Theory of Computation Notes

Contributor: Abhishek Sharma

[Founder at TutorialsDuniya.com]

Computer Science Notes

Formal language and Automata theory

Computation:- The process of solving a problem to obtain a result is called computation.

→ The computation process can be represented by using Mathematical models.

Types of mathematical models:-

The Mathematical models are four types are there in given below:

* finite automata
* push down automata
* linear Bounded automata
* Turing Machine.

finite automata:- finite automata is understanding only one language, that language is called Regular language (RL) followed by with the help of regular grammer (RG)

Push down automata:- push down automata is understanding by particular language this language is known as context free language (CFL) is followed by with the understand by the also regular language.

linear bounded automata:- linear bounded automata is understand for language is context sensitive language (CSL) and formed by the with the help of context sensitive grammer (CSG) and long with the understand by two more languages are gRL and CFL Language.

Turing Machine:- Turing machine is understand for language is recursive enumerable language (REL) and is followed by with the help of grammer is recursive enumerable grammer (REGL) and also along with the understand by three more language RL & CFL & CSL language.

Relation btw automata:-

The relation btw above four language and grammers

$$RL \subseteq CFL \subseteq CSL \subseteq REL$$

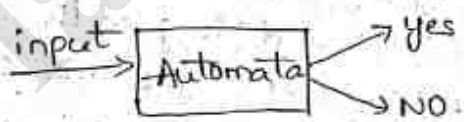
$$RG \subseteq CFG \subseteq CSG \subseteq REG$$

Why study automata theory :-

* This theory is a fundamental course of compil. science.
* Automata theory is the study of abstract mach and automata as well as the competational pro that can be solved using them.
* Automata theory will help you understand hou people have thought about computer science as a science.
* Automata theory is mainly about.
  1. what kind of things can you really compute mechanically.
  2. How fast it take to do it (time complexity)
  3. How much space does it take to do it (space com)

Eg :- 1. Binary strings ends with 0?
  1. 10101101<u>0</u> → accepted
  2. 10100010<u>1</u> → Rejected.

Eg : 2 Declaration statement in c language like
  int a, b, c;



The central concepts of Automata theory

Basic concepts :- 1. Symbol
  2. Alphabet
  3. strings
  4. languages.

1. Any formal language can be constructed by the basic concepts of automata theory.
2. Basic concepts of building blocks of automata theory.
  1. symbol :- Symbol is an obj (or) a thing
    Eg : a, b, c, d ----
    0, 1, 2, 3 ----

#, *, +, -, @ - etc.

symbols are used to form a string.

2. Alphabet :- It is a non empty and finite set of symbols

* It is denoted by Σ

Eg:- $\Sigma = \{a, b, \ldots z\}$

$\Sigma = \{A, B, \ldots Z\}$

$\Sigma = \{0, 1, 2, \ldots 9\}$

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

$\Sigma = \{a, b, c, \ldots z, 0, 1, 2, \ldots 9\}$

3. String :- It is a sequence of symbols from Sigma

Eg:- $S_1 = abc$

$S_2 = 010$

$S_3 = a, b, c$

$S_4 = 01234$

Length of a string :- The no. of symbols appears in a given string S is called of length of S

* It is denoted by $|s|$

Eg:- $S_1 = abc$ then $|S_1| = 3$

$S_2 = 010$ then $|S_2| = 3$

$S_3 = a, b, c$ then $|S_3| = 1$

$S_4 = 01234$ then $|S_4| = 5$

Language :- It is a collection of strings over sigma (Σ)

Eg:- let $\Sigma = \{0, 1\}$

L_1 = set of strings have length of 2

$L_1 = \{00, 01, 10, 11\}$ → finite set.

Eg:- L_2 = set of strings have length of 3

$L_2 = \{000, 001, 010, 01, 100, 101, 110, 111\}$ → finite set.

Eg:- L_3 = set of strings ends with 0

$L_3 = \{000, 00, 0, 10, 1010, 11110 \ldots\}$ → infinite set

power of Σ :- $\Sigma\{0, 1\}$

NULL string :- A string without symbol is a null string.

