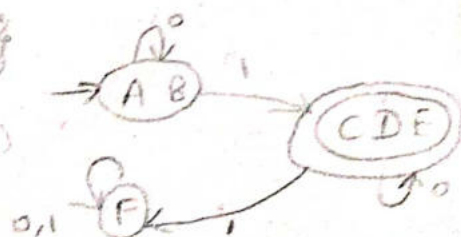


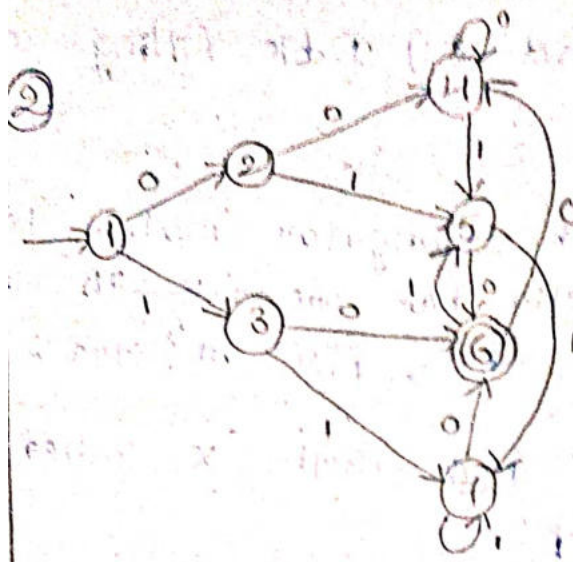
	0	1
A	B	D
B	A	C
C	E	F
D	E	F
E	E	F
F	F	F

B					
C	X	X			
D	X	X			
E	X	X			
F	X	X	X	X	X
	A	B	C	D	E



$\{A, B\}, \{C, D, E\}, \{D, E\}, \{F\}$
 $\Rightarrow \{A, B\}, \{C, D, E\}, \{F\}$





δ	Q	
→ 1	2	3
2	4	5
3	6	7
4	4	5
5	6	7
⑥	4	4, 5
7	6	7

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

(1, 2)

$$\left. \begin{array}{l} 1 \xrightarrow{0} 2 \\ 2 \xrightarrow{0} 4 \end{array} \right\} \begin{array}{l} 1 \xrightarrow{1} 3 \\ 2 \xrightarrow{1} 5 \end{array}$$

(1, 3)

$$\begin{array}{l} 1 \xrightarrow{0} 2 \\ 3 \xrightarrow{0} 6 \end{array}$$

(1, 4)

$$\begin{array}{l} 1 \xrightarrow{0} 2 \\ 4 \xrightarrow{0} 4 \end{array}$$

(1, 5)

$$\begin{array}{l} 1 \xrightarrow{1} 3 \\ 4 \xrightarrow{1} 5 \end{array}$$

(1, 7)

$$\begin{array}{l} 1 \xrightarrow{0} 2 \\ 5 \xrightarrow{0} 6 \end{array}$$

(2, 2)

$$\begin{array}{l} 1 \xrightarrow{0} 2 \\ 7 \xrightarrow{0} 6 \end{array}$$

(2, 3)

$$\begin{array}{l} 2 \xrightarrow{0} 4 \\ 3 \xrightarrow{0} 6 \end{array}$$

(2, 4)

$$\begin{array}{l} 2 \xrightarrow{0} 4 \\ 4 \xrightarrow{0} 4 \end{array}$$

$$\begin{array}{c|c|c|c}
 (2,5) & (2,7) & (3,4) & (3,5) \\
 \hline
 2 \xrightarrow{0} 4 & 2 \xrightarrow{0} 4 & 3 \xrightarrow{0} 6 & 3 \xrightarrow{0} 6 \\
 5 \xrightarrow{1} 6 & 7 \xrightarrow{0} 6 & 2 \xrightarrow{0} 4 & 5 \xrightarrow{0} 6
 \end{array}$$

$$\begin{array}{l}
 3 \xrightarrow{1} 7 \\
 5 \xrightarrow{1} 7
 \end{array}$$

(3,2)

$$\begin{array}{c|c|c}
 3 \xrightarrow{0} 6 & 3 \xrightarrow{1} 7 & (4,5) \\
 7 \xrightarrow{1} 6 & 7 \xrightarrow{1} 7 & 4 \xrightarrow{0} 4 \\
 & & 5 \xrightarrow{1} 6
 \end{array}$$

(4,7)

$$\begin{array}{c|c|c}
 4 \xrightarrow{0} 4 & 5 \xrightarrow{0} 6 & 5 \xrightarrow{1} 7 \\
 7 \xrightarrow{0} 6 & 7 \xrightarrow{0} 6 & 7 \xrightarrow{1} 7
 \end{array}$$

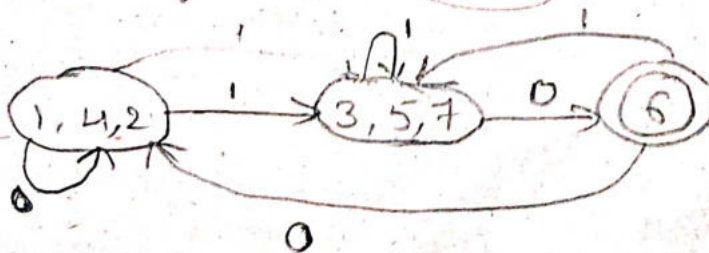
(1,2)

$$\begin{array}{c|c|c|c}
 1 \xrightarrow{0} 2 & 1 \xrightarrow{1} 3 & 1 \xrightarrow{0} 2 & 4 \xrightarrow{0} 5 \\
 2 \xrightarrow{1} 4 & 3 \xrightarrow{1} 5 & 4 \xrightarrow{0} 4 & 1 \xrightarrow{1} 3
 \end{array}$$

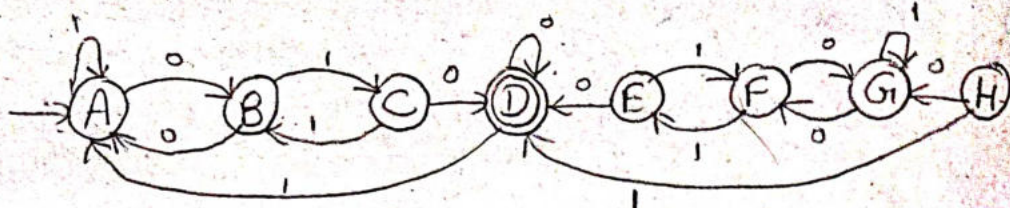
(2,4)

$$\begin{array}{c|c|c}
 2 \xrightarrow{0} 4 & 2 \xrightarrow{1} 5 & (3,5) \\
 4 \xrightarrow{0} 4 & 4 \xrightarrow{1} 5 & 3 \xrightarrow{0} 6 \\
 & & 5 \xrightarrow{0} 6
 \end{array}$$

(1,4,2) {3,5,7}, {6}



③



δ	0	1
$\rightarrow A$	B	A
B	A	C
C	D	B
D	D	A
E	D	F
F	G	E
G	F	G
H	G	D

B	X						
C	X	X					
D	X	X	X				
E	X	X			X		
F	X		X	X	X		
G		X	X	X	X	X	
H	X	X	X	X	X	X	X
	A	B	C	D	E	F	G

(A,B)	(A,C)	(A,E)	(A,F)	(A,G)
$A \Rightarrow B$ $A \vdash A$	$A \Rightarrow B$ $A \vdash B$	$A \Rightarrow B$ $A \vdash B$	$A \Rightarrow B$ $A \vdash B$	$A \Rightarrow B$ $A \vdash B$
$B \Rightarrow A$ $B \vdash A$	$C \Rightarrow D$ $E \Rightarrow D$	$F \Rightarrow G$ $F \vdash E$	$G \Rightarrow F$ $G \vdash F$	$G \Rightarrow F$ $G \vdash F$

(A,H)	(B,C)	(B,E)	(B,F)	(B,G)
$A \Rightarrow B$ $A \vdash A$	$B \Rightarrow A$ $B \vdash A$	$B \Rightarrow A$ $B \vdash A$	$B \Rightarrow A$ $B \vdash A$	$B \Rightarrow A$ $B \vdash A$
$H \Rightarrow G$ $H \vdash D$	$C \Rightarrow D$ $E \Rightarrow D$	$F \Rightarrow G$ $F \vdash F$		

(B,G)	(B,H)	(C,E)	(C,F)
$B \Rightarrow A$ $B \vdash C$	$B \Rightarrow A$ $B \vdash C$	$C \Rightarrow D$ $E \vdash B$	$C \Rightarrow D$ $C \vdash B$
$G \Rightarrow F$ $G \vdash G$	$H \Rightarrow G$ $H \vdash D$	$E \Rightarrow D$ $E \vdash F$	$F \Rightarrow G$ $F \vdash E$

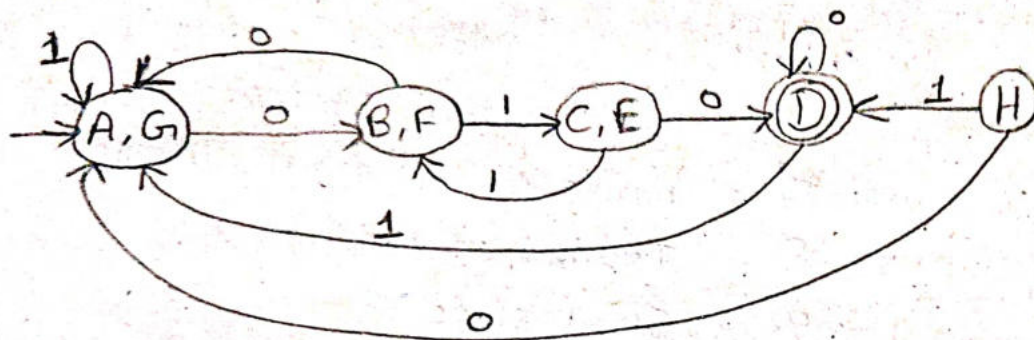
(C,G)	(C,H)	(E,F)	(E,G)	(E,H)
$C \Rightarrow D$ $C \vdash D$	$C \Rightarrow D$ $C \vdash D$	$E \Rightarrow D$ $E \vdash D$	$E \Rightarrow D$ $E \vdash D$	$E \Rightarrow D$ $E \vdash D$
$G \Rightarrow F$ $H \Rightarrow G$	$H \Rightarrow G$ $H \vdash G$	$F \Rightarrow G$ $G \Rightarrow F$	$G \Rightarrow F$ $H \Rightarrow G$	

(F,G)	(F,H)	(G,H)
$F \Rightarrow G$ $F \vdash E$	$F \Rightarrow G$ $F \vdash E$	$G \Rightarrow F$ $G \vdash G$
$G \Rightarrow F$ $G \vdash G$	$H \Rightarrow G$ $H \vdash D$	$H \Rightarrow G$ $H \vdash D$

Now,

After re-checking the transitions again,
here, (A, B) is 'X' and (B, G) is 'X'.

$\{A, G\}, \{B, F\}, \{C, E\}, \{D\}, \{H\}$



Finite Automata with output:-

There are 2 types of finite automata with output.

- 1) Moore Machines
- 2) Mealy Machines

* If the output is associated with state then machine is "Moore Machine".

* If the output is associated with transition then machine is known as "Mealy Machine".

Formal definition for Moore Machine:-

Moore Machine is defined as '6' tuple and is as follows.

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

where,

Q = finite set of states

Σ = finite set of input alphabet.

δ = Transition function.

q_0 = Start state

(big delta) Δ = finite set of output alphabet

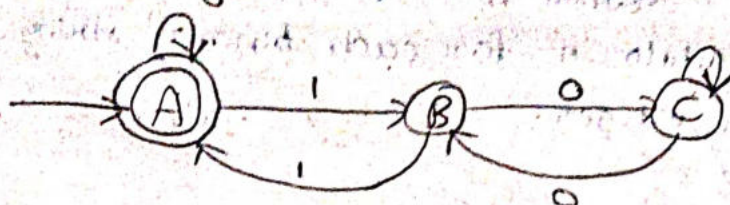
λ = output function.

∴ 1)

Design a moore machine which determines residue modulo 3 for each binary string treated as a binary integer.

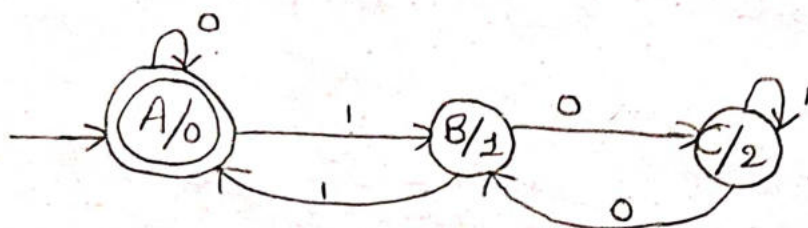
$$\Sigma = \{0, 1\}, \quad \Delta = \{0, 1, 2\}$$

<u>Input</u>	<u>No</u>	<u>Remain</u>	<u>State</u>
000	0	0	A
001	1	1	B
010	2	2	C
011	3	0	A
100	4	1	B
101	5	2	C
110	6	0	A
111	7	1	B



The outputs are $\{0, 1, 2\}$.

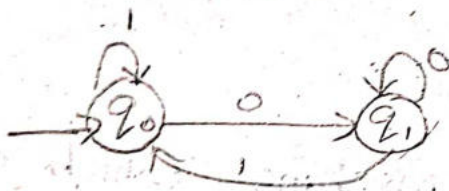
Hence the machine is,



2)

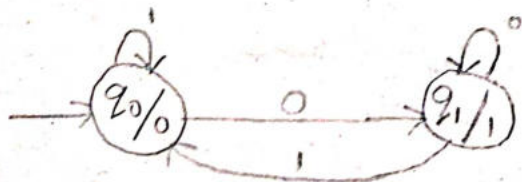
Design a Moore machine for 1's complement

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



The outputs are $\{0, 1\}$ and q_0 .

Hence, The machine is

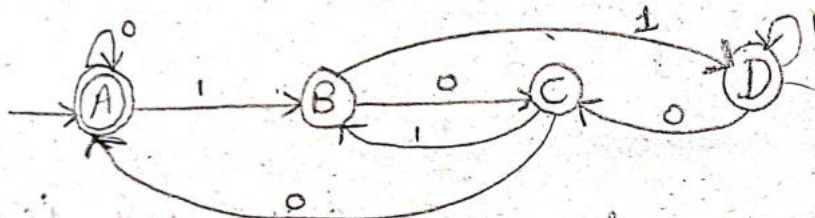


3)

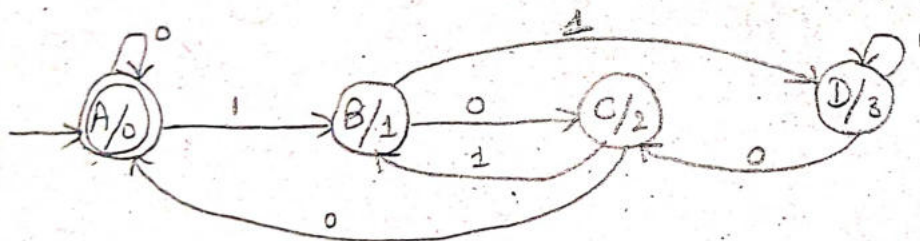
Design a ~~state~~ moore machine which determines residue modulo 4 for each binary string treated as binary integer.

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

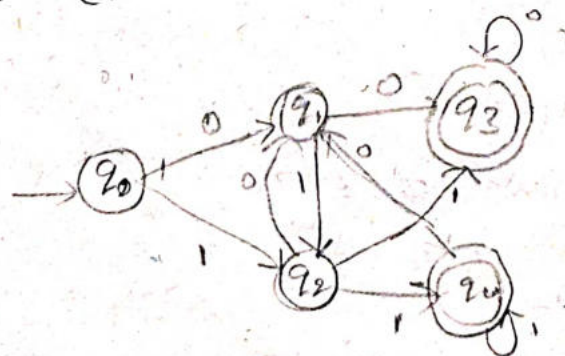
Input	no	remainder	State
000	0	0	A
001	1	1	B
010	2	2	C
011	3	3	D
100	4	0	A
101	5	1	B
110	6	2	C
111	7	3	D



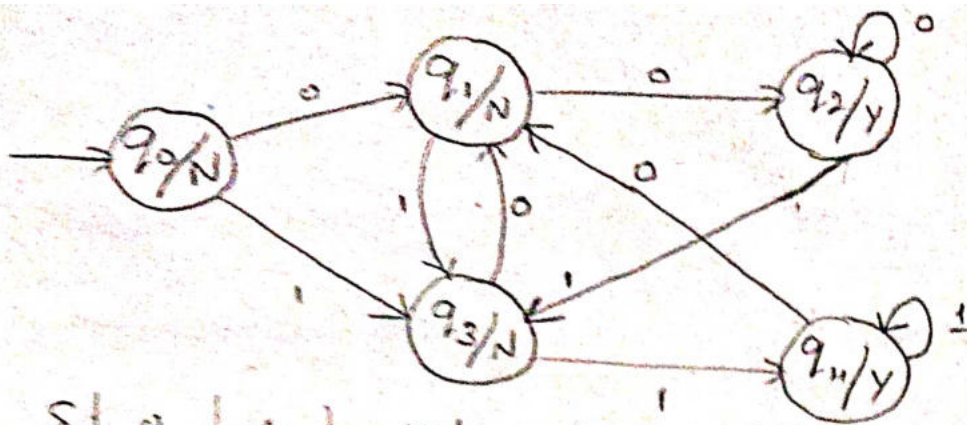
The outputs are $\Delta = \{0, 1, 2, 3\}$



4) Design a moore machine which accepts 0's and 1's which ends with two consecutive zero's (a) two consecutive one's

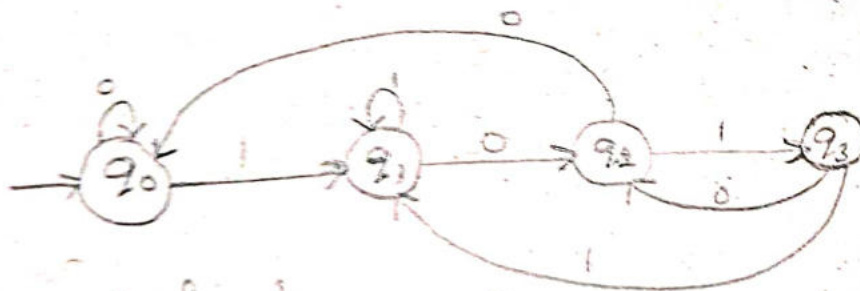


$$\Sigma = \{0, 1\}, \Delta = \{Y, N\}$$

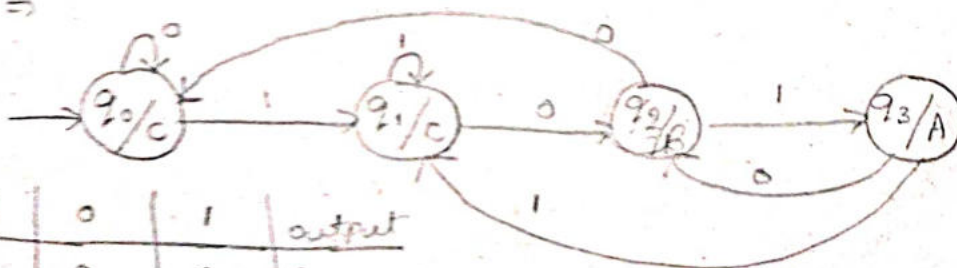


δ	0	1	output
$\rightarrow q_0$	q_1	q_3	N
q_1	q_2	q_3	N
q_2	q_1	q_4	Y
q_3	q_1	q_4	N
q_4	q_2	q_3	Y

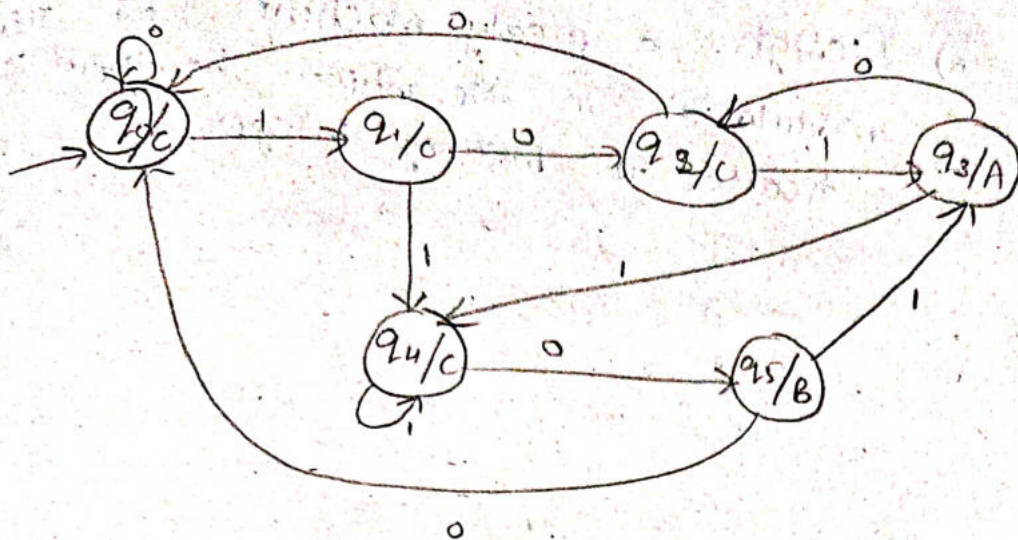
- 5) Design a moore machine which gives 'A' as output when the input ends with 101 and 'B' as output when it ends with 110, otherwise 'C' as output.



$$\Sigma = \{0, 1\}, \Delta = \{A, B, C\}$$



δ	0	1	output
$\rightarrow q_0$	q_0	q_1	C
q_1	q_1	q_1	C
q_2	q_0	q_3	B
q_3	q_1	q_2	A



② Mealy Machine :-

* A mealy machine is a 6 tuple denoted by $M = (Q, \Sigma, \delta, q_0, \Delta, \lambda)$

where,

Q = finite set of states

Σ = finite set of input alphabet

δ = transition function

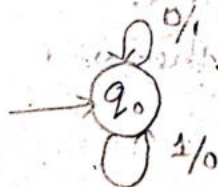
q_0 = start state

Δ = finite set of output alphabet

λ = output function.

$$\lambda \in (Q, a) \longrightarrow \Delta$$

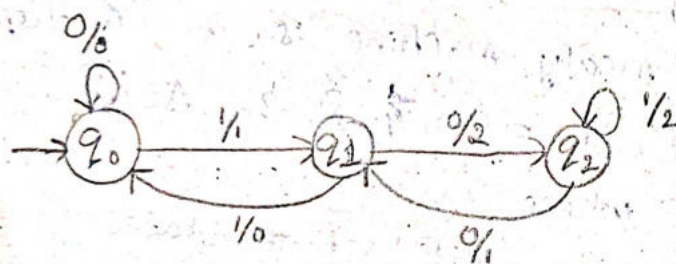
① Design a mealy machine for 1's complement



Present state	input $x = 0$		input $x = 1$	
	Next State	o/p	Next State	o/p
$\rightarrow q_0$	q_0	1	q_0	0

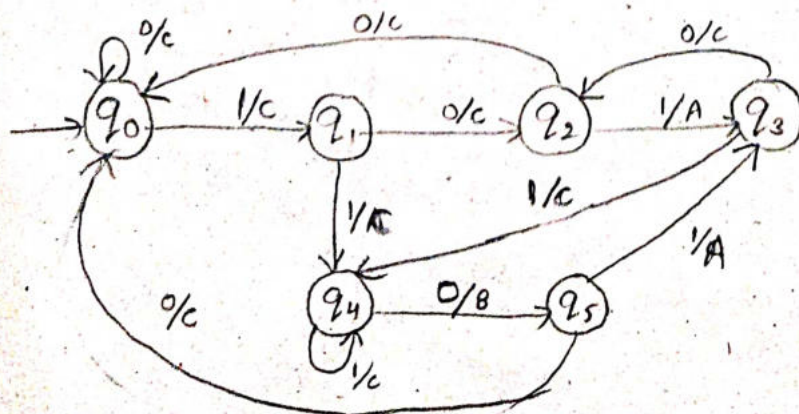
- 2) Design a mealy machine for residue modulo '3' in which the input is treated as binary integer.