It is denoted by ϵ

∴ length of null string $|\epsilon| = 0$

Σ^0 set of all strings length 0 = $\{\epsilon\}$

Σ^1 set of all strings length 1 = $\{0,1\}$

Σ^2 set of all strings length 2 = $\{00, 01, 10, 11\}$

Σ^3 set of all strings length 3 = $\{000, 001, 010, 011, 100, 110, 111\}$

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \text{ ----->}$ star closure :- set of all strings including null string.

$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \text{ ----->}$ positive closure :- excluding null string.

1. $\Sigma^* = \Sigma^0 \cup \Sigma^+$
   $= \epsilon \cup \Sigma^+$
   $\boxed{\Sigma^* = \epsilon \cup \Sigma^+}$

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

String operations :-

1. Length of a string :- It means no. of symbols in the given string 's'. It is denoted by $|s|$

   eg :- let s = "google" be a string over $\Sigma = \{a, b, c, \text{---} z\}$
   Length s => $|s| = 7$

2. Position of a symbol in string :- consider an input symbol "a" in the input string s the position "a" in s is "q" then it is denoted by $s_a(i)$

   $\boxed{S_a(i) = 1}$ if a is in i^{th} position of s
   $\boxed{= 0}$ otherwise

   eg :- Consider an input symbol g in s then

   $s_g(1) = 1$ $s_g(4) = 1$ $s_g(7) = 0$
   $s_g(2) = 0$ $s_g(5) = 1$
   $s_g(3) = 0$ $s_g(6) = 0$

3. Concatenation :- It means combine two (or) more strings into a single string.

   -> mathematically if $s_1 \& s_2$ are two strings then the concatenation s' of $s_1 \& s_2$ is given by

   $s' = s_1 \circ s_2$
   $= \{xy \mid x \in s_1, y \in s_2\}$
   $= \{yx \mid y \in s_1, x \in s_2\}$

Eg:- If S_1 = para and S_2 = graph then $S_1 o S_2$

$\therefore S_1 o S_2$ = paragraph

Eg:- If $\Sigma = \{a, b\}$, $S_1 = ab$

$S_2 = baa$

then $s' = S_1 o S_2 = abbaa$

Eg:- let $x = 0100101$ and $y = 1111$

$x o y = 0100101111$

Eg:- $abba\, o\epsilon = abba|\epsilon| = 0$

Note:- Eg:- let z be any string then $zo\epsilon = \epsilon oz = z$

Identity rule

Note:- Associativity rule:- $a(bc) = ab(c)$

$a\,o(boc) = (aob)oc$

4. Reverse of a string:- Reverse of a string means, Reverse the symbols in a string.

→ It is denoted by s^R where s is given string

Eg:- 1. if $S = mus$ then $s^R = sum$.

2. let $S = bottle$ then $s^R = elttob$ and $(s^R)^R = bottle$

3. let $|s| = |s^R|$.

let $x = K\,lim$ and $y = rettub$ then $(xoy)^R$

$x o y = K\,lim\,rettub$

$(xoy)^R = buttermilK$.

$y^R o\, x^R = y^R = butter \quad x^R = milK$.

$y^R o x^R$ = buttermilK $\quad [\because (xoy)^R = y^R o x^R$

Kleene closure (or) star closure :-

kleene closure is introduced by Kleene mathematician

It is a set which contains all strings including empty string (or) null string.

It is denoted by s^*

It is used for regular expression

$s^* = \{s^0, s^1, s^2, s^3, s^4 \cdots\}$

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

Eg:- if $s = \{a\}$ then find s^*

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

$s = \{a\}$

$S^0 = \{\epsilon\}$ $S^3 = \{aaa\}$

$S^1 = \{a\}$ $S^4 = \{aaaa\}$

$S^2 = \{aa\}$

$S^* = S^0 \cup S^1 \cup S^2 \cup S^3 \cup S^4 \cup \cdots$

$= \{\epsilon\} \cup \{a\} \cup \{aa\} \cup \{aaa\} \cup \{aaaa\} \cup \cdots$

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

Eg:- if $S = a, b$ then find S^*

$S = \{a, b\}$

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

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

$S^1 = \{a\} \{b\} = \{a, b\}$

$S^2 = \{aa, bb, ab, ba\}$

$S^3 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$

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

$= \{\epsilon, a, b, aa, bb, ab, ba, aaa, aab, aba, abb, baa, bbb\}$

Eg:- if $S = \{cc, d\}$ then find S^*

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

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

$S^1 = \{d\}$

$S^2 = \{cc, dd\}$

$S^3 = \{ccd, dcc, ddd\}$

$S^4 = \{cccc, ccdd, ddcc, dddd\}$

$\therefore S^* = S^0 \cup S^1 \cup S^2 \cup S^3 \cup \cdots$

$= \{\epsilon, d, cc, dd, ccd, dcc, ddd, cccc, ccdd, ddcc, dddd\}$

Positive closure:- It was introduced by Kleen mathematician

-> It is a set which contains all strings excluding null (or) empty string.

-> Denoted by S^+

-> used for regular Expression

$$\therefore S^+ = S^1 \cup S^2 \cup S^3 \cup \cdots$$

eg:- 1) if $S=\{a\}$ then S^+

$S^+ = S^1 \cup S^2 \cup S^3 \cup S^4 \cup \cdots$

$S^1 = \{a\}$ $\quad S^4 = \{aaaa\}$

$S^2 = \{aa\}$

$S^3 = \{aaa\}$

$S^+ = S^1 \cup S^2 \cup S^3 \cup S^4 \cup \cdots$

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

2) if $S=\{a, ba\}$ then find S^+

$S^1 = \{a\}$

$S^2 = \{aa, ba\}$

$S^3 = \{aaa, aba, baa\}$

$S^4 = \{aaaa, aaba, baaa, baba\}$

$S^+ = S^1 \cup S^2 \cup S^3 \cup S^4 \cdots$

$= \{a, aa, ba, aaa, baa, aba, aaaa, aaba, baaa, baba\}$

Note :- Relation ship b/w S^* and S^+

$S^* = S^0 \cup S^+$ $\quad S^+ = S^* - S^0$

$\boxed{S^* = \epsilon \cup S^+}$ $\quad \boxed{S^+ = S^* - \epsilon}$

Parts of a string :- 1. prefix of a string
2. Suffix of a string
3. proper prefix of a string
4. proper suffix of a string.

1) prefix of a string :- The number of leading symbols of given string.

⇒ let 'x' be a string then prefix of 'x' is denoted by prefix(x).

eg:- if $x=abc$ is a string then prefix of x is

prefix(x) = $\{\epsilon, a, ab, abc\}$

2) Suffix of a string :- The number of trailing symbols in a given string.

⇒ It is denoted by suffix(x)

eg:- if $x=abc$ is a string then suffix of x is

suffix(x) = $\{\epsilon, c, bc, abc\}$

3) proper prefix of a string :- The no. of leading symbols except the given string.

→ It is denoted by proper prefix (x)

Eg:- if x: abc is a string then proper prefix (x)

proper prefix (x) = {∈, a, ab}

4) proper suffix of a string:- The no. of trailing symbols except the given string.

→ It is denoted by proper suffix (x)

Eg:- If x = abc is a string then proper suffix (x)

proper suffix (x) = {∈, c, bc}

language:-

* Introduction * operations of language

Introduction:- A language is a finite and non empty set of strings. It is denoted by L.

→ All these strings are formed from the alphabet Σ.

language notation L(M):- is a language defined by machine M that accepts a set of strings.

→ L(G) is a language defined by the Grammer "G" that recognise a set of strings.

→ L(r) is a language defined by the regular Expression that represent a set of strings.