Number	Binary	remain	State
0	000	0	A
1	001	1	B
2	010	2	C
3	011	0	A
4	100	1	B
5	101	2	C
6	110	0	A
7	111	1	B



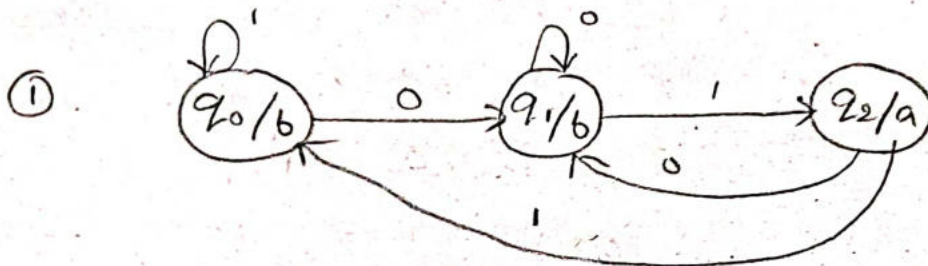
present state	$X = \{0\}$		$X = \{1\}$	
	Next State	o/p	Next State	o/p
q ₀	q ₀	0	q ₁	1
q ₁	q ₂	2	q ₀	0
q ₂	q ₁	1	q ₂	2

- 3) Design a mealy machine which gives 'A' as output when input ends with 101, 'B' as output when input ends with 110. otherwise 'c' as output.



$\frac{d}{dt} = \{0\}$			$\frac{d}{dt} = \{1\}$	
Present State	Next State	O/P	Next State	O/P
q_0	q_0	C	q_1	C
q_1	q_2	C	q_1	C
q_2	q_0	C	q_3	A
q_3	q_2	C	q_4	C
q_4	q_5	B	q_4	C
q_5	q_0	C	q_3	A

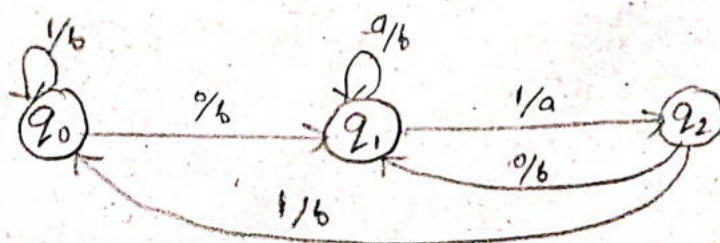
Conversion of moore to mealy machine



P.S	0	1	O/P
q_0	q_1	q_0	b
q_1	q_1	q_2	b
q_2	q_1	q_0	a

Table for mealy

P.S	$\frac{d}{dt} = 0$		$\frac{d}{dt} = 1$	
	N.S	O/P	N.S	O/P
q_0	q_1	b	q_0	b
q_1	q_1	b	q_2	a
q_2	q_1	b	q_0	b

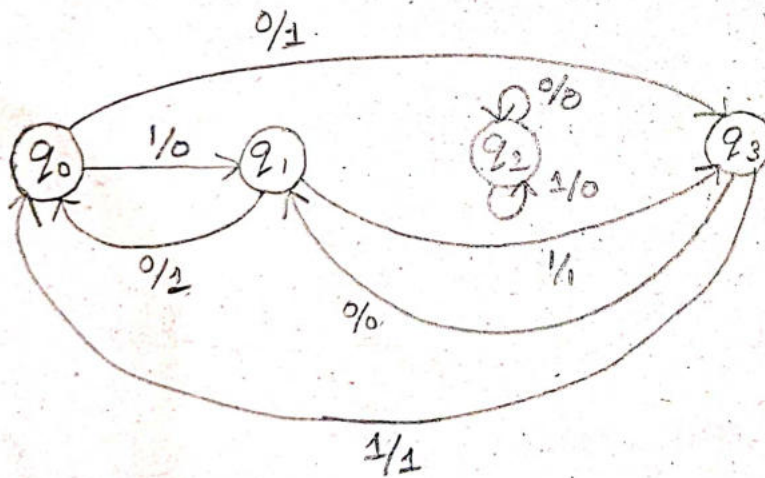


consider the following moose machine transition and convert this into equivalent mealy machine

②

P.S	N.S	q₂	N.S	O/P
q ₀	q ₃		q ₁	1
q ₁	q ₀		q ₃	0
q ₂	q ₂		q ₂	0
q ₃	q ₁		q ₀	1

P.S	0		1	
	N.S	O/P	N.S	O/P
q ₀	q ₃	1	q ₁	0
q ₁	q ₀	1	q ₃	1
q ₂	q ₂	0	q ₂	0
q ₃	q ₁	0	q ₀	1



15/02/24

Conversion of mealy machine to moore machine

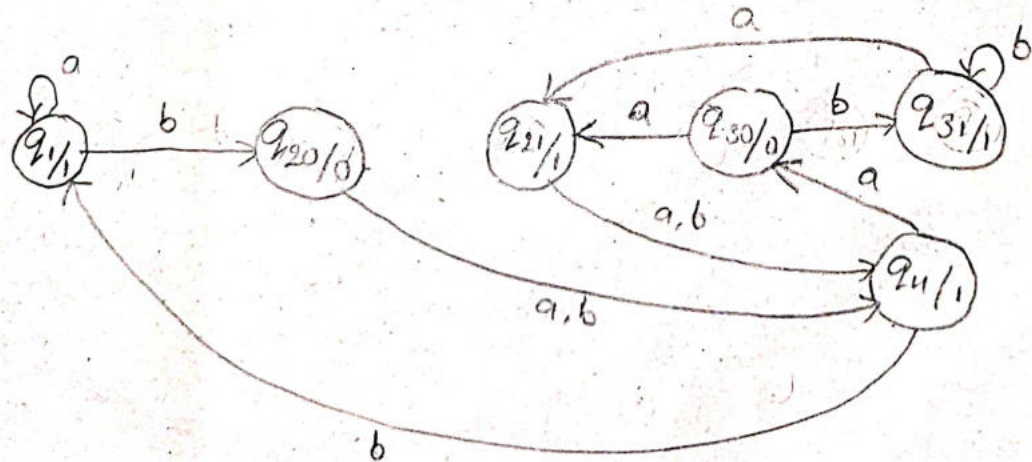
i)

Present state	$\Sigma = a$		$\Sigma = b$	
	Next state	o/p	n.s	o/p
q_1	q_1	1	q_2	0
q_2	q_4	1	q_4	1
q_3	q_2	1	q_3	1
q_4	q_3	0	q_1	1

Given mealy machine, table

Transition table for moore machine

P.S	a	b	o/p
q_1	q_1	q_{20}	1
q_{20}	q_4	q_4	0
q_{21}	q_4	q_4	1
q_{30}	q_{21}	q_{31}	0
q_{31}	q_{21}	q_{31}	1
q_4	q_{30}	q_1	1



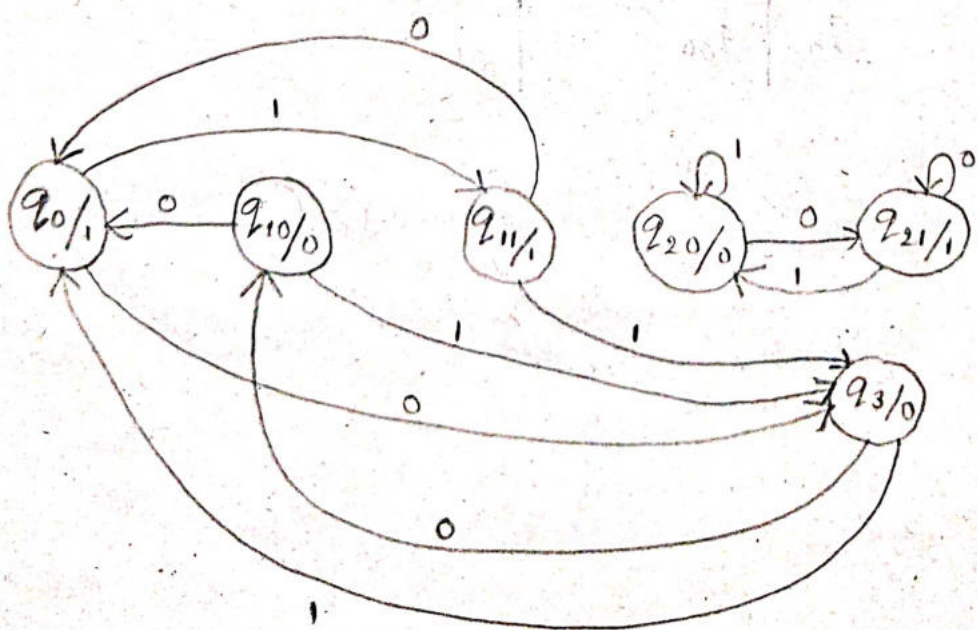
2)

$$\sum_1 = 0$$

$$\sum_1 = 1$$

P.S	N.S	O/P	N.S	O/P
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

P.S	0	1	O/P
q ₀	q ₃	q ₁	1
q ₁₀	q ₀	q ₃	0
q ₁₁	q ₀	q ₃	1
q ₂₀	q ₂₁	q ₂₀	0
q ₂₁	q ₂₁	q ₂₀	1
q ₃	q ₁₀	q ₀	0



19/02/24

2. Regular Expressions & Languages

Regular Sets

These are sets which are accepted by finite automata.

ex:-

$$L = \{\epsilon, 00, 0000, 000000, \dots\}$$

i.e. set of even number of 0's.

Regular Expression (RE)

* It means of representing certain subset of strings over Σ and prove that regular sets are precisely those accepted by finite automata (or) transition systems.

* The Language accepted by finite automata are easily described by simple expression called "regular expressions".

* The Language accepted by some regular expression is known as "regular language".

* Let Σ be an alphabet which is used to denote the input set. The regular expression over Σ can be defined as follows:-

① $\emptyset$ is a regular expression which denotes empty set.

② ϵ is a regular expression and denotes the set $\{\epsilon\}$ and it is a null string.

③ for each a in Σ , a is a regular exp and denotes the $\{a\}$.

④ If π and s are RE's denoting the languages L_1 and L_2 respectively then,

- $\pi + s$ is equivalent to $L_1 \cup L_2$
- πs is equivalent to $L_1 L_2$
- π^* is equivalent to L^*

1) Write the regular expression (RE) for a language accepting all combinations of 'a' over $\Sigma = \{a\}$

A

Given,

$$\Sigma = \{a\}$$

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

$$R.E = a^*$$

2)

Design an R.E for the language of accepting all combinations of 'a' over $\Sigma = \{a\}$ except null string.

A

Given,

$$\Sigma = \{a\}$$

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

$$R.E = a^+$$

- 3) Design an R.E for the language containing any no. of a's and b's over $\Sigma = \{a, b\}$

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

R.E = $(a+b)^*$

- 4) Construct the R.E for the language accepting all the strings which are ending with 00 over $\Sigma = \{0, 1\}$

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

R.E = $(0+1)^*00$

- 5) Design the R.E, starts with 'a' & end with 'b' over $\Sigma = \{a, b\}$.

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

R.E = $a(a+b)^*b$

- 6) If L = the language starting & ending with 'a' and having any combinations of 'b' in b/w them.

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

R.E = $a(b^*)a$

7) from last 3rd digit, should be 'b'
over $\Sigma = \{a, b\}$.

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

$$R.E = (a+b)^* b^+ (a+b) (a+b)$$

8) write a R.E for the language accepting
any no. of a's followed by any no. of
b's and followed by any no. of c's
over $\Sigma = \{a, b, c\}$.

A) $\Sigma = \{a, b, c\}$

$$R.E = a^* b^* c^*$$

9) Construct a R.E for the language which
consists of exactly two b's over
the $\Sigma = \{a, b\}$

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

$$R.E = a^* b a^* b a^*$$

10) write a R.E to denote a language L'
which accepts all the strings with
atleast two b's over $\Sigma = \{a, b\}$

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

$$R.E = a^* b^+ a^* b^+ a^*$$

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

20/09/24

Identities for Regular Expressions:

1) $I_1 : \emptyset + R = R$

2) $I_2 : \emptyset R = R \emptyset = \emptyset$

3) $I_3 : \wedge R = R \wedge = R$

4) $I_4 : \wedge^* = \wedge$

~~5) $I_5 : \emptyset^* = \wedge$~~

5) $I_5 : R + R = R$

6) $I_6 : R^* R^* = R^*$

7) $I_7 : R \cdot R^* = R^* R$

8) $I_8 : (R^*)^* = R^*$

9) $I_9 : \wedge + R \cdot R^* = \wedge + R^* R = R^*$

10) $I_{10} : (PQ)^* P = P(QP)^*$

11) $I_{11} : (P+Q)^* = (P^* Q^*)^* = (P^* + Q^*)^*$

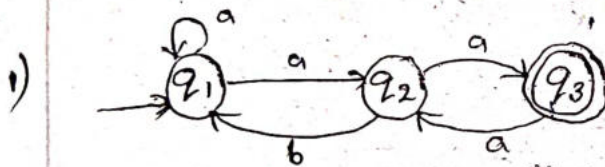
12) $I_{12} : (P+Q)R = PR + QR$

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

Ardan's Theorem

Let P and Q be two regular expressions over Σ , if P does not contain null (Λ) then the following equation, in $R = Q + RP$ has a unique solution (only one solution) given by $R = Q \cdot P^*$

Conversion of finite Automata to Regular Expression



$$q_1 = \Lambda + q_1 a + q_2 b \rightarrow (1)$$

$$q_2 = q_1 a + q_3 a \rightarrow (2)$$

$$q_3 = q_2 a \rightarrow (3)$$

Substitute q_3 in eq (2)

$$q_2 = q_1 a + q_2 a a$$

By Arden's theorem

$$\text{It is in } R = Q + RP$$

$$R = Q P^*$$

$\Rightarrow$ then

$$q_2 = q_1 a (aa)^* \rightarrow (4)$$

Substitute q_2 in eq (1)

$$q_1 = \Lambda + q_1 a + q_1 a (aa)^* b$$

$$q_1 = \Lambda + q_1 (a + a(aa)^* b)$$

By Arden's theorem

$$q_1 = \Lambda (a + a(aa)^* b)^*$$

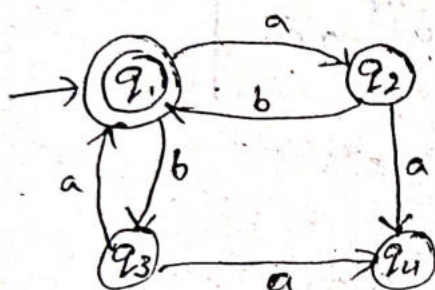
$$q_1 = (a + a(aa)^*b)^* \quad \text{in eq (4)}$$

$$q_2 = q_1 a (aa)^*$$

$$q_2 = (a + a(aa)^*b)^* a (aa)^* \quad \text{in eq (3)}$$

$$q_3 = (a + a(aa)^*b)^* a (aa)^* a$$

②



$$q_1 = \Lambda + q_3 a + q_2 b \rightarrow \textcircled{1}$$

$$q_2 = q_1 a \rightarrow \textcircled{2}$$

$$q_3 = q_1 b \rightarrow \textcircled{3}$$

$$q_4 = q_3 a + q_2 a \rightarrow \textcircled{4}$$

substitute ②, ③ eqns in eq ①

$$q_1 = \Lambda + q_1 b a + q_1 b a$$

$$q_1 = \Lambda + q_1 (ba + ba)$$

$$q_1 = \Lambda + q_1 (aba)$$

By arden's theorem,

$$q_1 = \Lambda (ba)^*$$

$$\Rightarrow q_1 = (ba)^*$$

$$q_2 = q_1 a$$

$$q_2 = (ba)^* a$$

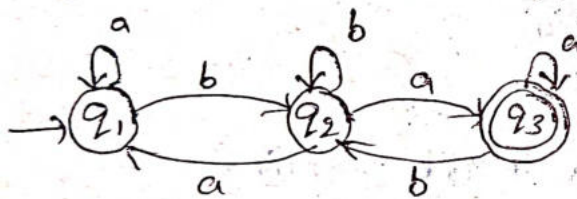
$$q_3 = q_1 b$$

$$q_3 = (ba)^* b$$

$$q_4 = q_3 a + q_2 a$$

$$q_4 = (ba)^* ab + (ba)^* aa$$

③



$$q_1 = 1 + q_1 a + q_2 b \rightarrow \textcircled{1}$$

$$q_2 = q_2 b + q_1 b + q_3 b \rightarrow \textcircled{2}$$

$$q_3 = q_3 a + q_2 a \rightarrow \textcircled{3}$$

from eq(3)

$$q_3 = q_2 a + q_3 a$$

By Arden's theorem, $R = \emptyset + RP$
 $R = \emptyset P^*$

$$q_3 = q_2 a (a)^* \rightarrow \textcircled{4}$$

Substitute eq(4) in eq(2)

$$q_2 = q_2b + q_1b + q_2a(a)^*b$$

$$\Rightarrow q_2 = q_1b + q_2(b + a(a)^*b)$$

By Arden's theorem

$$q_2 = q_1b(b + a(a)^*b)^* \rightarrow (5)$$

Substitute eq(5) in eq(1)

$$q_1 = 1 + q_1a + q_2b$$

$$q_1 = 1 + q_1a + q_1bb(b + a(a)^*b)^*$$

$$\Rightarrow q_1 = 1 + q_1[a + bb(b + a(a)^*b)^*]$$

$$\Rightarrow q_1 = 1 [a + bb(b + a(a)^*b)^*]^*$$

$$\Rightarrow q_1 = [a + bb(b + a(a)^*b)^*]^*$$

q_1 in (4) & (5) eqns-

$$q_2 = q_1b(b + a(a)^*b)^*$$

$$q_2 = (b) [a + bb(b + a(a)^*b)^*]^* [b + a(a)^*b]^*$$

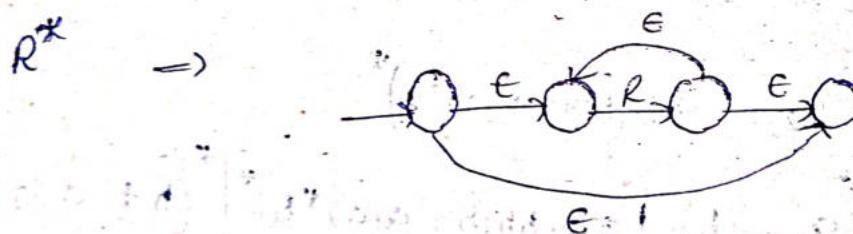
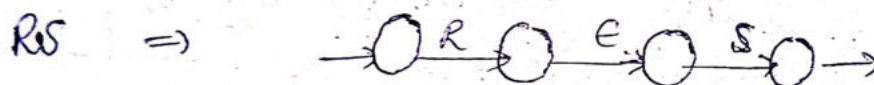
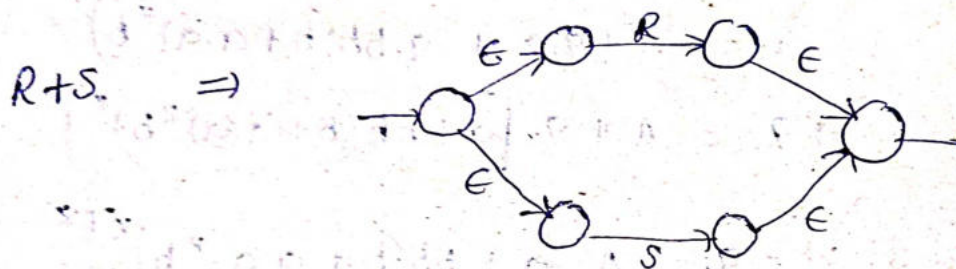
$$q_3 = q_2a(a)^*$$

$$q_3 = ab(a)^* [a + bb(b + a(a)^*b)^*]^* [b + a(a)^*b]^*$$

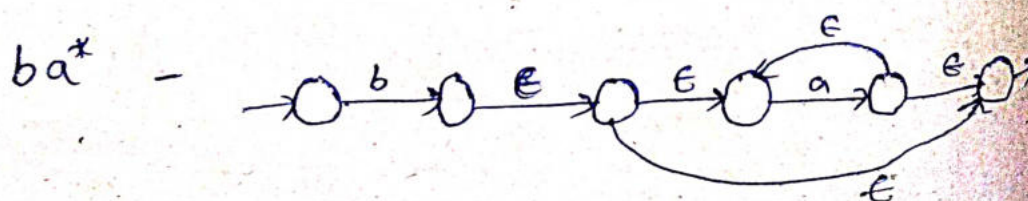
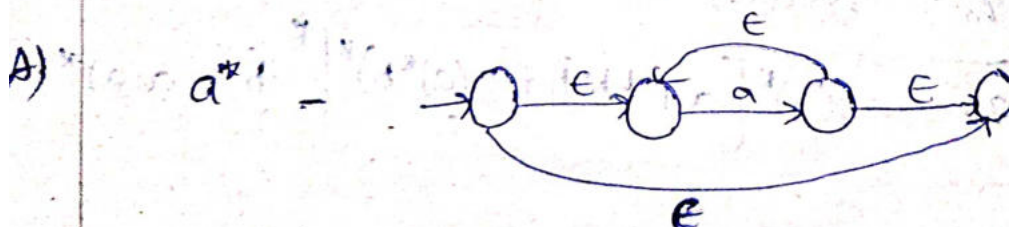
Ques