Eg:-

1) let Σ = {z} then the language of all possible strings is given by Σ = {z}

L = {∈, z, zz, zzz, zzzz}

2) The language of all possible of even length strings over Σ = {z}

$L = \{z^0, z^2, z^4, z^6 z^8 ---\}$

$L = \{z^{2n} / n \geq 0\}$

Eg:- The language of all possible strings of odd length over. Σ = {z}

$L = \{z^1 z^3 z^5 z^7 z^9 ---\}$

$L = \{ z^{2n-1} / n \geq 1 \}$

operations on language :-

1. union
2. Intersection
3. complementation
4. symmetric difference.
5. Reversal of language
6. polindrome language
7. kleene closure (or) star closure of language
8. Demorgan's law

1) 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 defing as $L_1 \cup L_2 = \{ x / x \in L_1 \text{ or } x \in L_2 \}$

Eg :- let $L_1 = \{0, 01, 011\}$ and $L_2 = \{\epsilon, 001, 1^5\}$ then $L_1 \cup L_2 =$

$\{\epsilon, 0, 01, 011, 001, 1^5\}$

2) let $L_i = 2^i$ then $\bigcup_{i=0}^{7} L_i =$

$\bigcup_{i=0}^{7} L_i = L_0 \cup L_1 \cup L_2 \cup L_3 \cup L_4 \cup L_5 \cup L_6 \cup L_7$

$= 2^0 \cup 2^1 \cup 2^2 \cup 2^3 \cup 2^4 \cup 2^5 \cup 2^6 \cup 2^7$

$= \{\epsilon\} \cup \{2\} \cup \{22\} \cup \{222\} \cup \{2222\} \cup \{22222\} \cup \{222222\} \cup \{2222222\}$

$\{\epsilon, 2, 22, 222, 2222, 22222, 222222, 2222222\}$

2) Intersection :- It is a simple operation on two languages. and l_1 & l_2 be two languages and their intersection is denoted by $L_1 \cap L_2$

→ mathematically intersection of L_1 & L_2 is defined as

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

let $L_1 = \{\epsilon, 00, 0000, 000000, \ldots\}$ and

$L_2 = \{\epsilon, 0, 000, 00000, \ldots\}$ then $L_1 \cap L_2$

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

Let $L_1 = \{\epsilon\}$ and $L_2 = \{\phi\}$ $\therefore L_1 \cap L_2 = \phi$ //

3) Complementation :-

It is a simple operation performed on single langu

→ let L be a language over Σ then the complement of L is denoted by $\bar{L}$ (or) L^c therefore $\bar{L}$ or L^c =

$$\boxed{\bar{L} \text{ or } L^c = \Sigma^* - L}$$

→ Mathematically the complementation of L is defin as $\bar{L} = \{x | x \in \Sigma^* \text{ and } x \notin L\}$

Eg:- if $\Sigma = \{a, b\}$ and $L = \{a, b, aa\}$ then $L^c = ?$

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

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

$$\Sigma^1 = \{a\}\{b\}$$

$$\Sigma^2 = \{aa, ab, ba, bb\}$$

$$\Sigma^3 = \{aaa, aab, baa, aba, bba, bab, bbb, abb\}$$

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

$$= \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, baa, aba, bba, bab, bbb, abb\} - \{a, b, aa\}$$

$$= \{\epsilon, ab, ba, bb, aaa, aab, baa, aba, bba, bab, bbb, abb\}$$

2) let $L = \{\epsilon, 1, 11, 111, 1111 \cdots\}$ over $\Sigma = \{0, 1\}$ then find $\bar{L}$

$$\bar{L} = \Sigma^* - L$$

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

$$\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, \cdots\} - \{\epsilon, 1, 11, 111, 1111 \cdots\}$$

$$= \{0, 00, 01, 10, 000, 001, 010, 011, 100, 101, 110 \cdots\}$$

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 & L_2 is defined as

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

-> Here x is in L_1 or L_2 but not in both.

eg:- let L be a language over Σ

1) $L \oplus \phi = (L \cup \phi) - (L \cap \phi)$
$= L - \phi$
$= L$

2) $L \oplus L = (L \cup L) - (L \cap L)$
$= L - L$
$= \phi$

3) $L \oplus \Sigma^* = (L \cup \Sigma^*) - (L \cap \Sigma^*)$
$= \Sigma^* - L$
$= \overline{L}$

4) $L \oplus \overline{L} = (L \cup \overline{L}) - (L \cap \overline{L})$
$= \Sigma^* - \phi$
$= \Sigma^*$

eg:- let $L_1 = \{00, 0000, 000000, \dots\}$ $L_2 = \{0, 1111, 111111, \dots\}$ then

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

$= \{00, 0000, 000000 \dots, 0, 1111, 111111, \dots\} - \{\phi\}$

$= \{00, 0000, 000000, \dots, 0, 1111, 111111, \dots\}$

eg:- If $\Sigma = \{0, 1\}$ then $\Sigma^{\leq 2} \oplus \{\epsilon, 0, 00, 101, \dots\}$

$\Sigma^{\leq 2} = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2$

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

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

$\Sigma^{\leq 2} \oplus \{\epsilon, 0, 00, 101\}$

$= (\Sigma^{\leq 2} \cup \{\epsilon, 0, 00, 101, \dots\}) - (\Sigma^{\leq 2} \cap \{\epsilon, 0, 00, 101, \dots\})$

$= \{\epsilon, 0, 1, 00, 01, 10, 11\} \cup \{\epsilon, 0, 00, 101, \dots\} - \{\epsilon, 0, 1, 00, 01, 10\} \cap \{\epsilon, 0, 00, 101\}$.

concatenation of language:-

concatenation of language 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

$$L_1 L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$$

eg:- i) Let $L_1 = \{bc, bcc, cc\}$ and $L_2 = \{cc, cccc\}$ then

i) $L_1 L_2 = ?$ ii) $L_2 L_1 = ?$

$L_1 L_2 = \{bc, bcc, cc\} \circ \{cc, cccc\}$

$= \{bccc, bccccc, bcccc, bccccc, cccc, ccccc\}$

$L_2 L_1 = \{cc, cccc\} \circ \{bc, bcc, cc\}$

$= \{ccbc, ccbcc, cccc, ccccbc, ccccbcc, ccccc\}$

ii) Let $L_1 = \{0,1\}^*$ and $L_2 = \{0,1\}$ then $L_1 L_2 = ?$

$L_1 = \{0,1\}^*$

$L_1 = L_1^0 \cup L_1^1 \cup L_1^2 \cup L_1^3 \cup L_1^4 \dots$

$L_1^0 = \{\epsilon\}$

$L_1^1 = \{0,1\}$

$L_1^2 = \{00, 01, 10, 11\}$

$L_1^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

$L_1 = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, \dots\}$

$L_2 = \{0,1\}$

$L_1 L_2 = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, \dots\} \{0,1\}$.

$= \{0, 00, 10, 000, 010, 110, 000, 0010, 0100, 0110, 1000, 1010, 1100, 1110,$
$1, 01, 11, 001, 011, 101, 111, 0001, 0011, 0101, 0111, 1001, 1101, 1111\}$

3) for any language $L\epsilon = \epsilon L = L$

4) for any language $L\phi = \phi L = \phi$

Reversal of Language:-

Language = set of strings.

It is similar to Reverse of strings.

It means Reverse the all strings in that language

It can be denoted by "L^R"

mathematically it can be defined as $L^R = \{w^R / w \in L\}$

Eg:- 1) If $L = \{a_1 b, a_2 b_2, a_3 b_3\}$ then L^R

$L^R = \{a, b, b_2 a_2, b_3 a_3\}$

2) If $L = \{0, 01, 011\}$ then L^R

$L^R = \{0, 10, 110\}$

3) If $L = \{0^{i, i+1} / i \geq 0\}$ then L^R

$L^R = \{1^{i+1} 0^i / i \geq 0\}$

Kleene closure of a language:-

It is a set of all strings including null string (or) empty string.

It is denoted by L^*

mathematically it is defined as $L^* = Lx^i, i \geq 0$

Here, x^0, x^1 is 0 number of x.

positive closure of a language:-

It is a set of all strings excluding Null string (or) Empty string.