Conversion of Regular Expression to finite Automata

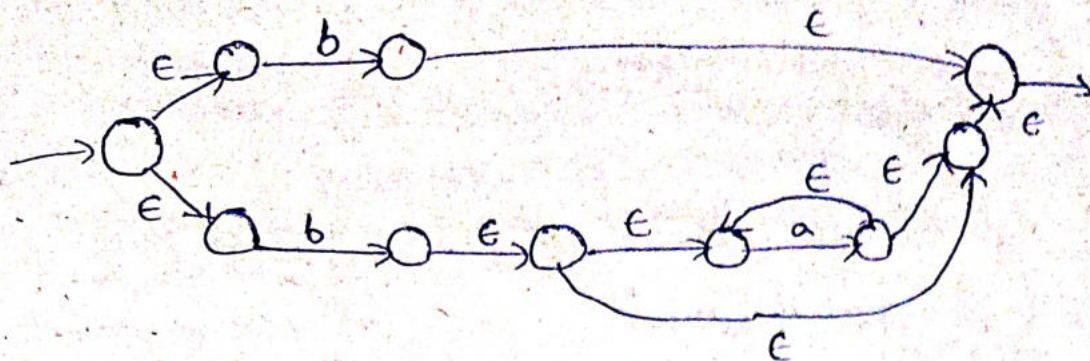
Notations :-



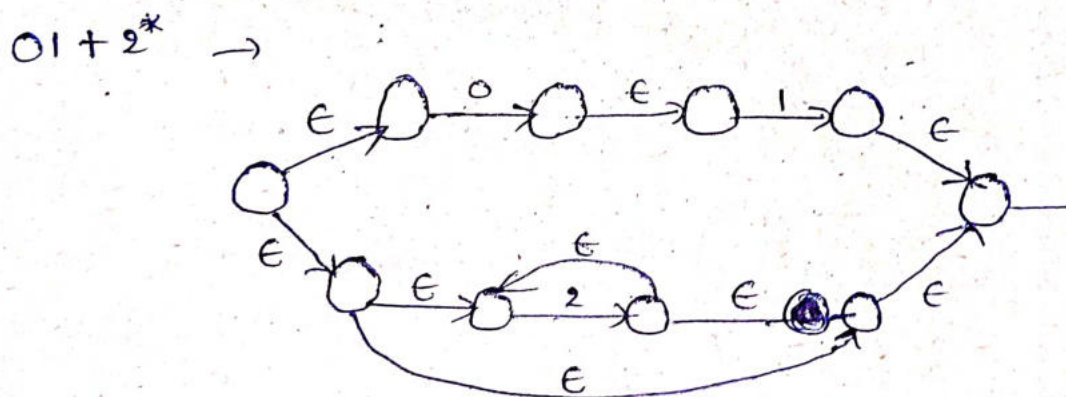
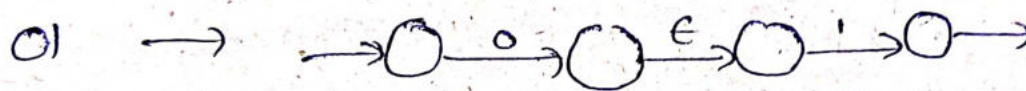
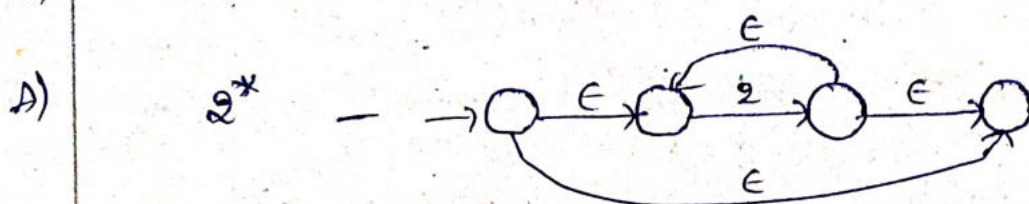
1) Construct NFA for $RE = b + ba^*$



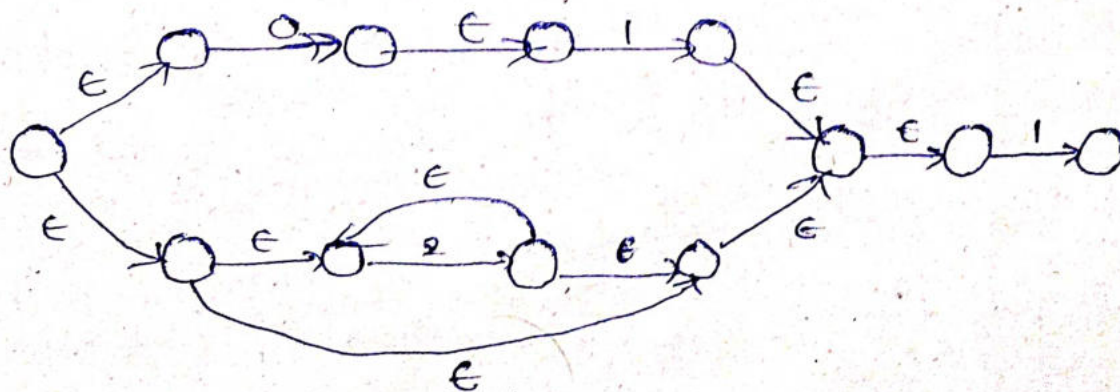
$b + ba^*$



Q) Construct an NFA for $RE = (01 + 2^*)1$

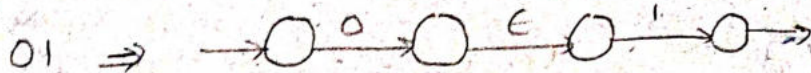


$(01 + 2^*)1$

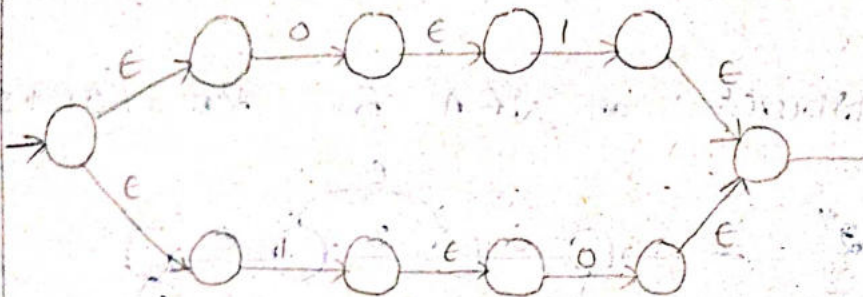


3)

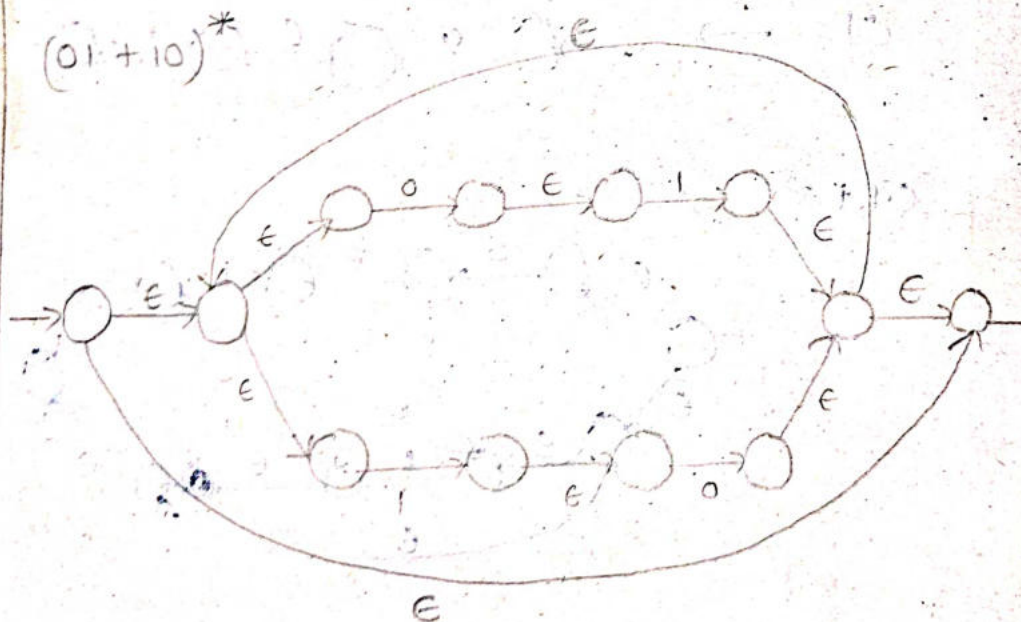
$$(01 + 10)^*$$



$01 + 10$



$$(01 + 10)^*$$



Imp. Pumping Lemma for Regular Sets

* Pumping Lemma is a powerful technique used for proving a language is not regular.

* It cannot be used to prove that a language is regular.

Statement :-

* Let L be a regular language & there exists a constant n (depends on language) such that for every string w in L with $|w| \geq n$ then we can break or divide w into 3 strings as

• $w = xyz$ such that

(i) $|y| > 0$

(ii) $|xy| \leq n$

(iii) for every $i \geq 0$ the string xy^iz is also in L .

* If any language satisfies all the three then the language is said to be regular.

Steps To prove a language is not regular using Pumping Lemma :-

1) Assume the language L is regular.

2) Let n be the constant (or) choose n value depends on L .

3) Pick any string w from the language L such that $|w| \geq n$.

4) Split the string w into 3 strings into

such that $w = xyz$ such that

(i) $|y| > 0$

(ii) $|xy| \leq n$

(iii) xy^iz belongs to L for $i \geq 0$

5) If any string xy^iz is not in L then it violates the pumping lemma. Hence, we say that L is not regular.

23/02/24

1) Show that Language $L = \{0^n 1^n / n \geq 1\}$ is not regular.

sol

Let us assume,

L is regular

Take, $n=4$

$$\Rightarrow w = 0^4 1^4$$

$$= 0000 1111$$

Case - (i) :- y is in 0's

$$w = \frac{0000}{x} \frac{1111}{y} \frac{1111}{z}$$

(i) $|y| > 0$

$2 > 0$ (T)

(ii) $|xy| \leq n$

$4 \leq 4$ (T)

(iii) xy^iz

Assume $i=2$

$$= 00(00)^2 1111$$

$$= 000000 1111$$

As it is not in L , we conclude that given L is not regular.

Case - (ii) :-

y is in 1's.

$$W = \frac{0000}{x} \frac{111}{y} \frac{11}{z}$$

$$\begin{array}{l|l} \textcircled{i} & |y| > 0 \\ & 2 > 0 \text{ (T)} \end{array} \quad \left| \quad \begin{array}{l} \textcircled{ii} & |xy| \leq n \\ & 6 \leq 4 \text{ (F)} \end{array} \right.$$

$$\textcircled{iii} \quad xy^iz$$

i = 2 (Assume)

$$= 0000(11)^2 11$$

$$= 0000 11 11 11$$

As it is not in L' ,
we conclude that, given L' is not regular.

Case - (iii) :-

y is in both 0 and 1

$$W = \frac{000}{x} \frac{0111}{y} \frac{111}{z}$$

$$\begin{array}{l|l} \textcircled{i} & |y| > 0 \\ & 2 > 0 \text{ (T)} \end{array} \quad \left| \quad \begin{array}{l} \textcircled{ii} & |xy| \leq n \\ & 5 \leq 4 \text{ (F)} \end{array} \right.$$

$$\textcircled{iii} \quad xy^iz$$

i = 2 (Assume)

$$= 000(01)^2 111$$

$$= 000 01 01 111$$

As it is not in L' ,
we conclude that, Given L' is not regular.

From Above 3 cases,
we say that,

Given L is not regular.

show that $L = \{0^p/p \text{ is a prime number}\}$
is not regular.

Let us assume,

L is ~~not~~ regular.

Let us take,

$$p = 2, 3, 5, 7, \dots$$

take, $p = 3 = n$

$$N = 0^3 = 000$$

Case - (i) :- y is in o's

$$W = \begin{pmatrix} 0 & 0 & 0 \\ x & y & z \end{pmatrix}$$

(i) $|y| > 0$ (ii) $|xy| \leq n$
 $1 > 0$ (T) $2 \leq 3$

(iii) $x y^i z$
 $i = 2$ (Assume)
 $= 0(0)^2 0$
 $= 0000$

As it is not in $\mathcal{L}$.

We conclude that,

Given L' is not regular.

3) show that the language $L = \{ww^R / w \in (0,1)^*\}$ is not regular.

Sol)

Let us assume,

L is regular.

$$w = 001$$

$$n = 3$$

~~Let us take,~~

Let,

Given,

~~$w = ww^R$~~

$$w = 001100$$

Case - (i):-

y is in 0's

$$w = \frac{00}{y} \frac{11}{x} \frac{00}{z}$$

$$(i) \quad |y| > 0 \quad |x| > 0 \quad (T)$$

$$(ii) \quad |xy| \leq n \quad 4 \leq 3 \quad (F)$$

$$(iii) \quad xy^iz$$

$$i = 2$$

$$= (00)^2 \cdot 1100$$

$$= 00001100$$

As it is not in L :

we conclude that,

Given L is not regular.

Closure properties of regular sets :-

- * Regular languages are closed under "union".
- * Regular languages are closed under "Concatenation".
- * Regular languages are closed under "intersection".
- * Regular languages are closed under "Kleen Closure".
- * Regular languages are closed under "reversal".

3. Regular Grammar.

Regular Grammar

* It is defined as, $G = (V, T, P, S)$ where,

$V =$ ~~var~~ set of variables
(or)
Non-Terminal symbols.

$T =$ set of Terminal Symbols.

$P =$ set of production rules.

$S =$ starting symbol which belongs V .

The production rules P are of the form,

$A \rightarrow aB$, $A \rightarrow a$ where A, B are non-terminal symbols and a is a terminal symbols.

ex:-

Consider $G = (V, T, P, S)$ with $V = \{S, A\}$,

$T = \{0, 1\}$, $S = \{S\}$, production rules P are
 $S \rightarrow 0S$, $S \rightarrow 1B$, $B \rightarrow \epsilon$

This is called a regular language and it can be represented by some DFA.

Right Linear Grammar:-

* A Grammar $G = (V, T, P, S)$ is right linear if each production of the grammar has one of the following '3' forms.

(i) $A \rightarrow aB$

(ii) $A \rightarrow a$

(iii) $A \rightarrow \epsilon$

examples:-

$S \rightarrow 0A$

$A \rightarrow 10A$

$A \rightarrow \epsilon$

Left Linear Grammar

A Grammar, $G_1 = (V, T, P, S)$ is left linear if each production of the grammar has one of the following 3 forms.

(i) $A \rightarrow Ba$

(ii) $A \rightarrow a$

(iii) $A \rightarrow \epsilon$

examples :-

(i) $S \rightarrow S10$

$S \rightarrow 0$

(ii)

$S \rightarrow a$

$S \rightarrow aB/\epsilon$

$S \rightarrow Ab$

It consists of right and left linear then we cannot call it as "regular grammar".

Equivalence b/w regular grammar and finite automata

Construction of R.E from a given finite automata

Let, $G_1 = (V, T, P, S)$ where

V = set of all variables

T = set of all terminals

P = set of all productions

S = starting symbol

To obtain the productions of the grammar

i.e. $\{P\}$ we apply the following cases:-

Case (i):- If the initial state not equal to final state
i.e. $q_0 \notin F$ then apply following steps

(a) If there is a transition, $\delta(q_i, a) = q_j \notin F$
then include a production

(b) § If there is a transition $\delta(q_i, a) = q_j \in F$ then include the production,

$$\left. \begin{array}{l} q_i \rightarrow a q_j \\ q_i \rightarrow a \end{array} \right\} \text{ in } P.$$

(c) An initial state of finite automata is the start symbol of 'G'.



(i) $(A, a) \rightarrow A$
 $A \rightarrow aA$

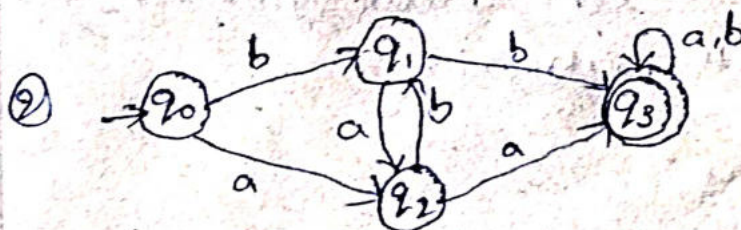
(iii) $(B, a) \rightarrow B$
 $B \rightarrow aB$
 $B \rightarrow a$

(ii) $(A, b) \rightarrow B$
 $A \rightarrow bB$
 $A \rightarrow b$

(iv) $(B, b) \rightarrow B$
 $B \rightarrow bB$
 $B \rightarrow b$

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

$$= \left(\{A, B\}, \{a, b\}, \left\{ \begin{array}{l} A \rightarrow aA, A \rightarrow bB, A \rightarrow b, B \rightarrow a \\ B \rightarrow a, B \rightarrow bB, B \rightarrow b \end{array} \right\}, \{A\} \right)$$



(i) $(q_0, a) \rightarrow q_2$

(ii) $(q_0, b) \rightarrow q_1$
 $q_0 \rightarrow b q_1$

$$q_0 \rightarrow a q_2$$

(iii) $(q_1, a) \rightarrow q_2$

(iv) $(q_1, b) \rightarrow q_3$

$$q_1 \rightarrow a q_2$$

$$\begin{array}{l} q_1 \rightarrow b q_3 \\ q_1 \rightarrow b \end{array}$$

$$(q_2, b) \rightarrow q_1$$

$$(q_2, a) \rightarrow q_3$$

$$q_2 \rightarrow aq_1$$

$$q_2 \rightarrow aq_3$$

$$q_2 \rightarrow a$$

$$(q_3, a) \rightarrow q_3$$

$$(q_3, b) \rightarrow q_3$$

$$q_3 \rightarrow aq_3$$

$$q_3 \rightarrow bq_3$$

$$q_3 \rightarrow a$$

$$q_3 \rightarrow b$$

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

$$= (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{q_0 \rightarrow aq_2, q_0 \rightarrow bq_1,$$

$$\Rightarrow V = \{q_0, q_1, q_2, q_3\}$$

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

$$P = \{q_0 \rightarrow aq_2, q_0 \rightarrow bq_1, q_1 \rightarrow aq_2, q_1 \rightarrow bq_3, q_1 \rightarrow b, q_2 \rightarrow aq_1, q_2 \rightarrow aq_3, q_2 \rightarrow a, q_3 \rightarrow aq_3, q_3 \rightarrow a, q_3 \rightarrow bq_3, q_3 \rightarrow b\}$$

$$S = \{q_0\}$$

Construction of Left Linear grammar for given finite automata

Step-1 :-

Reverse the directions of all the edges.

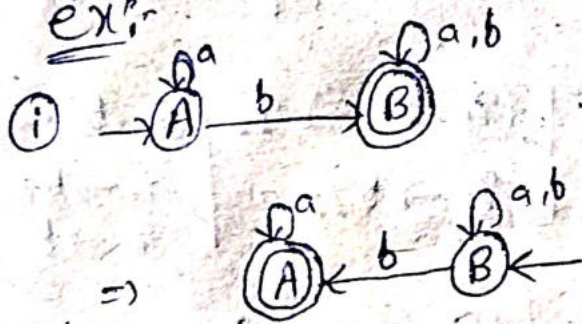
Step-2 :-

Make the initial state as final state and final state as initial state.

Step-3

After construction of new finite automata find the right linear grammar and then reverse the rightside of all productions to get the result in left linear grammar.

ex:-



$$\begin{array}{ll} (A, a) \rightarrow A & (B, b) \rightarrow A \\ A \rightarrow aA & B \rightarrow bA \\ A \rightarrow a & B \rightarrow b \end{array}$$

$$\begin{array}{ll} (B, a) \rightarrow B & (B, b) \rightarrow B \\ B \rightarrow aB & B \rightarrow bB \end{array}$$

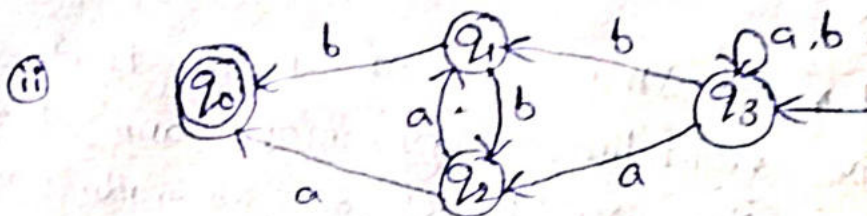
reverse the rightside of all productions.

$$\begin{array}{l} A \rightarrow Aa \\ A \rightarrow a \\ B \rightarrow Ab \\ B \rightarrow b \\ B \rightarrow Ba \\ B \rightarrow bB \end{array}$$

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

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

$$P = \{A \rightarrow Aa, A \rightarrow a, B \rightarrow Ab, B \rightarrow b, B \rightarrow Ba, B \rightarrow bB\}$$



$$\begin{array}{c|c|c}
 (q_3, a) \rightarrow q_3 & (q_3, b) \rightarrow q_3 & (q_3, a) \rightarrow q_2 \\
 q_3 \rightarrow a q_3 & q_3 \rightarrow b q_3 & q_3 \rightarrow a q_2
 \end{array}$$

$$\begin{array}{c|c|c}
 (q_3, b) \rightarrow q_1 & (q_1, b) \rightarrow q_2 & (q_1, b) \rightarrow q_0 \\
 q_3 \rightarrow b q_1 & q_1 \rightarrow b q_2 & q_1 \rightarrow b q_0 \\
 & & q_1 \rightarrow b
 \end{array}$$

$$\begin{array}{c|c}
 (q_2, a) \rightarrow q_1 & (q_2, a) \rightarrow q_0 \\
 q_2 \rightarrow a q_1 & q_2 \rightarrow a q_0 \\
 & q_2 \rightarrow a
 \end{array}$$

Now, reverse all the productions.

$$q_3 \rightarrow q_3 a, q_3 \rightarrow q_3 b, q_3 \rightarrow q_2 a, q_3 \rightarrow q_1 b$$

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

$$q_2 \rightarrow q_0 a, q_2 \rightarrow a$$

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

$$V = \{q_0, q_1, q_2, q_3\}, T = \{a, b\}, S = \{q_3\}$$

$$P = \{q_3 \rightarrow q_3 a, q_3 \rightarrow q_3 b, q_3 \rightarrow q_2 a, q_3 \rightarrow q_1 b, q_1 \rightarrow q_2 b,$$

Conversion of regular grammar to finite Automata

Step-1:-

Each production, $A_i \rightarrow a A_j$ includes a transition from $q_i \rightarrow q_j$ labeled on 'a'.

Step-2:-

Each production $A_k \rightarrow a$ includes a transition from $q_k \xrightarrow{a} q_f$ where q_f is final state.

Construct finite Automata for the given grammar

$$\textcircled{i} \quad G = (\{A_0, A_1\}, \{a, b\}, P, A_0)$$

where,

$$P = \{A_0 \rightarrow a A_1, A_1 \rightarrow b A_1, A_1 \rightarrow a, A_1 \rightarrow b A_0\}$$



Steps:-



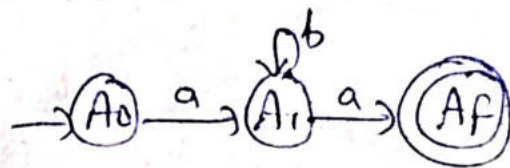
ii $A_0 \rightarrow a A_1$



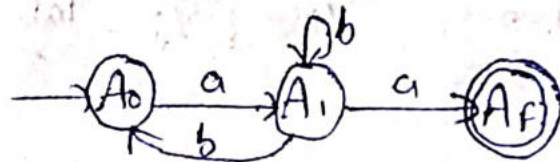
iii $A_1 \rightarrow b A_1$



(iv) $A_1 \rightarrow a$

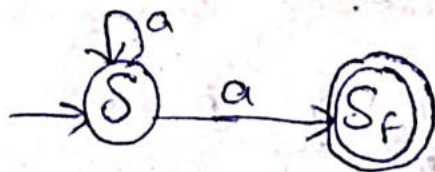


(v) $A_1 \rightarrow bA_0$



(2) $S \rightarrow as$

$S \rightarrow a$

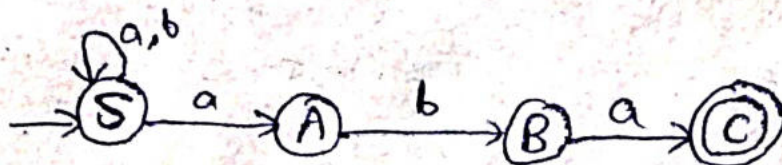


(3) $S \rightarrow as / bs / aA$

$A \rightarrow bB$

$B \rightarrow aC$

$C \rightarrow \lambda$



3. (b) Context free Grammar

* The Context free Grammar (CFG), $G = (V, T, P, S)$
where,

V = set of variables

T = set of terminals

S = start symbol

P = set of production rules.

* Every production rule derives, $\alpha \rightarrow \beta$
where $|\alpha| = 1$ & $\alpha \in V$

$\beta \in (V + T)^*$

* The languages generated by Context free Grammar are called "Context free languages".

ex:

① Let $G = (\{S\}, \{a\}, \{S\}, \{S\})$, P is defined as

$S \rightarrow aS$
 $S \rightarrow \epsilon$

defines the language.

②

$S \rightarrow aS$

$S \rightarrow \epsilon$

$S = aS$

$S \rightarrow aS$

$S \rightarrow aaaS$

$S \rightarrow aaaS$

$S = \epsilon$

a

aa

aaa

$\therefore$ The language defined is a^*

$$② \quad G = (\{S\}, \{a, b\}, \{S\}, \{P\})$$

P defined as $S \rightarrow aSa / bSb$

$S \rightarrow a/b/e$

①

$$S \rightarrow aSa$$

$$S \rightarrow aaSaa$$

$$S \rightarrow aaaSaaa$$

$$S \rightarrow bSb$$

$$S \rightarrow bbsbb$$

$$S \rightarrow bbbSbbb$$

$$S \rightarrow aSa$$

$$S \rightarrow absba$$

$$S \rightarrow ababa$$

$$S \rightarrow abbbba$$

$$S \rightarrow abba$$

$$s = bSb$$

$$s = a$$

$$s = b$$

$$s = \epsilon$$

from the above strings, we conclude that

$\therefore$ The language is defined by P is

all palindromes.

③

$$G = (\{S\}, \{a, b\}, \{S\}, \{P\})$$

$$P = \{S \rightarrow asb, S \rightarrow \epsilon\}$$

①

$$S \rightarrow asb$$

$$S \rightarrow aasbb$$

$$S \rightarrow aaasbbb$$

$$s = \epsilon$$

$$a^1b^1$$

$$a^2b^2$$

$$a^3b^3$$

from all these cases, the language is $a^n b^n$

$$(4) G = (\{S\}, \{a, b\}, \{S\}, \{P\})$$

$$S \rightarrow Sasbs$$

$$S \rightarrow Sbsas$$

$$S \rightarrow \epsilon$$

(A)

$$S \rightarrow Sasbs$$

$$S \rightarrow Sbsas$$

$$S \rightarrow \epsilon$$

$$S \rightarrow ab$$

$$S \rightarrow ba$$

~~Sbsasasbs~~

Hence the language consists of equal no. of a's and equal no. of b's.

$$(5) G = (\{S\}, \{a, b\}, \{S\}, \{P\})$$

$$S \rightarrow asbb / abb$$

(A)

$$S \rightarrow asbb$$

$$S \rightarrow aabbbb$$

$$S \rightarrow asbb$$

$$S \rightarrow aasbbbbb$$

$$S \rightarrow aaasbbbbbb$$

$$a^n \cdot b^{2n}$$

Hence the language is $a^n \cdot b^{2n}$.

Derivation using Grammar

The process of deriving string of terminals from the start symbol by repeatedly replacing a variable (or) Non-terminal by set of productions is called "Derivation".

* It is denoted by " $\Rightarrow$ "

15/03/24

1)

Consider $G = (\{E\}, \{+, -, *, /, (,), id, \{s\}, \{p\}\})$.

P is defined as

$$E \rightarrow E * E$$

$$E \rightarrow E + E$$

$$E \rightarrow (E)$$

$$E \rightarrow E - E$$

$$E \rightarrow E / E$$

$$E \rightarrow id$$

derive the string $id * id + id$.

A

$$E \Rightarrow E * E \quad (E \rightarrow E * E)$$

$$\Rightarrow E * E + E \quad (E \rightarrow E + E)$$

$$\Rightarrow id * E + E \quad (E \rightarrow id)$$

$$\Rightarrow id * id + E \quad (E \rightarrow id)$$

$$E \Rightarrow id * id + id \quad (E \rightarrow id)$$

2) $G_1 = (\{S\}, \{a\}, S, P)$ P is defined by
 $S \rightarrow aS, S \rightarrow \epsilon$ & derive the string aaa

A) $S \Rightarrow aS \quad (S \rightarrow aS)$
 $S \Rightarrow aaS \quad (S \rightarrow aS)$
 $S \Rightarrow aaaS \quad (S \rightarrow aS)$
 $S \Rightarrow aaa \quad (S \rightarrow \epsilon)$

3) $G = (\{S\}, \{a, b\}, S, P)$ P is defined by
 $S \rightarrow aS/bS/a/b$ & derive the string abba

A) $S \Rightarrow aS \quad (S \rightarrow aS)$
 $S \Rightarrow abS \quad (S \rightarrow bS)$
 $S \Rightarrow abbs \quad (S \rightarrow bS)$
 $S \Rightarrow abbaS \quad (S \rightarrow aS)$
 $S \Rightarrow abbab \quad (S \rightarrow b)$

Left Most Derivation:- $\left[\begin{array}{c} \text{---} \\ \text{Lmd} \end{array} \right]$

The process of obtaining a string of terminals from a sequence of replacements such that everytime leftmost non-terminal (or) variable is replaced at each and every step is called "Left most Derivation".

Right most Derivation:- $\left[\begin{array}{c} \text{---} \\ \text{rmd} \end{array} \right]$

The process of obtaining of a string of terminals from a sequence of replacements such that replacing the right most variable at each and every step is called "Right most Derivation".

1) Consider $G = (\{E\}, \{+, -, *, /, (,)\}, \{id\}, \{id\}, \{id\})$
derive the string id + id * id by using left most derivation and right most derivation. Given productions are
 $E \rightarrow E * E, E \rightarrow E + E, E \rightarrow E - E, E \rightarrow (E),$
 $E \rightarrow E / E, E \rightarrow id$

LM D:-

$E \xRightarrow{\text{Lmd}} E + E \quad (E \rightarrow E + E)$
 $\Rightarrow id + E \quad (E \rightarrow id)$
 $\Rightarrow id + E * E \quad (E \rightarrow E * E)$
 $\Rightarrow id + id * E \quad (E \rightarrow id)$
 $\Rightarrow id + id * id \quad (E \rightarrow id)$
 $\Rightarrow id + id * id$

$E \xRightarrow{\text{rmd}} E * E \quad (E \rightarrow E * E)$
 $\Rightarrow E + E * E \quad (E \rightarrow E + E)$
 $\Rightarrow id + E * E \quad (E \rightarrow id)$
 $\Rightarrow id + id * E \quad (E \rightarrow id)$
 $\Rightarrow id + id * id \quad (E \rightarrow id)$
 $\Rightarrow id + id * id$

RMD:-

$$E \xRightarrow{\text{RMD}} E + E \quad (E \rightarrow E + E)$$

$$\Rightarrow E + E * E \quad (E \rightarrow E * E)$$

$$\Rightarrow E + E * \text{id} \quad (E \rightarrow \text{id})$$

$$\Rightarrow E + \text{id} * \text{id} \quad (E \rightarrow \text{id})$$

$$\Rightarrow \text{id} + \text{id} * \text{id} \quad (E \rightarrow \text{id})$$

$$\Rightarrow \text{id} + \text{id} * \text{id}$$

$$E \xRightarrow{\text{RMD}} E * E \quad (E \rightarrow E * E)$$

$$\Rightarrow E + E * E \quad (E \rightarrow E + E)$$

$$\Rightarrow E + E * \text{id} \quad (E \rightarrow \text{id})$$

$$\Rightarrow E + \text{id} * \text{id} \quad (E \rightarrow \text{id})$$

$$\Rightarrow \text{id} + \text{id} * \text{id} \quad (E \rightarrow \text{id})$$

$$\Rightarrow \text{id} + \text{id} * \text{id}$$

2)

Consider $G = (\{S, X\}, \{a, b\}, S, P)$ where P

defined as $S \rightarrow baXas/ab$ then derive

$$X \rightarrow Xab/aa$$

$baaaababaaab$. And the lmd & rmd above string.

A)

LMD:-

$$S \Rightarrow baXas \quad (S \rightarrow baXas) \quad (X \rightarrow)$$

$$\cancel{S \Rightarrow baaaaas} \quad (X \rightarrow aa)$$

$$\cancel{S \Rightarrow baaaabaxas} \quad (S \rightarrow baxas)$$

$$\cancel{S \Rightarrow baxabaxas}$$

$$\Rightarrow baXabas \quad (X \rightarrow Xab)$$

$$\Rightarrow baXababas \quad (X \rightarrow aa)$$

$$\Rightarrow baaaaababaaab \quad (S \rightarrow ab)$$

$$\Rightarrow baaaaababaaab$$

RMD :-

$S \Rightarrow_{\text{RMD}} baxas \quad (S \rightarrow baxas)$

$\Rightarrow baxabas \quad (x \rightarrow xab)$

$\Rightarrow baxababas \quad (x \rightarrow xab)$

$\Rightarrow baxababaab \quad (S \rightarrow ab)$

$\Rightarrow baaababab \quad (x \rightarrow aa)$

③

$G = (\{S, A, B\}, \{a, b\}, S, P)$ where

P is defined as $S \rightarrow aB/bA$

$A \rightarrow a/as/bA$

$B \rightarrow b/bS/aBB$

then

derive

$aabbabba$.

A)

LMD :-

$S \Rightarrow aB \quad (S \rightarrow aB)$

$S \Rightarrow aab \quad (S \rightarrow aB)$

$S \Rightarrow aaba \quad (S \rightarrow bA)$

$S \Rightarrow aabba \quad (S \rightarrow bA)$

$S \Rightarrow aabbas \quad (A \rightarrow as)$

$S \Rightarrow aabbab \quad (S \rightarrow bA)$

$S \Rightarrow aabbabba \quad (A \rightarrow bA)$

$S \Rightarrow aabbabbaa \quad (A \rightarrow a)$

RMD:-

$$\begin{aligned} S &\Rightarrow aB \quad (S \rightarrow aB) \\ &\Rightarrow aab \quad (S \rightarrow aB) \\ &\Rightarrow aab \end{aligned}$$

3) $G = (\{S, A, B\}, \{a, b\}, S, P)$ where
 P is defined as $S \rightarrow aB/bA$,
 $A \rightarrow a/as/bA$ then
 $B \rightarrow b/bS/aBB$
 derive $aabbabba$ using lmd, rmd .

LMD:-

$$\begin{aligned} S &\Rightarrow aB \quad (S \rightarrow aB) \\ &\Rightarrow aabb \quad (B \rightarrow aBB) \\ &\Rightarrow aabb \quad (B \rightarrow b) \\ &\Rightarrow aabbS \quad (B \rightarrow bS) \\ &\Rightarrow aabbab \quad (S \rightarrow aB) \\ &\Rightarrow aabbabs \quad (B \rightarrow bS) \\ &\Rightarrow aabbabba \quad (S \rightarrow bA) \\ &\Rightarrow aabbabba \quad (A \rightarrow a) \end{aligned}$$

RMD:-

$$\begin{aligned} S &\Rightarrow aB \quad (S \rightarrow aB) \\ &\Rightarrow aabb \quad (B \rightarrow aBB) \\ &\Rightarrow aabbs \quad (B \rightarrow bS) \\ &\Rightarrow aabbsA \quad (S \rightarrow bA) \\ &\Rightarrow aabbsba \quad (A \rightarrow a) \\ &\Rightarrow aabbsbba \quad (B \rightarrow bS) \\ &\Rightarrow aabbabba \quad (S \rightarrow bA) \\ &\Rightarrow aabbabba \quad (A \rightarrow a) \\ &\Rightarrow aabbabba \end{aligned}$$

Sentential form :-

Let $G = (V, T, P, S)$ is a context free grammar, any string 'w' in $(V+T)^*$ which is derived from start symbol such that $S \xRightarrow{*} w$ is called as "sentential form of terminals".

* If a string is derived from left most derivation then string of terminals is called "Left sentential form".

* If a string is derived from right most derivation then string of terminals is called "right sentential form".

Derivation tree (or) Syntax tree (or) Parse tree

The derivation can be shown pictorially as a tree to illustrate how a string is derived from context free grammar such trees are called as "Derivation trees".

Formal definition of Derivation Tree :-

* Let $G = (V, T, P, S)$ be a context free grammar of derivation tree (or) Parse tree then these are properties shown below.

Step-1 :-

The root node as label 'S' (start symbol)

Step-2 :-

Every node has a label which is a in

Step-3;

Every leaf node labeled from 'T'

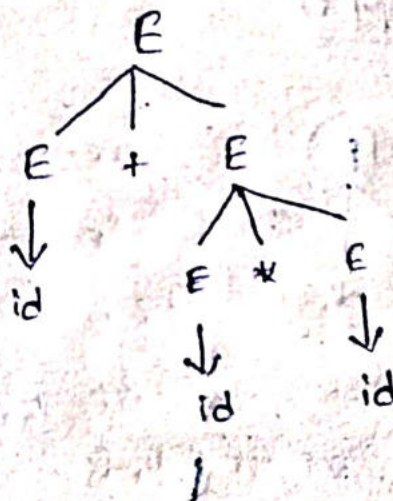
Step-4;

Every internal node has labeled from 'V'

Step-5;

If an internal node 'A' has 'n' descendent with label $x_1, x_2, \dots, x_n$ then there must exist production rule $A \rightarrow x_1, x_2, \dots$

ex:



Yield of Parse Tree:

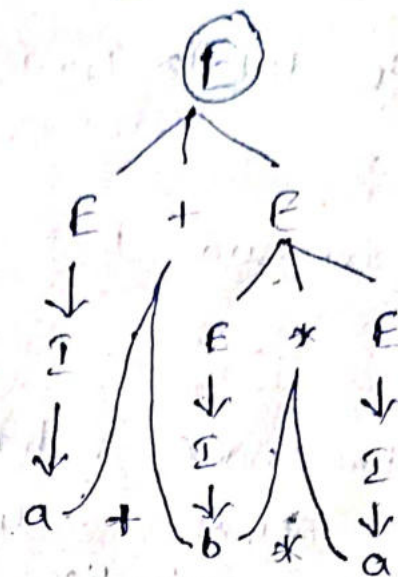
The Yield of tree is string of terminal symbols obtained by reading the leaf nodes of the tree from left to right without considering the null symbol.

* Construct the derivation tree for the string $a + b * a$ using the grammar

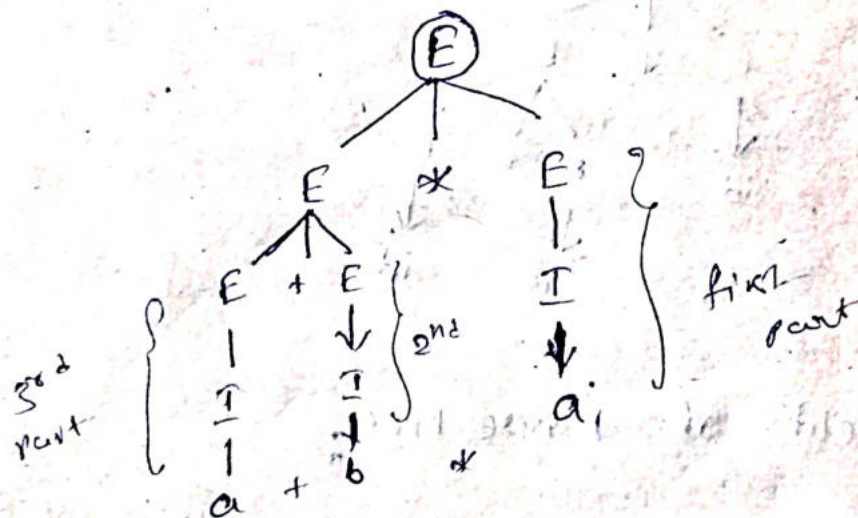
$E \rightarrow I, E \rightarrow E + E, E \rightarrow E * E, E \rightarrow (E),$

$I \rightarrow a, I \rightarrow b.$

LMD



RMD



* Derive baaba using LMD & RMD

$S \rightarrow AAA/AA$

$A \rightarrow AA/aA/Ab/a/b$

LMD

$S \rightarrow AAA$

$S \rightarrow bAAA$

$S \rightarrow baAA$

$S \rightarrow baaAA$

$S \rightarrow baabaA$

$S \rightarrow AAA$

$A \rightarrow Ab$

$A \rightarrow aA$

$A \rightarrow aA$

$A \rightarrow b$

$A \rightarrow a$

RMD

$S \rightarrow AAA$

$S \rightarrow AA\cancel{A}b \quad (A \rightarrow \cancel{A}b)$

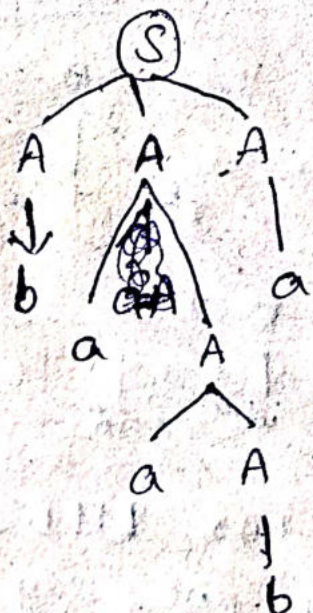
$S \rightarrow AAba \quad (A \rightarrow Ab)$

$S \rightarrow AAAb a \quad (A \rightarrow AA)$

$S \rightarrow AAaba \quad (A \rightarrow a)$

$S \rightarrow Aaaba \quad (A \rightarrow a)$

LMD?



RMD!

(S)

Ambiguous Grammar :-

A Grammar G is said to be ambiguous if there exists 2 (or) more left derivations (or) 2 (or) more right derivations for the same string. is known as "Ambiguous Grammar".