It is denoted by L^+

Mathematically it is defined as $L^+ = L^1 \cup L^2 \cup L^3 \cup \ldots$

$$L^+ = \cup x^i, \; i \geq 1$$

eg:-1 If $L = \{\epsilon, z, zz, zzz, \ldots\}$ over $\Sigma = \{z\}$ then

$L^* = ?$

$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ldots$

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

$L^1 = \{z\}$

$L^2 = \{zz\}$

$L^3 = \{zzz\}$

$L^* = \{\epsilon, z, zz, zzz, \ldots\}$

2) If $L = \{\epsilon, 0, 1, 00, 01, 10, 11, \ldots\}$ over $\Sigma = \{0, 1\}$ then

$L^* = ?$

$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ldots$

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

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

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

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

$\therefore L^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111 \ldots\}$

Eg:3 If $\{000\}^*$ =?

$\{000\}^* = \{000\}^0 \cup \{000\}^1 \cup \{000\}^2 \cup \{000\}^3 \cup \cdots$

$= \{\epsilon\} \cup \{000\} \cup \{000000\} \cup \{000\,000\,000\} \cup \cdots$

$= \{\epsilon, 000, 000000, 000000000\}$

4) ϕ^* =?

$\{\phi\}^* = \{\phi\}^0 \cup \{\phi\}^1 \cup \{\phi\}^2 \cup \{\phi\}^3 \cup \cdots$

$= \{\epsilon\} \cup \{\} \cup \{\} \cup \{\} \cup \cdots$

$= \{\epsilon\}$

5) If $L = \{\epsilon\}$ then L^* =?

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

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

$L^1 = \{\epsilon\}^1$

$L^2 = \{\epsilon\}^2$

$L^3 = \{\epsilon\}^3$

$\therefore L^* = \{\epsilon\} \cup \{\epsilon\}^1 \cup \{\epsilon\}^2 \cup \{\epsilon\}^3$

$= \{\epsilon\}$

//

Demorgan's laws :-

Demorgan's laws used to express the intersection of languages interms of union and different languages.

i) $L_1 - (L_2 \cup L_3) = (L_1 - L_2) \cap (L_1 - L_3)$

ii) $L_1 - (L_2 \cap L_3) = (L_1 - L_2) \cup (L_1 - L_3)$

iii) $(L_1 \cup L_2)' = L_1' \cap L_2'$

iv) $(L_1 \cap L_2)' = L_1' \cup L_2'$

v) $L_1 \cap L_2 = (L_1' \cup L_2')'$

Finite Automata :-

* Introduction
* Componente of FA
* elements of FA
* Representation of FA.
* Examples.

Introduction :-

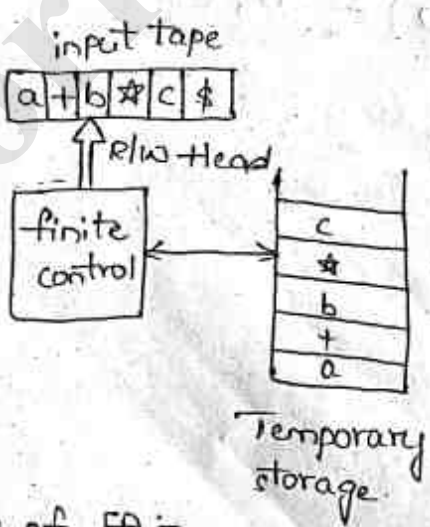
It is a self operating machine.

It is a system which obtains, transforms, transmits and uses information to perform its functions without direct participation of humans.

Finite automata is used in mathematical analysis

Application of finite Automata is lexical Analysis phases in compiler design.

Model of finite automata :-



components of FA :-

1) Input tape 2) R/W Head 3) Temporary storage.

4) Finite control.

1) Input tape:-

It is a memory storage used to store the input data memory area is divided into number of CELLS.

Each cell can hold only one input symbol at a time.

The input string ends with End Marker.

2) R/W head:-

It is used by the finite control to read the input data from left side to right side in the input tape.

R/W head can move from left to right and reads only one input symbol at a time.