*

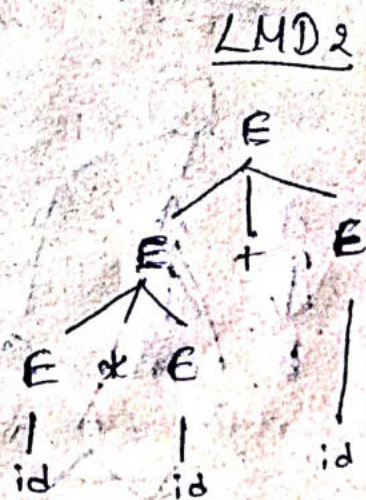
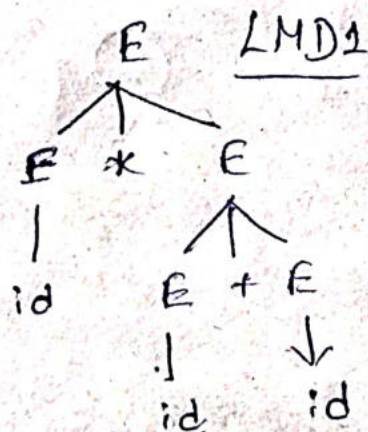
Consider the following grammar,

$E \rightarrow E + E, E \rightarrow E * E, E \rightarrow id$ & verify that the given grammar is ambiguous or not.

Sol

Consider the string $id * id + id$.

The LMD for the string is



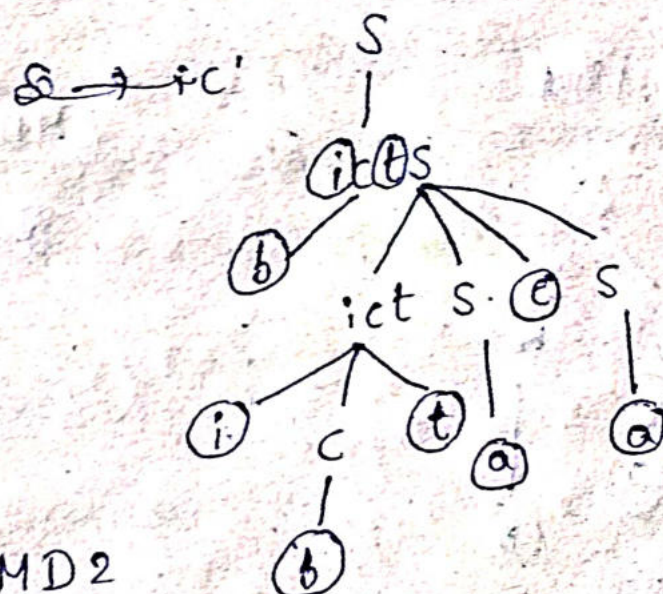
As we have two LMD's for same string then, we say that the given grammar is ambiguous.

* Verify that the following grammar is ambiguous or not. string is ibtibtaea

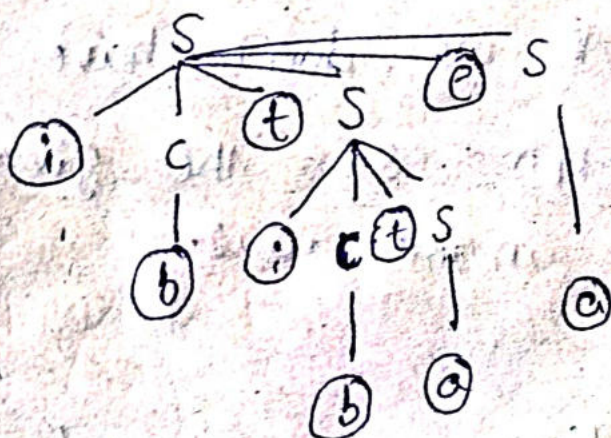
$S \rightarrow ict / ictses / a$

$c \rightarrow b$

LMD 1



LMD 2



As we have two LMDs for same string then the given grammar is ambiguous.

* Check whether the grammar is ambiguous (or) not.

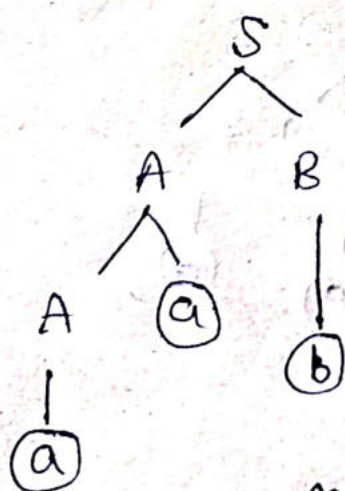
$$S \rightarrow AB$$

$$A \rightarrow Aa/a$$

$$B \rightarrow b$$

String = aab

LMD 1



~~LMD 2~~

8

As we ~~don't~~ have only one LMD, so the given grammar is unambiguous.

02/04/24

Minimisation of Context free grammar (or) Simplification of Context free grammar

* Minimisation of CFG is done in various steps.

- ① Elimination of ϵ -productions
- ② Elimination of unit-productions
- ③ Elimination of useless symbols.

① Elimination of ϵ -productions

① Construct V_n [set of all nullable variables].

② for each production

① if the grammar contains $A \rightarrow \epsilon$ then replace A with ϵ in remaining productions.

② Remove $A \rightarrow \epsilon$ from the grammar.

Remove ϵ from following grammar.

$$A \rightarrow 0B1/1B1$$

$$B \rightarrow 0B/1B/\epsilon$$

Since there is a production $B \rightarrow \epsilon$ we replace B with ϵ in all productions.

$$A \rightarrow 0B1/1B1$$

Substituting $B \rightarrow \epsilon$

$$A \rightarrow 01/11/0B1/1B1$$

$$B \rightarrow 0\epsilon/1\epsilon/\epsilon$$

$$B \rightarrow 0B/1B/\epsilon$$

← substituting $B \rightarrow \epsilon$

$$B \rightarrow 00/01/1B/1$$

Now the grammar is,

$$A \rightarrow 01/11/0B1/1B1$$

$$B \rightarrow 00/01/1B/1$$

*

Replacing $\gamma \rightarrow \epsilon$

$$S \rightarrow XYX / YX / XY / Y / XX / X / \epsilon$$

$$X \rightarrow 0X / 0$$

$$Y \rightarrow 1Y / 1$$

sol)

As, there are no S productions, we simply eliminate $S \rightarrow \epsilon$ and the grammar is

$$S \rightarrow XYX / YX / XY / Y / XX / X$$

$$X \rightarrow 0X / 0$$

$$Y \rightarrow 1Y / 1$$

$$B \rightarrow aBb / ab$$

*

Remove ϵ -production from the grammar.

$$S \rightarrow aA, A \rightarrow BB, B \rightarrow aBb / \epsilon$$

sol)

$$\textcircled{1} \quad \underline{B \rightarrow \epsilon}$$

$$S \rightarrow aA$$

$$A \rightarrow BB / B / \epsilon$$

$$B \rightarrow aBb / ab$$

$$\textcircled{2} \quad \underline{A \rightarrow \epsilon}$$

$$S \rightarrow aA / a$$

$$A \rightarrow BB / B$$

$$B \rightarrow aBb / ab$$

Remove the ϵ -productions from the grammar

$$S \rightarrow XYX$$

$$X \rightarrow 0X/\epsilon$$

$$Y \rightarrow 1Y/\epsilon$$

① $X \rightarrow \epsilon$

$$S \rightarrow XYX / XY / YX / Y$$

$$X \rightarrow 0X / 0$$

$$Y \rightarrow 1Y / \epsilon$$

② $Y \rightarrow \epsilon$

$$S \rightarrow XYX / XY / YX / XX / X / \epsilon$$

$$X \rightarrow 0X / 0$$

$$Y \rightarrow 1Y / 1$$

As there are no S production, we simply eliminate $S \rightarrow \epsilon$ and the grammar is

$$S \rightarrow XYX / XY / YX / XX / X$$

$$X \rightarrow 0X / 0$$

$$Y \rightarrow 1Y / 1$$

② Elimination of unit productions

* A product is called "unit production" if it is of the form $A \rightarrow B$.

Procedure to eliminate unit production

① for each pair of non-terminals

A and B such that there is a production $A \rightarrow B$ and non-unit productions from B are $B \rightarrow X_1/X_2 \dots X_n$ where $X_i \in (V \cup T)^*$ are string of terminals and non-terminals then create the new production A as $A \rightarrow X_1/X_2 \dots X_n$.

② Do the same for all such pairs A and B simultaneously to remove all the unit productions.

Eliminate unit production

$$S \rightarrow Aa/B$$

$$B \rightarrow A/bb$$

$$A \rightarrow a/bcB$$

① $S \rightarrow B$

$$S \rightarrow Aa/A/bb$$

$$B \rightarrow A/bb$$

$$A \rightarrow a/bcB$$

2011

$$\textcircled{2} \quad \underline{B \rightarrow A}$$

$$S \rightarrow Aa/A/bb$$

$$B \rightarrow a/bcB/bb$$

$$A \rightarrow a/bcB$$

$$\textcircled{3} \quad \underline{S \rightarrow A}$$

$$S \rightarrow Aa/a/bcB/bb$$

$$B \rightarrow a/bcB/bb$$

$$A \rightarrow a/bcB$$

2)

Eliminate unit production

$$S \rightarrow A/bb$$

$$A \rightarrow B/b$$

$$B \rightarrow S/a$$

sol)

$$\textcircled{1} \quad \underline{S \rightarrow A}$$

$$S \rightarrow B/b/bb$$

$$A \rightarrow B/b$$

$$B \rightarrow S/a$$

$$\textcircled{2} \quad \underline{A \rightarrow B}$$

$$S \rightarrow B/b/bb$$

$$A \rightarrow S/a/b$$

$$B \rightarrow S/a$$

$$\textcircled{3} \quad \underline{B \rightarrow S}$$

$$S \rightarrow B/b/bb$$

$$A \rightarrow S/a/b$$

$$B \rightarrow B/b/bb/a$$

$$\textcircled{4} \quad \underline{B \rightarrow B}$$

$$B \rightarrow b/bb/a$$

$$S \rightarrow B/b/bb$$

$$A \rightarrow S/a/b$$

$$\textcircled{5} \quad \underline{S \rightarrow B}$$

$$S \rightarrow a/b/bb$$

$$A \rightarrow S/a/b$$

$$B \rightarrow a/b/bb$$

$$\textcircled{6} \quad \underline{A \rightarrow S}$$

$$S \rightarrow a/b/bb$$

$$A \rightarrow a/b/bb$$

$$B \rightarrow a/b/bb$$

3)

Eliminate unit productions,

$$A \rightarrow aBB/b$$

$$B \rightarrow S/aab$$

$$S \rightarrow Cab/\epsilon$$

$$C \rightarrow S$$

$$\textcircled{1} \quad \cancel{A \rightarrow B}$$

$$\cancel{A \rightarrow aBB / S / aab}$$

$$\cancel{B \rightarrow S / aab}$$

$$\textcircled{1} \quad \underline{S \rightarrow E}$$

$$\textcircled{2} \quad \underline{C \rightarrow S}$$

$$A \rightarrow aBB / B$$

$$A \rightarrow aBB / B$$

$$B \rightarrow aab$$

$$B \rightarrow aab$$

$$S \rightarrow Cab$$

$$S \rightarrow Cab$$

$$C \rightarrow S$$

$$C \rightarrow Cab$$

$$\textcircled{3} \quad \underline{A \rightarrow B}$$

$$A \rightarrow aBB / aab$$

$$B \rightarrow aab$$

$$S \rightarrow Cab$$

$$C \rightarrow Cab$$

3/04/24

③ Removing useless symbols :-

* A Symbol is useless, if it is not deriving any terminal symbol

(or)

If it is not reachable to start symbol.

* To check the variables are reachable from start symbol, we construct dependency graph.

* By using dependency graph identify non-reachable variables and they are useless variables.

* Remove these non-reachable variables and production rules from the grammar, verify whether all the variables are verifying deriving terminal symbols or not.

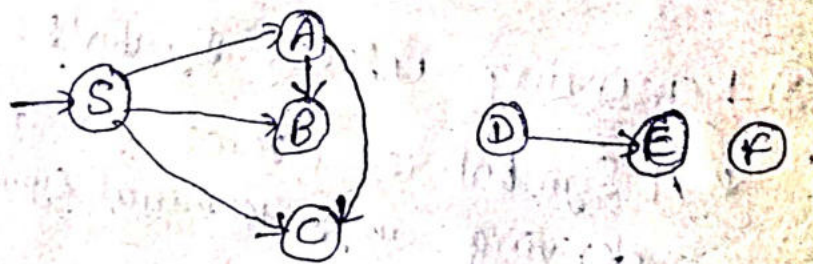
* If any variable not deriving terminal symbols then they are useless variable.

1) Remove useless symbols from grammar

$S \rightarrow ABa/BC$, $A \rightarrow ac/Bcc$, $C \rightarrow a$
 $B \rightarrow bcc$, $D \rightarrow E$, $E \rightarrow d$, $F \rightarrow e$

sol)

Construct dependency graph to identify non-reachable variables



from the above dependency graph,
 D, E, F are not reachable, so they are useless variables.

By removing the useless variables from grammar then we get,

$S \rightarrow ABa/BC$
 $A \rightarrow ac/Bcc$
 $B \rightarrow bcc$
 $C \rightarrow a$

Verifying that all the variables are generating terminal symbols or not

$$\textcircled{1} S \rightarrow ABa$$

$$S \rightarrow aCBa$$

$$S \rightarrow aaBa$$

$$S \rightarrow aabcca$$

$$\textcircled{2} A \rightarrow aC$$

$$A \rightarrow aa$$

$$\textcircled{3} B \rightarrow bcc$$

$$\textcircled{4} C \rightarrow a$$

From the above grammar,

all the variables are deriving terminal symbols.

Hence,

there are no useless symbols in the grammar.

The final grammar is,

$$S \rightarrow ABa/BC$$

$$A \rightarrow aC/Bcc$$

$$B \rightarrow bcc$$

$$C \rightarrow a$$

2)

Remove useless symbols from grammar

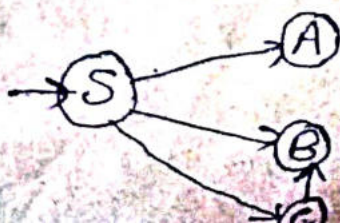
$$S \rightarrow AB/CA$$

$$S \rightarrow BC$$

$$A \rightarrow a$$

$$C \rightarrow aB/b$$

dependency graph



from the dependency graph,
there are no useless symbols.

~~Verifying that all the variables are
generating terminal symbols (or) not.~~

① $S \rightarrow$

Here, 'B' variable not deriving, so
we remove 'B'. then the grammar
we get,

$$S \rightarrow CA$$
$$A \rightarrow a$$
$$C \rightarrow b$$

As all the above variables are
generating ~~non~~ terminal symbols
then final grammar is

$$S \rightarrow CA$$
$$A \rightarrow a$$
$$C \rightarrow b$$

Remove useless symbols

$$S \rightarrow aAa, A \rightarrow bBb$$

$$B \rightarrow ab, C \rightarrow ab$$

dependency graph,



from the dependency graph, 'C' is useless symbol.

By removing the useless variable from graph, we get grammar is,

$$S \rightarrow aAa$$

$$A \rightarrow bBb$$

$$B \rightarrow ab$$

verifying that all variables are generating terminal symbols or not

① $S \rightarrow aAa$

$$S \rightarrow abBba$$

$$S \rightarrow ababBa$$

$$S \rightarrow abababa$$

② $A \rightarrow bBb$

$$A \rightarrow babB$$

$$A \rightarrow babab$$

③ $B \rightarrow ab$

As all the above variables are generating terminal symbols then final grammar is,

$$S \rightarrow aAa$$

$$A \rightarrow bBb$$

4) Remove useless symbol from grammar

$$S \rightarrow aS/A/BC$$

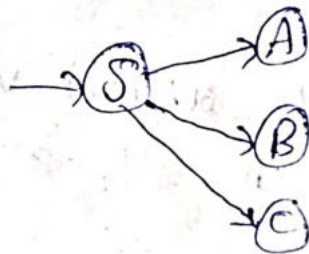
$$A \rightarrow a$$

$$B \rightarrow aa$$

$$C \rightarrow aCb$$

sol)

dependency graph is



from the dependency graph,
there are no useless symbols.

Verifying above variables whether
they are generating terminals or not

$$A \text{ ① } S \rightarrow A$$

$$\text{② } A \rightarrow a$$

$$S \rightarrow a$$

$$\text{③ } B \rightarrow aa$$

④

$$C \rightarrow aCb$$

$$C \rightarrow aacbb$$

here,

'C' is not generating
terminal symbols.

so, 'C' is a useless variable.

After removing useless variable then
final grammar is

$$S \rightarrow aS/A$$

Chomsky Normal form (CNF):-

* A CFG is said to be in CNF, if and only if the grammar does not contain ϵ -productions, unit productions, useless symbols and all the production of the grammar is of the form

$$A \rightarrow BC$$

$$A \rightarrow a$$

Conversion of CFG to CNF

- 1) If the start symbol 'S' occurs on some right side, create a new start symbol 'S'' and a new production $S' \rightarrow S$.
- 2) Remove null productions.
- 3) Remove unit productions and useless symbols.
- 4) Replace each production

$$A \rightarrow B_1, B_2, B_3, \dots, B_n \text{ where } n > 2$$

with $A \rightarrow B_1 C$ where $C \rightarrow B_2, B_3, \dots, B_n$

Repeat this step for all productions having 2 or more symbols on right side.

- 5) If the rightside of any production is in the form, $A \rightarrow aB$ where 'a' is terminal, A, B are non-terminals then the production is replaced by $A \rightarrow XB$, $X \rightarrow a$. Repeat this step.

15/04/24

1) Simplify the following grammar

$$S \rightarrow A / \epsilon C 1$$

$$A \rightarrow B / 01 / 10$$

$$C \rightarrow CD / \epsilon$$

Ans

Step-1 :- Eliminate ϵ productions

$$C \rightarrow \epsilon$$

$$S \rightarrow A / \epsilon C 1 / 01$$

$$A \rightarrow B / 01 / 10$$

$$C \rightarrow CD / D$$

Step-2 :- Eliminate unit productions

① $S \rightarrow A$

$$S \rightarrow \epsilon C 1 / 01 / B / 10$$

$$A \rightarrow B / 01 / 10$$

$$C \rightarrow CD / D$$

② $A \rightarrow B$

As we ~~simply~~ don't have B's production
then eliminate $A \rightarrow B$

$$S \rightarrow \epsilon C 1 / 01 / B / 10$$

$$A \rightarrow 01 / 10$$

$$C \rightarrow CD / D$$

$$\textcircled{3} \quad \underline{C \rightarrow D}$$

$$S \rightarrow 0C1/01/B/10$$

$$A \rightarrow 01/10$$

$$C \rightarrow CD$$

$$\textcircled{4} \quad \underline{S \rightarrow B}$$

$$S \rightarrow 0C1/01/10$$

$$A \rightarrow 01/10$$

$$C \rightarrow CD$$

Step-3 :- Eliminate useless symbols.



$$S \rightarrow 0C1/01/10$$

$$C \rightarrow CD$$

here, 'A' is not reaching from 'S'.

⑦ Here, 'C' is not generating terminals
So, we remove 'C' production.

∴ The grammar is

~~$$S \rightarrow 0C1/01/10$$~~

$$S \rightarrow 01/10$$

a)

Simplify the grammar

$$S \rightarrow aA / aBB$$

$$A \rightarrow aAA / \epsilon$$

$$B \rightarrow bB / b\epsilon$$

$$C \rightarrow B$$

b)

Step-1 :- Eliminate ϵ productions

$$\underline{A \rightarrow \epsilon}$$

$$S \rightarrow aA / a / aBB$$

$$A \rightarrow aAA / aA / a$$

$$B \rightarrow bB / b\epsilon$$

$$C \rightarrow B$$

Step-2 :- Remove unit productions

$$\underline{C \rightarrow B}$$

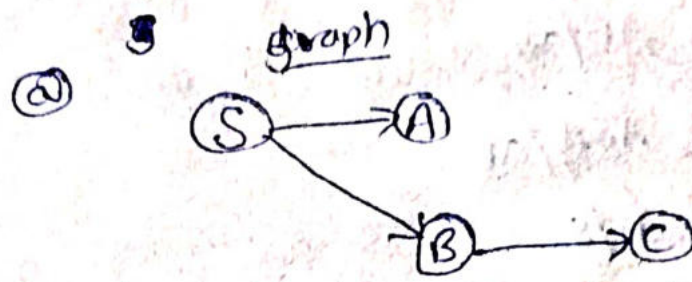
$$S \rightarrow aA / a / aBB$$

$$A \rightarrow aAA / aA / a$$

$$B \rightarrow bB / b\epsilon$$

$$C \rightarrow bB / b\epsilon$$

Step-3:- Remove useless symbols.



(b) verifying

$$\begin{array}{l|l|l}
 S \rightarrow aA & A \rightarrow a & B \rightarrow bB \\
 S \rightarrow aa & & B \rightarrow bbbC \\
 & & C \rightarrow bB \\
 & & C \rightarrow bbb
 \end{array}$$

here,

B, C, A does not generate terminals.

$\therefore$ The final grammar is

$$\begin{array}{l}
 S \rightarrow aA/a \\
 A \rightarrow aAA/aA/a
 \end{array}$$

16/04/21

Conversion of CFG to CNF

1)

$$S \rightarrow AB/aB$$

$$S \rightarrow ASA/aB$$

$$A \rightarrow B/s$$

$$B \rightarrow b/\epsilon$$

Sol)

Step-1 :-

As start symbol 'S' is present in right hand side. So, we introduce new start symbol, 'S'

i.e

$$S' \rightarrow S$$

$$S \rightarrow ASA/aB$$

$$A \rightarrow B/s$$

$$B \rightarrow b/\epsilon$$

Step-2 :-

Eliminate ϵ productions

$$\underline{B \rightarrow \epsilon}$$

$$S' \rightarrow S$$

$$S \rightarrow \cancel{ASB}/AS$$

$$S \rightarrow ASA/aB/a$$

$$A \rightarrow \epsilon/s/B$$

$$B \rightarrow b$$

$$\underline{A \rightarrow \epsilon}$$

$$S' \rightarrow S$$

$$S \rightarrow ASA/aB/a/AS/SA/s$$

$$A \rightarrow B/s$$

$$B \rightarrow b$$

Step-3:- Eliminate unit productions.

① $S' \rightarrow S$

$$S' \rightarrow ASA/aB/a/AS/SA$$

$$S \rightarrow ASA/aB/a/AS/SA$$

$$A \rightarrow B/S$$

$$B \rightarrow b$$

② $A \rightarrow B$

$$S' \rightarrow ASA/aB/a/AS/SA$$

$$S \rightarrow ASA/aB/a/AS/SA$$

$$A \rightarrow b/S$$

$$B \rightarrow b$$

③ $A \rightarrow S$

$$S' \rightarrow ASA/aB/a/AS/SA$$

$$S \rightarrow ASA/aB/a/AS/SA$$

$$A \rightarrow b/ASA/aB/a/AS/SA$$

$$B \rightarrow b$$

Step-4:- ~~if~~ productions containing more than 2 non termi
on right

$$S' \rightarrow ASA$$

$$S \rightarrow ASA$$

$$A \rightarrow ASA$$

Let, $X = SA$ and add production
 $X \rightarrow SA$

i.e.

$$S' \rightarrow AX/a/AS/SA/aB$$

$$S \rightarrow AX/a/AS/SA/aB$$

$$A \rightarrow b/AX/a/AS/SA/aB$$

$$B \rightarrow b$$

Step - 5 :-

Let, $a = Y$ add production

$$Y \rightarrow a$$

$$S' \rightarrow AX/a/SA/AS/YB$$

$$S \rightarrow AX/a/SA/AS/YB$$

$$A \rightarrow b/AX/a/SA/AS/YB$$

$$B \rightarrow b$$

2)

$$S \rightarrow AB/aB$$

$$A \rightarrow aab/\epsilon$$

$$B \rightarrow bA$$

sol)

Step - 1 :- As S' is not in right side.

then S' is the start symbol.

$$S \rightarrow AB/aB$$

$$A \rightarrow aab/\epsilon$$

$$B \rightarrow bA$$

Q

Step-2 :- Eliminate ϵ productions

$$\underline{A \rightarrow \epsilon}$$

$$S \rightarrow AB/B/aB$$

$$A \rightarrow aab$$

$$B \rightarrow bbA/bb$$

Step-3 :- Eliminate unit productions

$$\underline{S \rightarrow B}$$

$$S \rightarrow AB/bbA/bb/aB$$

$$A \rightarrow aab$$

$$B \rightarrow bbA/bb$$

Step-4 :-

As there are no productions which contains more than 2 non-terminal

Step-5 :-

$$S \rightarrow bbA$$

$$A \rightarrow aab$$

$$B \rightarrow bbA$$

Let, $a = x$, $b = y$ and add
 $x \rightarrow a$, $y \rightarrow b$

$$S \rightarrow YYA/AB/YY/xB$$

$$A \rightarrow axxy$$

Let,

$Xb = R$, $YA = T$ and add

$$R \rightarrow Xb$$

$$T \rightarrow YA$$

$$S \rightarrow AB/YT/YX/XB$$

$$A \rightarrow XR$$

$$B \rightarrow YT/YX$$

$$X \rightarrow a$$

$$Y \rightarrow b$$

$$R \rightarrow Xb$$

$$T \rightarrow YA$$

19/04/24

Greibach Normal form (GNF)

A grammar is said to be in GNF, if all the productions are of the form

$$A \rightarrow a, A \rightarrow aC_1C_2C_3 \dots C_n$$

where

$A, C_1, C_2 \dots C_n$ are Non-Terminals
 a is terminal.

Conversion CF GR to GNF

Steps :-

- ① Convert the grammar into CNF, if it is not in CNF.
- ② Rename the variables into some A_i 's as $A_1, A_2, \dots$ in the increasing order of i .
- ③ For each production of the form,
 $A_i \rightarrow A_j \alpha$ then apply the following.
 - (a) If $i < j$, leave the production as it is.
 - (b) If $i = j$, apply left recursion removal.
 - (c) If $i > j$, apply substitution rule on R.H.S.

Substitution Rule :-

Let $G = (V, T, P, S)$ be a Context free grammar and $A \rightarrow \alpha, B \alpha_2$ be a production in P and $B \rightarrow \beta_1 / \beta_2 / \dots / \beta_n$ is also a production in P .

Let $G' = (V, T, P', S)$ can be constructed by removing A 's production from P and adding α to P' by substituting the productions in B we get,

$$A \rightarrow \alpha, \beta_1 \alpha_2 / \alpha, \beta_2 \alpha_2 / \alpha, \beta_3 \alpha_2 / \dots / \alpha, \beta_n \alpha_2$$

Elimination of left recursion :-

* In a grammar, productions of the form $A \rightarrow A \alpha / \beta$ is called "left recursion".

* To eliminate left recursion, we will write the grammar by introducing new variable

$$Z \text{ as } A \rightarrow \beta Z / \beta, Z \rightarrow \alpha Z / \alpha$$

1) Convert the given grammar into GNF.

$$S \rightarrow CA/BB$$

$$B \rightarrow b/SB$$

$$C \rightarrow b$$

$$A \rightarrow a$$

Ans)

Step-1:

The given grammar is already in CNF

Step-2:

Rename the variables.

$$A_1 \rightarrow A_2 A_3 / A_4 A_4$$

$$A_4 \rightarrow b / A_1 A_4$$

$$A_2 \rightarrow b$$

$$A_3 \rightarrow a$$

Step-3

$$A_1 \rightarrow A_2 A_3 \text{ (no change) } : (i < j) \text{ i.e. } (1 < 2)$$

$$A_1 \rightarrow A_4 A_4 \text{ (no change) } :$$

$$A_4 \rightarrow b \text{ (in GNF)}$$

$$A_4 \rightarrow A_1 A_4 \text{ (} i > j \text{ i.e. } 4 > 1)$$

Apply substitution rule, then we get
i.e. A_i productions are substituting.

$$A_4 \rightarrow A_2 A_3 A_4 / A_4 A_4 A_4$$

$$A_4 \rightarrow A_2 A_3 A_4 \text{ (} i > j \text{ (} 4 > 2))$$

$$A_4 \rightarrow b A_3 A_4 \text{ (} A_2 \rightarrow b \text{) by substitution rule}$$

$$A_4 \rightarrow b A_3 A_4 / A_4 A_4 A_4 / b$$

$$A_4 \rightarrow A_4 A_4 A_4 \text{ (} i = j \text{ (} 4 = 4)) \text{ left}$$

whereas it is in left recursion

we have to eliminate left recursion
by introducing new non-terminal 'Z'.

$$A_4 \rightarrow \underline{b A_4 A_4 A_4} / \underline{b A_3 A_4} / \underline{b}$$

$\downarrow$
 α

$\downarrow$
 β

$\downarrow$
 β

$$A \rightarrow A\alpha/\beta$$

$\downarrow$ conversion

$$A \rightarrow \beta Z/\beta$$

$$Z \rightarrow \alpha Z/\alpha$$

$$A_4 \rightarrow b A_3 A_4 Z / b A_3 A_4 / b Z / b$$

$$Z \rightarrow A_4 A_4 Z / A_4 A_4$$

∴ The productions are,

$$A_1 \rightarrow A_2 A_3 / A_4 A_4$$

$$A_4 \rightarrow b A_3 A_4 Z / b A_3 A_4 / b Z / b$$

$$A_2 \rightarrow b$$

$$A_3 \rightarrow a$$

$$Z \rightarrow A_4 A_4 Z / A_4 A_4$$

The productions A_1 and Z are not in GNF.

we apply substitution,
final productions are

$$A_1 \rightarrow b A_3 / b A_3 A_4 Z A_4 / b A_3 A_4 A_4 / b Z A_4 / b A_4$$

$$Z \rightarrow b A_3 A_4 Z A_4 Z / b A_3 A_4 Z A_4 / b A_3 A_4 A_4 Z /$$

$$b A_3 A_4 A_4 / b Z A_4 Z / b Z A_4 / b A_4 Z / b A_4$$

$$A_4 \rightarrow b A_3 A_4 Z / b A_3 A_4 / b Z / b$$

$$A_2 \rightarrow b$$

$$A_3 \rightarrow a$$

•

2) Convert the grammar into GNF

$$S \rightarrow AA/a$$

$$A \rightarrow SS/b$$

A) Step-1 :-

The given grammar is in CNF.

Step-2 :- Rename the variables.

$$A_1 \rightarrow A_2 A_2 / a$$

$$A_2 \rightarrow A_1 A_1 / b$$

Step-3 :-

$$A_1 \rightarrow A_2 A_2 \quad (i < j) \quad \text{no change}$$

$$A_1 \rightarrow a \quad (\text{already in GNF})$$

$$A_2 \rightarrow A_1 A_1 \quad (i > j)$$

Apply Substitution rule

$$A_2 \rightarrow A_2 A_2 A_1 / a A_1$$

$$\Rightarrow A_2 \rightarrow A_2 \underbrace{A_2 A_1}_{\alpha} / \underbrace{a A_1}_{\beta}$$

$$A_2 \rightarrow A_2 A_2 A_1 \quad (i = j) \quad (\text{left recursion})$$

$$A_2 \rightarrow a A_1 z / b z / a A_1 / b$$

$$z \rightarrow A_2 A_1 z / A_2 A_1$$

The productions are

$$A_1 \rightarrow A_2 A_2 / a$$

$$A_2 \rightarrow a A_1 z / b z / a A_1 / b$$

$$Z \rightarrow A_2 A_1 z / A_2 A_1$$

Here,

A_1, Z productions are not in GNF.

we apply substitution,

$$A_1 \rightarrow a / a A_1 z A_2 / b z A_2 / a A_1 A_2 / b A_2 / \cancel{a A_1 z a / b z a / a A_1 a / b a}$$

$$Z \rightarrow a A_1 \cancel{A_2 z} / a A_1 \cancel{A_1} / b z A_1 z / b z A_1 / a A_1 A_1 z / a A_1 A_1 / b A_1 z / b A_1$$

$\therefore$ final productions are

$$A_1 \rightarrow a A_1 z A_2 / b A_2 z / a A_1 A_2 / b A_2 / a$$

$$A_2 \rightarrow a A_1 z / b z / a A_1 / b$$

$$\cancel{Z \rightarrow A_2 A_1 z / A_2 A_1}$$

$$Z \rightarrow a A_1 \cancel{A_2 z} / a A_1 \cancel{A_1} / b z A_1 z / b z A_1 / a A_1 A_1 z / a A_1 A_1 / b A_1 z / b A_1$$

3)

Convert the grammar into GNF

$$A_1 \rightarrow A_2 A_3$$

$$A_2 \rightarrow A_3 A_1 / b$$

$$A_3 \rightarrow A_1 A_2 / a$$

A)

The given grammar is already in CNF.

$$A_1 \rightarrow A_2 A_3 \quad (\text{no change})$$

$$A_2 \rightarrow A_3 A_1 \quad (\text{no change})$$

$$A_2 \rightarrow b$$

$$A_3 \rightarrow A_1 A_2 \quad (i \geq j)$$

substitution rule.

$$A_3 \rightarrow A_2 A_3 A_2 \quad (i > j)$$

$$A_3 \rightarrow A_3 A_1 A_3 A_2 / b A_3 A_2$$

$$A_3 \rightarrow A_3 A_1 A_3 A_2 \quad (i = j)$$

$$A_3 \rightarrow \underbrace{A_3 A_1 A_3 A_2}_{\alpha} / \underbrace{b A_3 A_2}_{\beta} / a$$

$$A_3 \rightarrow b A_3 A_2 Z / b A_3 A_2 / a Z / a$$

$$Z \rightarrow A_1 A_3 A_2 Z / A_1 A_3 A_2$$

The productions are

$$A_1 \rightarrow A_2 A_3$$

$$A_2 \rightarrow A_3 A_1 / b$$

$$A_3 \rightarrow b A_3 A_2 z / b A_3 A_2 / a z / a$$

$$z \rightarrow A_1 A_3 A_2 z / A_1 A_3 A_2$$

here,

A_1, A_2, z are not in GNF.

final productions are

$$A_2 \rightarrow A_3 b A_3 A_2 z A_1 / b A_3 A_2 A_1 / a z A_1 / a A_1 / b$$

$$A_1 \rightarrow b A_3 A_2 z A_1 A_3 / b A_3 A_2 A_1 A_3 / a z A_1 A_3 / a A_1 A_3 / b A_3$$

$$z \rightarrow b A_3 A_2 z A_1 A_3 A_3 A_2 z / b A_3 A_2 A_1 A_3 A_3 A_2 z /$$

$$a z A_1 A_3 A_3 A_2 z / a A_1 A_3 A_3 A_2 z / b A_3 A_3 A_2 z /$$

$$b A_3 A_2 z A_1 A_3 A_3 A_2 / b A_3 A_2 A_1 A_3 A_3 A_2 / a z A_1 A_3 A_3 A_2 /$$

$$a A_1 A_3 A_3 A_2 / b A_3 A_3 A_2$$

Pumping Lemma for Context-free Languages:

Let L be any context free language.
then there exists a positive integer
 n such that every string $w \in L$
with $|w| \geq n$ can be decomposed as
 $w = uvxyz$ such that

① for all $i \geq 0$ the string

$$uv^ixy^iz \in L$$

② $|vxy| \leq n$

③ $|vy| \geq 1$

Procedure for proving a language is not CFL by using pumping lemma.

1) Assume that the given language is Context free language.

2) Let n be a +ve integer.

3) Pick any string w from the language such that $|w| \geq n$.

4) Decompose the string w into u, v, x, y, z such that,

① $\forall i \geq 0$ the string $uv^ixy^iz \in L$

② $|vxy| \leq n$

③ $|vy| \geq 1$

for any string $uv^ixy^iz \notin L$ then it violates the pumping lemma and hence the given language is not context free

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

Assume that,

Given L is Context free language.

Let $n = 3$

$$= a^3 b^3 c^3$$

$$w = aaabbbccc$$

$$|w| \geq n \text{ i.e. } 9 \geq 3$$

Case (i) :-

u and v have same symbols.

$$w = \underbrace{aa}_u \underbrace{bb}_v \underbrace{cc}_x \underbrace{cc}_y \underbrace{cc}_z$$

consider, $i = 2$

$$= aa(a)^2 bbb(c)^2 cc$$

$$= aaaaabbbcccc \notin L$$

$\therefore$ Given language is not CFL.

prove that $L = \{ww \mid w \in \{0,1\}^*\}$ is not context free.

Assume that,

Given L is Context free language.

let $n = 3$

$$w = 001001 \quad (\because w = 001) \\ \in w \in \{0,1\}^*$$

$$|w| \geq n \text{ (i.e. } 6 \geq 3)$$

^{does not}
 V and y have same symbol length

$$w = \underbrace{001001}_{u \quad v \quad x \quad y \quad z} \quad w = \underbrace{001}_{u} \underbrace{001}_{v} \underbrace{001}_{x} \underbrace{001}_{y} \underbrace{001}_{z}$$

consider $i=2$

$$= 0(0)^2 1 (00)^2 1$$

$$= 0001 00001 \notin L$$

$\therefore$ Given language is not CFL

3) Show that $L = \{a^n / n \text{ is prime number}\}$ is not context free language

sol)

Assume that Given language is CFL

$$\text{Let } n = 7$$

$$w = aaaaaaa$$

$$|w| \geq 7 \quad (7) \quad (7 \geq 7)$$

case (i) V and y have same symbols

$$w = \underbrace{a}_{u} \underbrace{a}_{v} \underbrace{a}_{x} \underbrace{a}_{y} \underbrace{a}_{z}$$

Consider, $i=2$

$$= aa(a)^2 aa(a)^2 a$$

$$= aaaaaaa \notin L \quad \left(9 \text{ is not prime number} \right)$$

$\therefore$ Given language is not CFL.

Closure properties of Context free languages

- 1) CFL's are closed under union.
i.e. If L_1 and L_2 are two context free languages then $L_1 \cup L_2$ is also CFL.
- 2) CFL's are closed under concatenation.
i.e. If L_1 and L_2 are two CFL's then $L_1 \cdot L_2$ is also CFL.
- 3) CFL's are closed under Kleen's closure.
i.e. If L_1 is a CFL then L_1^+ is also CFL.
- 4) CFL's are not closed under intersection.
i.e. If L_1 and L_2 are two CFL's then $L_1 \cap L_2$ are not CFL's.
- 5) CFL's are not closed under complementation.
i.e. If L_1 is a CFL then $\bar{L}_1$ is not CFL.
- 6) CFL's are closed under substitution.
- 7) CFL's are closed under Homomorphism, and Inverse Homomorphism.
- 8) CFL's are closed under reversal.

10/4/24

4. Push down Automata

Introduction:-

- * A PDA is a way to implement Context free grammar as we design finite automata for regular grammar.
- * PDA is more powerful than finite state machine (FSM). FSM has a very limited memory but a PDA has more memory.

$$\boxed{\text{PDA} = \text{FSM} + \text{Stack}}$$

- * A stack is a way, we arrange elements one on the top of the stack.

- * A stack does two basic operations

① push

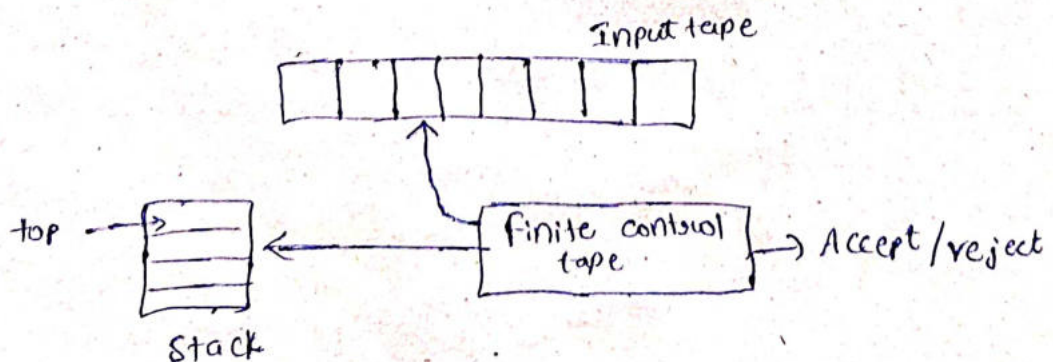
② pop.

↓
used to add
new element
on the top of
the stack.

↓
used to remove the top
element from the stack.

Basic Model of PDA :-

- * PDA has 3 components input tape, finite control unit, stack.



* A stack with infinite size.

* It has unlimited ~~storage~~ amount of storage space.

* Used to store data & remove the data temporarily.

Formal definition of PDA :-

Mathematically, a PDA is defined with 7 tuples like

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

where,

Q = ^{finite} Set of input states

Σ = finite set of input symbols.

Γ = finite no. of set of stack symbols.

q_0 = Initial state

z_0 = ^{the symbol} on top of the stack

F = Set of final states.

δ takes 3 tuples and is transition function which is defined as

$$\delta: (Q \times (\Sigma \cup \epsilon)) \times \Gamma \rightarrow (Q \times \Gamma)$$

where, i.e.

$$\delta(q, a, x)$$

where

(i) q is a state in Q

(ii) a is either an input symbol in Σ or a belongs to ϵ .

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

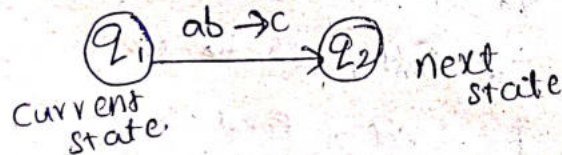
* The output of δ is a finite set or pairs like (p, γ) where p is a new state, γ is a set of stack symbols that replaces x at the top of the stack.

ex:-

- (i) If $\gamma = \epsilon$, then stack is popped.
- (ii) If $\gamma = x$, then the stack is unchanged.
- (iii) If $\gamma = YZ$, then x is replaced by z and y is pushed on the stack.

Graphical Representation:-

The Graphical representation of PDA is a transition diagram.



$$\delta(q_1, a, b) = (q_2, c)$$

here,

a = input symbol.

b = stack symbol.

c = This symbol is pushed on the stack.

*** Acceptance of PDA :-

* There are 2 ways to accept a language by PDA. They are

- ① Acceptance by empty stack
- ② Acceptance by final state.

① Acceptance by empty stack :-

* The given language is accepted by empty stack, to be defined as

$$L(M) = \{ w / \delta(q_0, w, z_0) \xRightarrow{*} (p, \epsilon, \epsilon) \}$$

* for some 'p' in Q.

i.e. if stack becomes empty after scanning entire string then it is accepted by PDA otherwise not accepted.

② Acceptance by final state :-

* The given language is accepted by final state, to be defined as

$$L(M) = \{ w / \delta(q_0, w, z_0) \xRightarrow{*} (p, \epsilon, \gamma) \}$$

* for some 'p' $\in F$ and $\gamma \in \Gamma^*$

i.e. even though the stack is not empty after scanning input string, if the finite control reaches to the final state then it is accepted, otherwise not accepted.

Instantaneous Description (ID)

* An ID is defined as $\{ \text{triplet } (q, w, s) \}$
where q is a state in Q .
 w is string input symbol from Σ^*
and
 s is string of stack symbol in Γ^*

ex: $\delta(q_0, x, z_0) = (q, xz_0)$

29/04/24

1) Design a PDA for the language

$$L = a^n b^n / n \geq 1$$

sol)

$$n=2, w = aabb$$

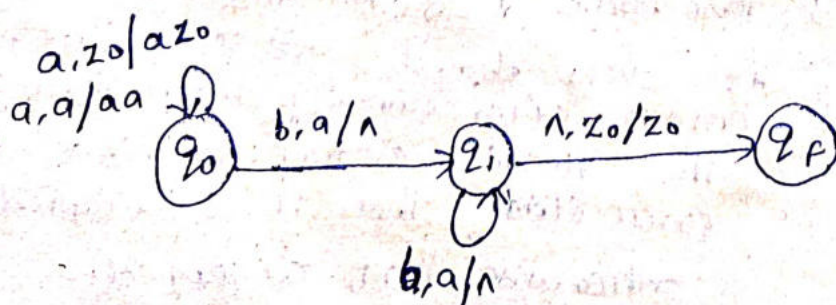
$$\delta(q_0, a, z_0) \rightarrow \delta(q_0, a, z_0)$$

$$\delta(q_0, a, a) \rightarrow \delta(q_0, aa)$$

$$\delta(q_0, b, a) \rightarrow \delta(q_1, \Lambda)$$

$$\delta(q_1, b, a) \rightarrow \delta(q_1, \Lambda)$$

$$\delta(q_1, \Lambda, z_0) \rightarrow (q_f, z_0)$$



example :- a^3b^3

$$(q_0, a^3b^3, z_0) \rightarrow (q_0, a^2b^3, az_0)$$

$$(q_0, a^2b^3, az_0) \rightarrow (q_0, ab^3, aaz_0)$$

$$(q_0, ab^3, aaz_0) \rightarrow (q_0, b^3, aaa)$$

$$(q_0, b^3, aaa) \rightarrow (q_1, b^2, aa)$$

$$(q_0, b^2, aa) \rightarrow (q_1, b, a)$$

$$(q_1, b, a) \rightarrow (q_1, \Lambda, z_0)$$

$$(q_1, \epsilon, z_0) \rightarrow (q_f, z_0)$$

2) Design a PDA for language L consists of equal no. of a's and b's.

$$\delta(q_0, a, z_0) \rightarrow (q_0, az_0)$$

$$\delta(q_0, b, z_0) \rightarrow (q_0, bz_0)$$

$$\delta(q_0, a, a) \rightarrow (q_0, aa)$$

$$\delta(q_0, b, b) \rightarrow (q_0, bb)$$

$$\delta(q_0, a, b) \rightarrow (q_1, \Lambda)$$

$$\delta(q_0, b, a) \rightarrow (q_1, \Lambda)$$

$$\delta(q_1, b, a) \rightarrow (q_1, \Lambda)$$

$$\delta(q_1, a, b) \rightarrow (q_1, \Lambda)$$

$$\delta(q_1, a, z_0) \rightarrow (q_1, a, z_0)$$

$$\delta(q_1, b, z_0) \rightarrow (q_1, b, z_0)$$

$$\delta(q_1, \Lambda, z_0) \rightarrow (q_f, z_0)$$

ex:

$$(q_0, abab, z_0) \rightarrow (q_0, bab, az_0)$$

$$(q_0, bab, az_0) \rightarrow (q_1, ab, z_0)$$

$$(q_1, ab, z_0) \rightarrow (q_1, b, az_0)$$

$$(q_1, b, az_0) \rightarrow (q_1, \Lambda, z_0)$$

$$(q_1, \epsilon, z_0) \rightarrow (q_f, z_0)$$

3) Design a pda for, $L = a^n b^m c^{n+m}$
 $w = a^2 b^3 c^5$

sol)

$$\delta(q_0, a, z_0) \rightarrow (q_0, az_0)$$

$$\delta(q_0, a, a) \rightarrow (q_0, aa)$$

$$\delta(q_0, b, a) \rightarrow (q_1, ba)$$

$$\delta(q_1, b, b) \rightarrow (q_1, bb)$$

$$\delta(q_1, b, b) \rightarrow (q_1, bb)$$

$$\delta(q_1, c, b) \rightarrow (q_2, \Lambda)$$

$$\delta(q_2, c, b) \rightarrow (q_2, \Lambda)$$

$$\delta(q_2, c, b) \rightarrow (q_2, \Lambda)$$

$$\delta(q_2, c, a) \rightarrow (q_3, \Lambda)$$

$$\delta(q_3, c, a) \rightarrow (q_3, \Lambda)$$

$$\delta(q_3, \epsilon, z_0) \rightarrow (q_f, z_0)$$

ex:

$$(q_0, a^2 b^3 c^5, z_0) \rightarrow (q_0, a b^3 c^5, a z_0)$$

$$(q_0, a b^3 c^5, a z_0) \rightarrow (q_0, b^3 c^5, a^2 z_0)$$

$$(q_0, b^3 c^5, a^2 z_0) \rightarrow (q_0, b^2 c^5, a b a^2 z_0)$$

$$(q_0, b^2 c^5, a b a^2 z_0) \rightarrow (q_0, b c^5, b^2 a^2 z_0)$$

$$(q_0, b c^5, b^2 a^2 z_0) \rightarrow (q_1, c^5, b^3 a^2 z_0)$$

$$(q_1, c^5, b^3 a^2 z_0) \rightarrow (q_2, c^4, b^2 a^2 z_0)$$

$$(q_2, c^4, b^2 a^2 z_0) \rightarrow (q_2, c^3, b a^2 z_0)$$

$$(q_2, c^3, b a^2 z_0) \rightarrow (q_2, c^2, a^2 z_0)$$

$$(q_2, c^2, a^2 z_0) \rightarrow (q_3, c, a z_0)$$

$$(q_3, c, a z_0) \rightarrow (q_3, \Lambda, z_0)$$

$$(q_3, \epsilon, z_0) \rightarrow (q_f, z_0)$$

10/11/21

Equivalence of CFG and PDA

① Conversion of CFG to PDA

Let $G = (V, T, P, S)$ is a context free grammar, we construct a PDA.

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where δ is defined by following rules.

Rule-1 :-

for each variable A

$$\delta(q, \epsilon, A) = \{(q, B) / A \rightarrow B \text{ is in } P\}$$

Rule-2 :-

for each terminal a

$$\delta(q, a, a) = (q, \epsilon)$$

1) Construct a PDA for following CFG.

$$G = (\{S, A\}, \{a, b\}, P, S)$$

$$P = \{S \rightarrow AA/a, A \rightarrow SA/b\}$$

sol)

$$\delta(q, \epsilon, S) = (q, AA), (q, a)$$

$$\delta(q, \epsilon, A) = (q, SA), (q, b)$$

$$\delta(q, a, a) = (q, \epsilon)$$

$$\delta(q, b, b) = (q, \epsilon)$$

$$M = (\{q\}, \{a, b\}, \{a, b, AA, SA\}, \{q\}, \{z_0\}, \{q\})$$

2) Construct a PDA for following CFG and test whether 010^4 is accepted by PDA.

$$S \rightarrow 0BB$$

$$B \rightarrow 0S/1S/0$$

2)

$$\delta(q, \epsilon, S) = (q, 0BB)$$

$$\delta(q, \epsilon, B) = (q, 0S), (q, 1S), \text{~~(q, 0)~~}, (q, 0)$$

$$\delta(q, 0, 0) = (q, 1)$$

$$\delta(q, 1, 1) = (q, 1)$$

$$M = S$$

Testing 010^4 is accepted or not

$$\delta(q, 010^4, S) \Rightarrow (q, 010^4, 0BB)$$

$$\Rightarrow (q, 10^4, BB)$$

$$\Rightarrow (q, 10^4, 1SB)$$

$$\Rightarrow (q, 10^4, SB)$$

$$\Rightarrow (q, 0^4, 0BBB)$$

$$\Rightarrow (q, 0^3, BBB)$$

$$\Rightarrow (q, 0^3, 0BB)$$

$$\Rightarrow (q, 0^2, BB)$$

$$\Rightarrow (q, 0^2, 0B)$$

$$\Rightarrow (q, 0, B)$$

$$\Rightarrow (q, 0, 0)$$

$$\Rightarrow (q, 1)$$

$\therefore 010^4$ is accepted.

3) Construct a PDA for CFG

$$S \rightarrow 0A$$

$$A \rightarrow 0AB/1$$

$$B \rightarrow 1$$

and test whether 0011 is accepted or not

Sol

$$\delta(q, \epsilon, S) = (q, 0A)$$

$$\delta(q, \epsilon, A) = (q, 0AB), (q, 1)$$

$$\delta(q, \epsilon, B) = (q, 1)$$

$$\delta(q, 0, 0) = (q, 1)$$

$$\delta(q, 1, 1) = (q, 1)$$

M =

$$\delta(q, 0011, S) = (q, 0011, 0A)$$

$$= (q, 011, A)$$

$$= (q, 011, 0AB)$$

$$= (q, 11, AB)$$

$$= (q, 11, 1B)$$

$$= (q, 1, B)$$

$$= (q, 1, 1)$$

$$= (q, \Lambda)$$

∴ 0011 is accepted by PDA.

(ii) Conversion of PDA to CFG

Let, M be the push down automata,
and $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ and

Let $G = (V, T, P, S)$ be a CFG

where, V is the set of variables
consists

(i) The special symbol S which is
the start symbol.

(ii) All symbols of the form PXQ ,
where P and Q are states in
 Q and X is stack symbol in
 Γ .

* The productions of G are as follows.

(a) for all states P , G has the production
 $S \rightarrow [q_0, Z_0, P]$

b) $\delta(q, a, z) = (q', \Lambda)$ induces the production $[q z q'] \rightarrow a$

c) $\delta(q, \epsilon, z) = (q', \Lambda)$ induces $[q z q'] \rightarrow \Lambda$

d) $\delta(q, a, z) = (q_1, z_1 z_2 \dots z_m)$ induces many productions of form,

$$[q z q'] \rightarrow a [q_1 z_1 q_2] [q_2 z_2 q_3] \dots [q_m z_m q'] \dots$$

1) Convert the following PDA into CFG.

$$M = (\{q, p\}, \{0, 1\}, \{x, z\}, \delta, q, z)$$

$$\delta(q, 1, z) = (q, xz)$$

$$\delta(q, 1, x) = (q, xx)$$

$$\delta(q, \epsilon, x) = (q, \epsilon)$$

$$\delta(q, 0, x) = (p, x)$$

$$\delta(p, 1, x) = (p, \epsilon)$$

$$\delta(p, 0, z) = (q, z)$$

1) First, we have to write the set of variables.

Variables are $\{s', [qxq], [qxp], [pxq], [pxp], [qzq], [qzp], [pzq], [pzp]\}$.

The set of productions are

$$S \rightarrow [q_0 z_0 q] : x$$

$$S \rightarrow [q_0 z_0 p]$$

$$(i) \delta(q, 1, z) = (q, \cancel{x}, z)$$

$$[qzq] \rightarrow 1 [qx p] [pzq]$$

$$[qzp] \rightarrow 1 [qx p] [pzp]$$

$$[qzq] \rightarrow 1 [qxq] [qzq]$$

$$[qzp] \rightarrow 1 [qxq] [qzp]$$

$$(ii) \delta(q, 1, x) = (q, \cancel{xx})$$

$$[qxq] \rightarrow 1 [qx p] [pxq]$$

$$[qx p] \rightarrow 1 [qx p] [px p]$$

$$[qxq] \rightarrow 1 [qxq] [qxq]$$

$$[qx p] \rightarrow 1 [qxq] [qx p]$$

$$(iii) \delta(q, \epsilon, x) = (q, \epsilon)$$

~~$$[qx p] \rightarrow \epsilon$$~~

$$[qxq] \rightarrow \epsilon$$

$$(iv) \delta(q, 0, x) = (p, x)$$

~~$$[qxq] \rightarrow 0 [pxq]$$~~

$$[qx p] \rightarrow 0 [px p]$$

$$\textcircled{v} \quad \delta(p, 1, x) = (p, \varepsilon)$$

$$[p x p] \rightarrow 1$$

$$[p x q] \rightarrow 1$$

$$\textcircled{vi} \quad \delta(p, 0, z) = (q, z)$$

$$[p z q] \rightarrow 0 [q z q]$$

$$[p z p] \rightarrow 0 [q z p]$$

01/05/24

2) Construct a CFG which accepts $\{a^n\}$ $n(A)$
~~where~~ where, $A = (\{q_0, q_1\}, \{a, b\}, \{z_0, z_1\}, \delta, q_0, z_0, \phi)$

$$\delta(q_0, b, z_0) = (q_0, z z_0)$$

$$\delta(q_0, \Lambda, z_0) = (q_0, \Lambda)$$

$$\delta(q_0, b, z) = (q_0, z z)$$

$$\delta(q_0, a, z) = (q_1, z)$$

$$\delta(q_1, a, z_0) = (q_0, z_0)$$

$$\delta(q_1, b, z) = (q_1, \Lambda)$$

30/11

Variables are $\{S, [q_0 z_0 q_0], [q_0 z_0 q_1],$
 $[q_1 z_0 q_0], [q_1 z_0 q_1], [q_0 z z_0], [q_0 z z_1],$
 $[q_1 z z_0], [q_1 z z_1]\}$

The set of productions are

$$S \rightarrow [q_0 z_0 q_0]$$

$$S \rightarrow [q_0 z_0 q_1]$$

$$(i) \delta(q_0, b, z_0) = (q_0, z z_0)$$

$$[q_0 z_0 q_0] \rightarrow b [q_0 z z_0] [q_1 z_0 q_0]$$

$$[q_0 z_0 q_1] \rightarrow b [q_0 z z_1] [q_1 z_0 q_1]$$

$$[q_0 z_0 q_0] \rightarrow b [q_0 z z_0] [q_0 z_0 q_0]$$

$$[q_0 z_0 q_1] \rightarrow b [q_0 z z_0] [q_0 z_0 q_1]$$

$$(ii) \quad \delta(q_0, \Lambda, z_0) = (q_0, \Lambda)$$

$$(q_0 z_0 q_0) \rightarrow \Lambda$$

$$(iii) \quad \delta(q_0, b, z) = (q_0, z z)$$

$$[q_0 z q_0] \rightarrow b [q_0 z q_1] [q_1 z q_0]$$

$$[q_0 z q_1] \rightarrow b [q_0 z q_1] [q_1 z q_1]$$

$$[q_0 z q_0] \rightarrow b [q_0 z q_0] [q_0 z q_0]$$

$$[q_0 z q_1] \rightarrow b [q_0 z q_0] [q_0 z q_1]$$

$$(iv) \quad \delta(q_0, a, z) = (q_1, z)$$

$$[q_0 z q_0] \rightarrow a [q_1 z q_0]$$

$$[q_0 z q_1] \rightarrow a [q_1 z q_1]$$

$$~~q_0 z q_1~~$$

$$(v) \quad \delta(q_1, a, z_0) = (q_0, z_0)$$

$$[q_1 z_0 q_0] \rightarrow a [q_0 z_0 q_0]$$

$$[q_1 z_0 q_1] \rightarrow a [q_0 z_0 q_1]$$

$$(vi) \quad \delta(q_1, b, z) = (q_1, \Lambda)$$

$$[q_1 z q_1] \rightarrow b$$

$a^n b^m c^n$

* Design a PDA for $a^n b^{2n}$

sol) $\delta(q_0, a, z_0) \rightarrow (q_0, az_0)$

$$\delta(q_0, a, a) \rightarrow (q_0, aa)$$

$$\delta(q_0, b, a) \rightarrow (q_1, a)$$

$$\delta(q_1, b, a) \rightarrow (q_2, \Lambda)$$

$$\delta(q_2, b, a) \rightarrow (q_1, a)$$

$$\delta(q_2, \Lambda, z_0) \rightarrow (q_f, z_0)$$

ex:

(q_0)

(q_1)

(q_2)

ex:

$$\delta(q_0, a^2 b^4, z_0) \rightarrow (q_0, ab^4, az_0)$$

$$\rightarrow (q_0, b^4, aaz_0)$$

$$\rightarrow (q_1, b^3, a^2 z_0)$$

$$\rightarrow (q_2, b^2, az_0)$$

$$\rightarrow (q_1, b, aaz_0)$$

$$\rightarrow (q_2, \Lambda, z_0)$$

$$\rightarrow (q_f, z_0)$$

* Design a PDA for $a^n b^m c^n$

sol)

$$\delta(q_0, a, z_0) \rightarrow (q_0, a z_0)$$

$$\delta(q_0, a, a) \rightarrow (q_0, aa)$$

$$\delta(q_0, b, a) \rightarrow (q_1, a)$$

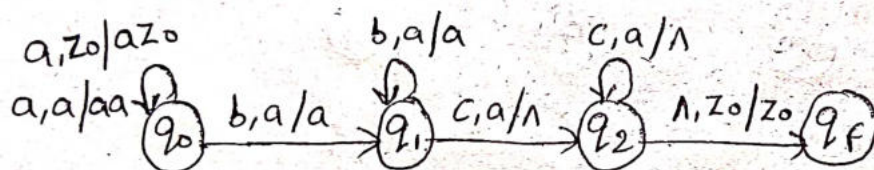
~~$$\delta(q_1, b, a) \rightarrow (q_1, a)$$~~

$$\delta(q_1, b, a) \rightarrow (q_1, a)$$

$$\delta(q_1, c, a) \rightarrow (q_2, \Lambda)$$

$$\delta(q_2, c, a) \rightarrow (q_2, \Lambda)$$

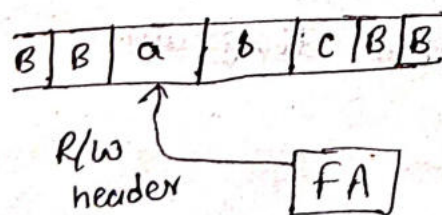
$$\delta(q_2, \Lambda, z_0) \rightarrow (q_f, z_0)$$



04/05/24

5. Turing Machines

- * Turing machine can be considered as "finite control machine" connected to R/w (Read/write) header, and the input tape which is divided into number of cells.
- * The machine checks the present symbol under the R/w head on the tape and the present state of automata in one move.
- * A Turing machine is made of finite control, an input tape which is divided into cells and a tape head that scans one cell at a time and can be moved to left as well as right, initially the n cells holds the input of length n where as the remaining ∞ of cells holds a special symbol called a Blank symbol B , & only one can be stored by each cell.



Defination :-

A Turing machine M is a 7 tuple variable as given below.

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

where,

Q = finite set of states of turing machine.

$\Sigma \subseteq \Gamma$ = subset of

Σ = input symbols.

Γ = finite set of symbols called as "tape symbols".

q_0 = initial state

B = Blank symbol.

$F \subseteq Q$ called as set of final states.

δ = a mapping from

$(Q \times \Gamma)$ to $(Q \times \Gamma \times \{L, R\})$

Representation of Turing Machine:-

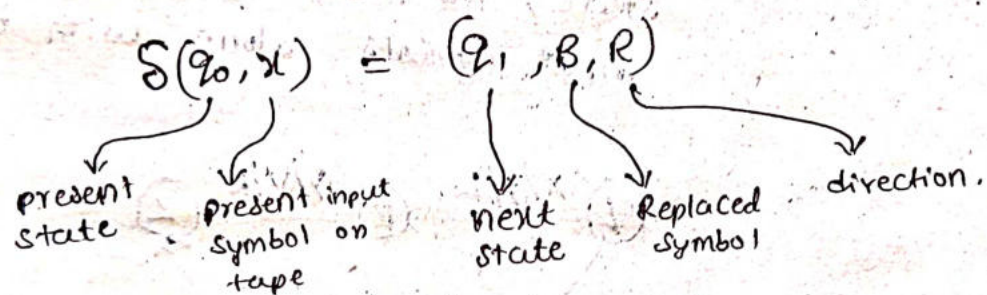
It is represented in 3 ways.

- 1) Instantaneous Description (ID) using move.
- 2) Transition table
- 3) Transition diagram.

① ID using move

* The arrangement of Turing machine depends on present state of finite control and the next input symbol which depends on an input tape at that instance of time.

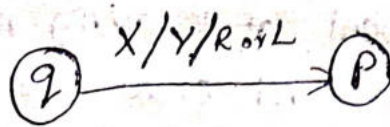
* If the current state is q_0 and the input of the next symbol appearing on the tape is 'x' and if the Turing machine changes its state to state q_1 in the right direction, replaces the input symbol 'x' with blank symbol 'B' then it is represented by following notation:



② Transition Diagram :-

* Transition diagram of next move function δ of Turing machine is a graphical representation, consisting of finite no. of nodes & directed labeled arcs b/w the nodes. Each And each node represents a state of Turing machine and a label on 'R' from one state to another state represents the information about required symbol, transition and the action on

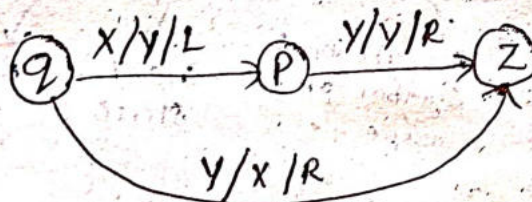
* Now we represent a move δ as,



③ Transsion Table :-

* It is a tabular representation of function like δ that takes two arguments and returns a state, replaced symbol and a tape direction.

* The rows of table corresponds to the state, the columns corresponds to input symbols along with blank symbols.



	X	Y	B
Q	P/Y/L	Z/X/R	-
P	-	Z/Y/R	-
Z	-	-	-

- 1) Design a Turing machine to recognize all strings consists of odd no of 1's over $\Sigma = \{1\}$

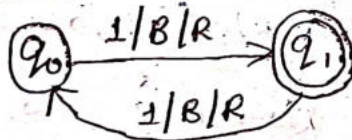
Diagram



Procedure:

q_0 is initial stage reading a symbol '1' from this - we enter into new state q_1 and replace '1' by a tape symbol 'B'.
 from q_1 if we read one more '1' then we again enter into q_1 by changing '1' into 'B'.

Diagram:-



As we are moving to the right side.

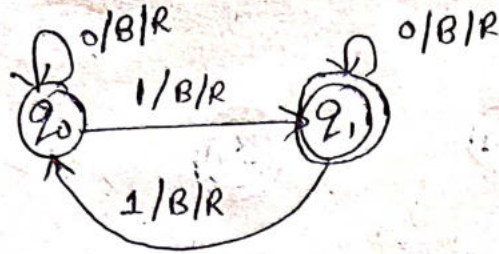
Table

	1	B
q_0	(q_1, B, R)	—
q_1	(q_0, B, R)	—

$$M = (\{q_0, q_1\}, \{1\}, \{1, B\}, \{\delta(q_0, 1) = (q_1, B, R), \delta(q_1, 1) = (q_0, B, R)\}, q_0, B, q_1)$$

ex:-
 $q_0 111$
 $B q_1 11$
 $BB q_0 1$
 $BBB q_1$

2) Design a Turing machine ~~ans~~ to accept all strings consists of odd number of 1's over $\Sigma = \{0, 1\}$



	0	1
q ₀	(q ₀ , B, R)	(q ₁ , B, R)
q ₁	(q ₁ , B, R)	(q ₀ , B, R)

$$M = (\{q_0, q_1\}, \{0, 1\}, \{0, 1, B\}, \delta, (q_0, 0) = (q_0, B, R), \\ \delta(q_0, 1) = (q_1, B, R), \delta(q_1, 0) = (q_1, B, R), \\ \delta(q_1, 1) = (q_0, B, R), q_0, B, q_1)$$

ex:

q₀ 1011

B q₁ 011

BB q₁ 11

BBB q₀ 1

BBBB q₁

q₀ 1010

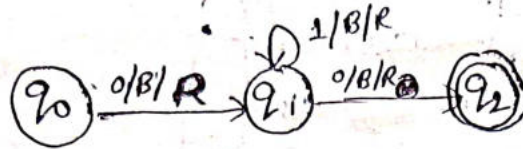
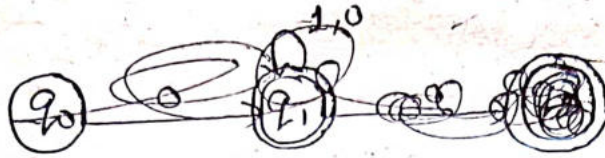
B q₁ 010

BB q₁ 10

BBB q₀ 0

BBBB q₁

3) Design a Turing machine to accept a string 01^*0 .



	0	1
q ₀	(q ₁ , B, R)	—
q ₁	(q ₂ , B, R)	(q ₁ , B, R)
q ₂	—	—

$$M = \{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, B\},$$

$$\left\{ \begin{aligned} \delta(q_0, 0) &= (q_1, B, R), & \delta(q_1, 0) &= (q_2, B, R), \\ \delta(q_1, 1) &= (q_1, B, R) \end{aligned} \right\}, q_0, B, q_2$$

ex:

q₀ 0 1 1 0

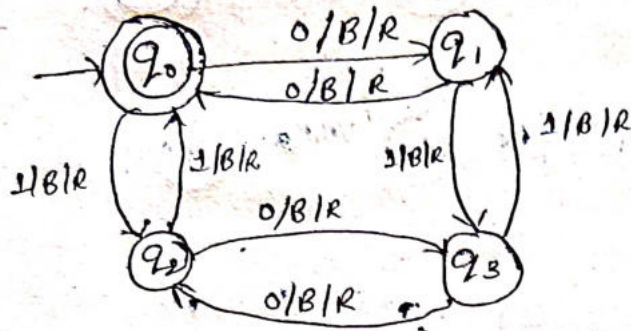
B q₁ 1 1 0

BB q₁ 1 0

BBB q₁ 0

BBBBB q₂

4) Design Turing machine to accept even no. of 0's and even no. of 1's.



	0	1
q ₀	(q ₁ , B, R)	(q ₂ , B, R)
q ₁	(q ₀ , B, R)	(q ₃ , B, R)
q ₂	(q ₃ , B, R)	(q ₀ , B, R)
q ₃	(q ₂ , B, R)	(q ₁ , B, R)

$$M = \{q_0, q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\},$$

$$\delta(q_0, 0) = (q_1, B, R), \delta(q_0, 1) = (q_2, B, R),$$

$$\delta(q_1, 0) = (q_0, B, R), \delta(q_1, 1) = (q_3, B, R),$$

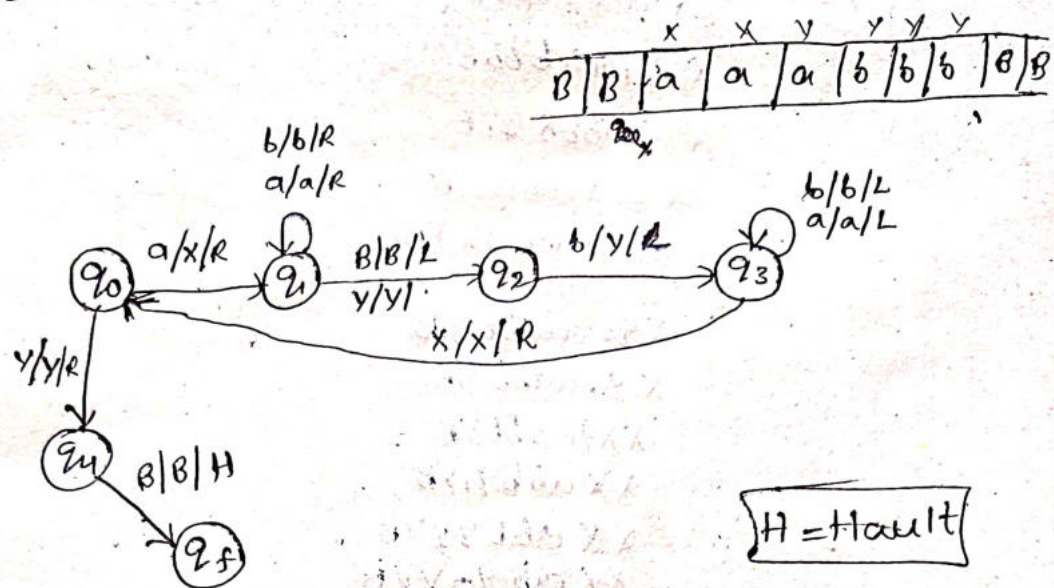
$$\delta(q_2, 0) = (q_3, B, R), \delta(q_2, 1) = (q_0, B, R)$$

$$\delta(q_3, 0) = (q_2, B, R), \delta(q_3, 1) = (q_1, B, R) \}$$

$$q_0, B, q_0 \}$$

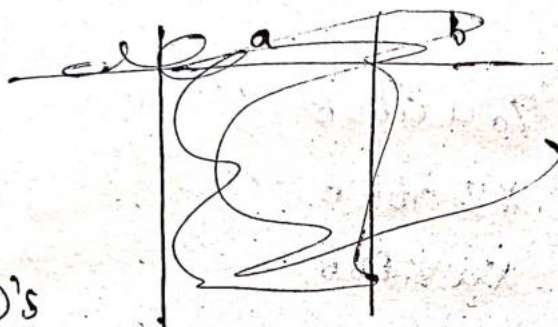
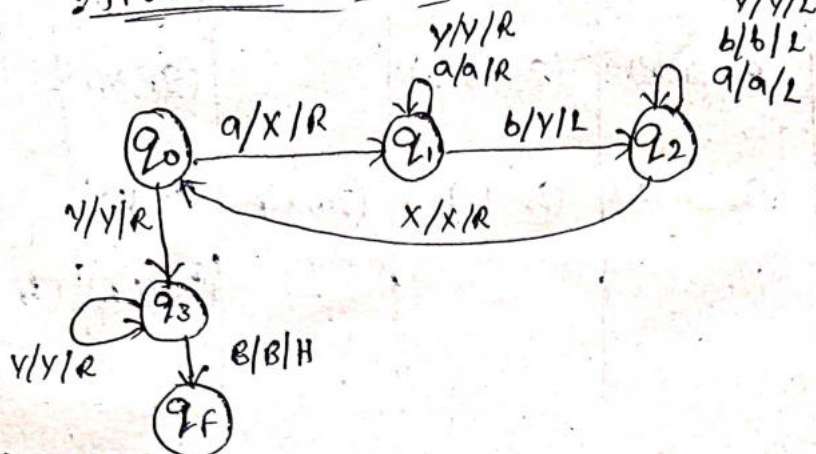
ex: 01010

5) Design a Turing machine for $a^n b^n$ such that $n \geq 1$.



06/05/24

Alternate method. for $a^n b^n$.



ID's

$$(q_0, a) = (q_1, x, R)$$

$$(q_0, a) = (q_1, a, R)$$

$$(q_1, y) = (q_1, y, R)$$

$$(q_1, b) = (q_2, y, L)$$

$$(q_2, a) = (q_2, a, L)$$

$$(q_2, b) = (q_2, b, L)$$

$$(q_2, y) = (q_2, y, L)$$

$$(q_2, x) = (q_2, x, R)$$

$$(q_0, y) = (q_3, y, R)$$

$$(q_3, y) = (q_3, y, R)$$

$$(q_3, B) = (q_f, H)$$

	a	b	x	y	B
q_0	(q_1, x, R)	-	-	(q_3, y, R)	-
q_1	(q_1, a, R)	(q_2, y, L)	-	(q_1, y, R)	-
q_2	(q_2, a, L)	(q_2, b, L)	(q_0, x, R)	(q_2, y, L)	-
q_3	-	-	-	(q_3, y, R)	(q_f, H)

ex:-

$q_0 a a b b B$

$x q_1 a b b B$

$x a q_1 b b B$

$x a q_2 y b B$

$x q_2 a y b B$

$x q_0 a y b B$

$x x q_1 y b B$

$x x y q_1 b B$

$x x y q_2 y B$

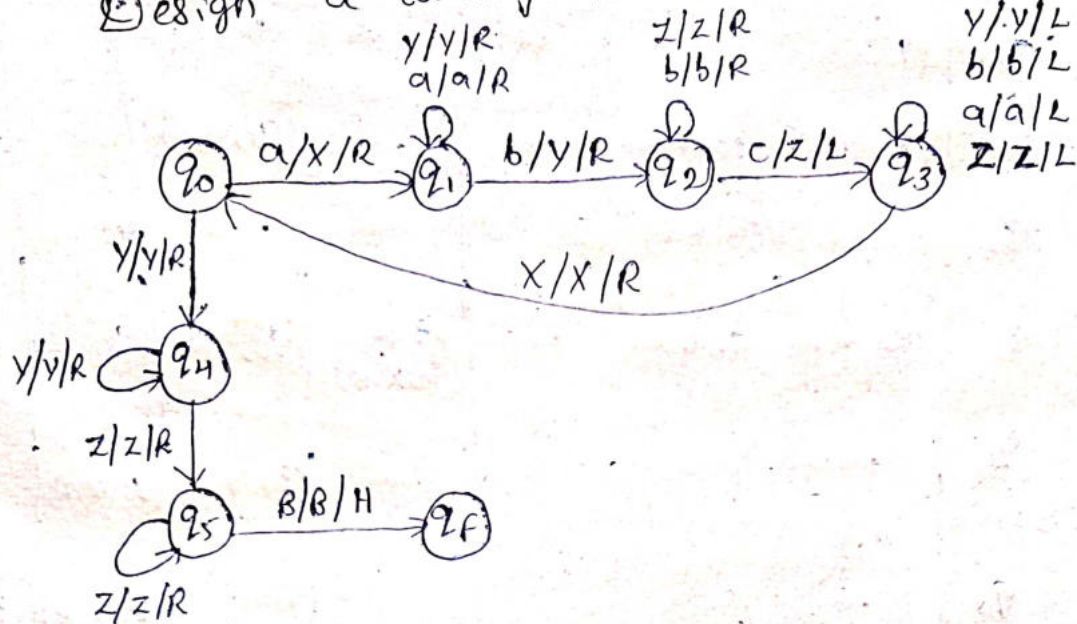
$x x q_2 y y B$

$x x q_0 y y B$

$x x y y q_3 B$

$x x y y B q_f$

6) Design a Turing machine for an bncn.



TD's

$$(q_0, a) = (q_1, x, R)$$

$$(q_1, a) = (q_1, a, R)$$

$$(q_1, y) = (q_1, y, R)$$

$$(q_1, b) = (q_2, y, R)$$

$$(q_2, b) = (q_2, b, R)$$

$$(q_2, z) = (q_2, z, R)$$

$$(q_2, c) = (q_3, z, L)$$

$$(q_3, a) = (q_3, a, L)$$

$$(q_3, b) = (q_3, b, L)$$

$$(q_3, y) = (q_3, y, L)$$

$$(q_3, z) = (q_3, z, L)$$

$$(q_3, x) = (q_0, x, R)$$

$$(q_0, y) = (q_4, y, R)$$

$$(q_4, y) = (q_4, y, R)$$

$$(q_4, z) = (q_5, z, R)$$

$$(q_5, z) = (q_5, z, R), (q_5, B) = (q_f, H)$$



	a	b	c	x	y	z	B
q_0	(q_1, x, R)	—	—	—	(q_4, y, R)	—	—
q_1	(q_1, a, R)	(q_2, y, R)	—	—	(q_1, y, R)	—	—
q_2	—	(q_2, b, R)	(q_3, z, L)	—	—	(q_2, z, R)	—
q_3	(q_3, a, L)	(q_3, b, L)	—	(q_0, x, R) (q_0, x, R)	(q_3, y, L)	(q_3, z, L)	—
q_4	—	—	—	—	(q_4, y, R)	(q_5, z, R)	—
q_5	—	—	—	—	—	(q_5, z, R)	(q_6, b)

7) Design a Turing machine for 1's complement.



TD's

$$(q_0, 0) = (q_0, 1, R)$$

$$(q_0, 1) = (q_0, 0, R)$$

	0	1
q_0	$(q_0, 1, R)$	$(q_0, 0, R)$

ex:

20101

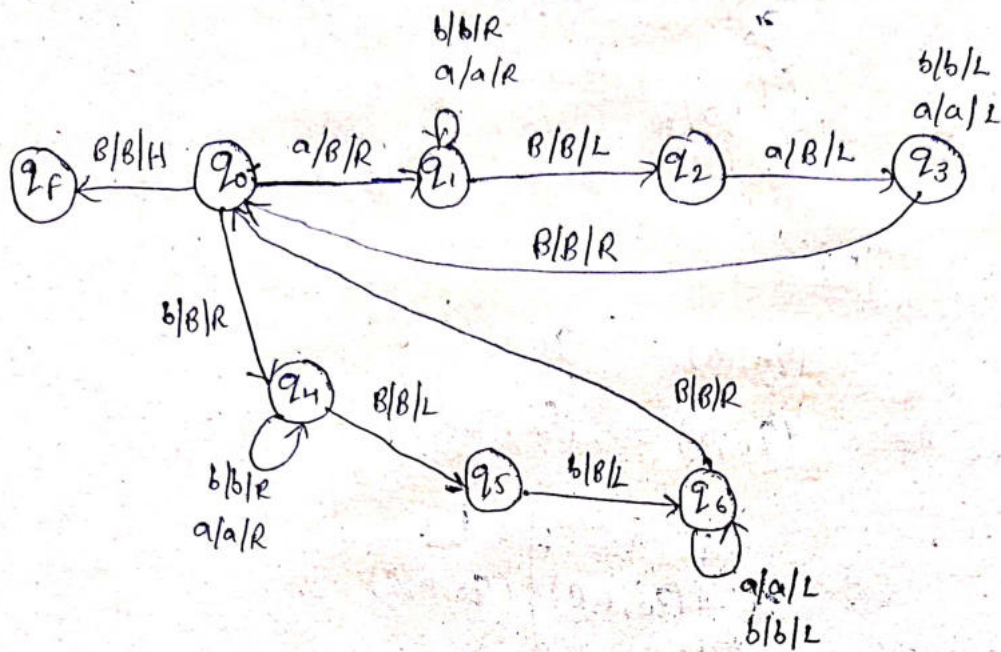
02001

01901

01090

07/05/24

8) Design a Turing machine for even palindrome i.e. $ww^R / w \in (a+b)^*$



TD's :-

$$(q_0, a) = (q_1, B, R)$$

$$(q_0, B) = (q_f, H)$$

$$(q_0, b) = (q_4, B, R)$$

$$(q_1, a) = (q_1, a, R)$$

$$(q_1, b) = (q_1, b, R)$$

$$(q_1, B) = (q_2, B, L)$$

$$(q_2, a) = (q_3, B, L)$$

$$(q_3, a) = (q_3, a, L)$$

$$(q_3, b) = (q_3, b, L)$$

$$(q_3, B) = (q_0, B, R)$$

$$(q_4, a) = (q_4, a, R)$$

$$(q_4, b) = (q_4, b, R)$$

$$(q_4, B) = (q_5, B, L)$$

$$(q_5, b) = (q_6, B, L)$$

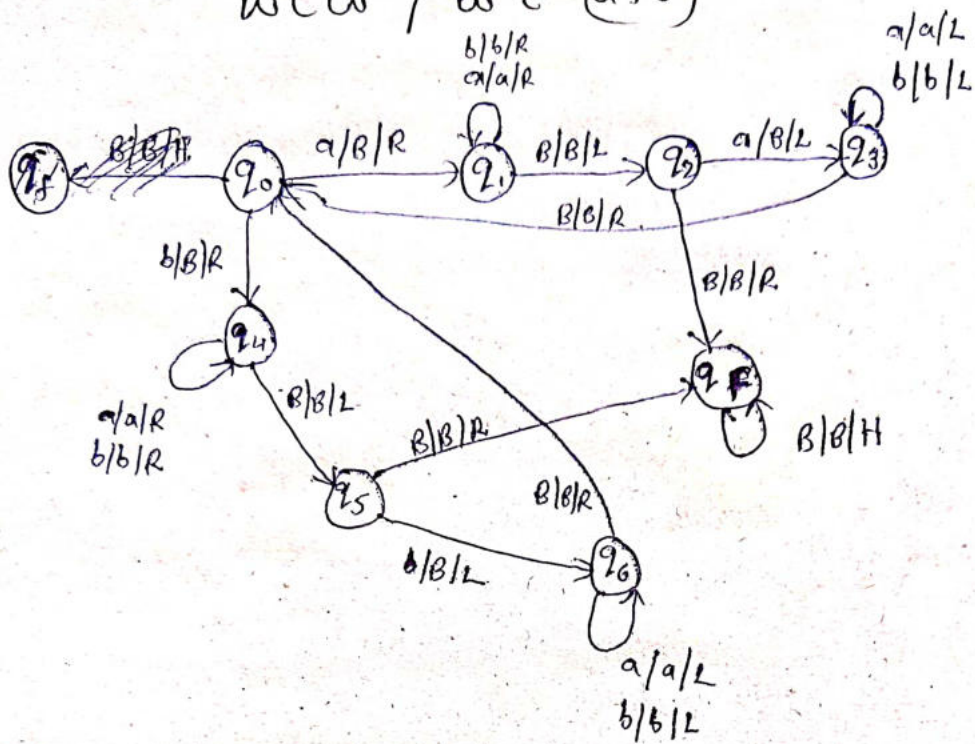
$$(q_6, a) = (q_6, a, L)$$

$$(q_6, b) = (q_6, b, L)$$

$$(q_6, B) = (q_0, B, R)$$

9) Design a Turing machine for odd palindrome i.e

$$w c w^R / w \in (a+b)^*$$



49 Computable languages and functions

A language is accepted by a Turing machine is recursively enumerable.

The term enumerable derives from the fact that it is precisely these languages whose strings can be enumerated by a Turing machine.

* The Turing machine has a computer of integer functions.

* In addition to being a language acceptor to the Turing machine may be viewed as a computer of engineering functions.

* ~~The functional~~

* The traditional approach is to represent integers in unary, the integer $i \geq 0$ is represented by string 0^i .

* If a function has k -arguments $i_1, i_2, \dots, i_k$ then these integers are initially placed on the tape separated by 1 is ~~at the end~~ ..

~~0~~ $0^{i_1} 1 0^{i_2} 1 0^{i_3} 1 \dots 0^{i_k} 1$

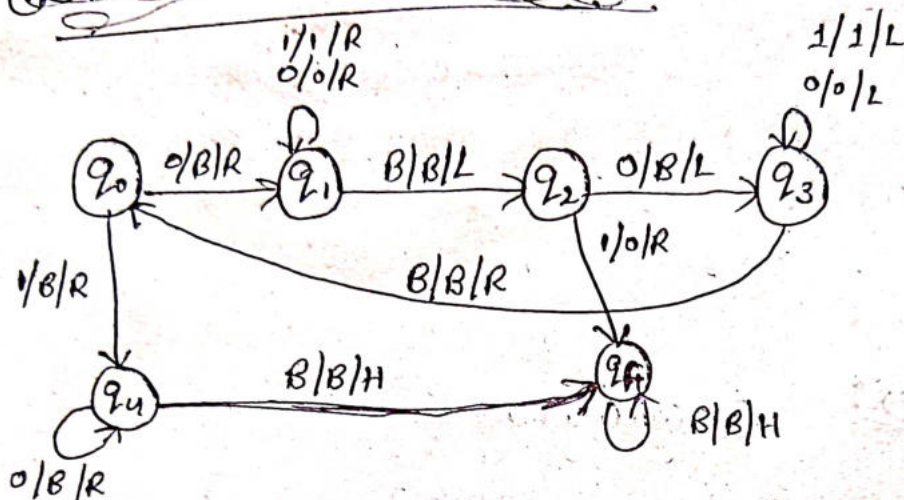
8/05/23

- 1) Design a Turing machine to compute proper subtraction, $m-n$ if $m > n$
 0 if $m < n$

B B 0000100 B

B B B B 00 B B B

~~for $m-n$ if $m > n$~~



ex:

$2_0.0000100\ B$

$B\ 2,000\ 100\ B$

$B\ 000\ 100\ 2, B$

$B\ 000\ 100\ 2_2\ B$

$B\ 000\ 100_3\ BB$

$B\ 2_3\ 000\ 10\ BB$

$B\ 2_0\ 000\ 10\ BB$

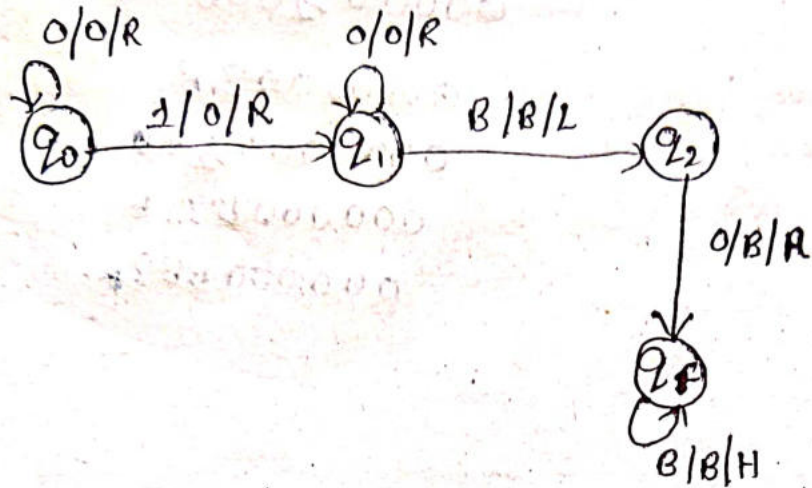
$BB\ 2,0010\ BB$

$BB\ 0010\ 2, BB$

$BB\ 0010\ 2_2\ BB$

2) Design a Turing machine to compute addition of two unary numbers.

00000100B B000000B



ex: $20\ 0000100\ B$

$\$ 00002_0.100\ B$

$000002_00\ B$

$00000002_1\ B$

$000000002_2\ B$

$0000000B2_3\ B$

$0000000BB2_4\ B$