3) Finite control unit:-

* It controls the entire process of a system (or) machine.
* Finite control reads the input data from the input buffer tape by moving read or write head from left to right.
* It stores the reading data in temporary storage.
* It stops the reading process when the read (or) write head reaches to end marker in the input tape.

* Elements of finite Automata:-

1. state 2. Transitions 3. Transition diagram / state diagram
4. Transition table.

1. state:- It is a behaviour that produce an action.

* It is a location in input buffer.
* The finite control reads the data from one state to another state in the buffer.
* states are classified into four types.
  i) initial state (or) starting state.
  ii) final state (or) Acceptance state
  iii) Intermediate state
  iv) Invalid state (or) Dead (or) Trap state.

Initial state (or) starting state:-

The finite control can starts its reading process from a state is called initial state.

2. Finite state or acceptance state :-

Finite control can stops its reading process after complete or end of given string then it reaches a state that state is called final state.

3. Intermediate state :-

The states between starting state and final state are called Internal (or) Intermediate state.

4. Invalid state (or) Dead (or) Trap state :- It is a invalid state when the finite control reads the input data from dead state then it always goes to dead state.

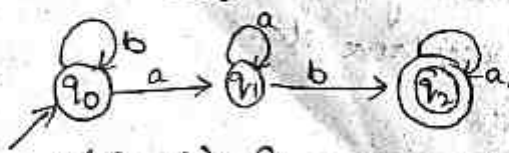
(2.) Transitions :-

* It means moving one place to another place.
* It is defined by Transition function.
* Transition function is denoted by δ.

3. Transition Diagram :-

* It is a graphical representation of finite automata.
* It is a directed graph.
* It is constructed by using the following symbols (or) Notations.

Symbol	Meaning
	state
	initial state
	final state
	dead state
	Transition

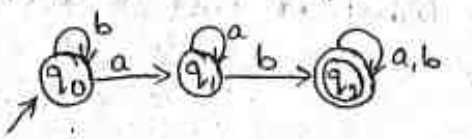
Ex :- 

$\delta(q_0, a) = q_1$
$\delta(q_0, b) = q_0$
$\delta(q_1, b) = q_2$
$\delta(q_1, a) = q_1$
$\delta(q_2, a) = q_2$
$\delta_2(q_2, b) = q_2$

4. Transition tables :-

* It is a tabular representation of finite automata.
* It is combination of rows and columns. Here rows represent states columns represents input symbols. The entry in between row and column is always states.

Eg :-



state	input symbols	
	a	b
→ q_0	q_1	q_0
q_1	q_1	q_2
(q_2)	q_2	q_2

** Representation of finite automata :-

Mathematically a finite automata is a 5-tuple machine like $M = (Q, \Sigma, \delta, q_0, F)$ where,

$Q \rightarrow$ set of states.

Q is a finite and Non-empty set of states.

$\Sigma \rightarrow$ It is finite and Non-empty set of input symbols.

$\delta \rightarrow$ It is a transition function which is defined as $\delta: Q \times \Sigma \rightarrow Q$.

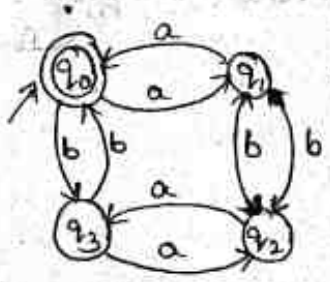
$q_0 \rightarrow$ It is a initial state or starting state and $q_0 \in Q$.

$F \rightarrow$ It is a finite set of final states and $F \subseteq Q$

Note :- * finite automata contains one initial state.

* finite automata, contains one or more final states.

Ex :-



Mathematically it represented as.

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

$Q = \{q_0, q_1, q_2, q_3\}$

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

δ is a transition function which is defined as

$\delta: Q \times \Sigma = Q.$

Transition table:

δ: state	input a	b
→ q_0	q_1	q_3
q_1	q_0	q_2
q_2	q_3	q_1
q_3	q_2	q_0

$q_0 = \{q_0\}$

$F = \{q_0\}$

** (imp) Acceptance of a string by FA:-

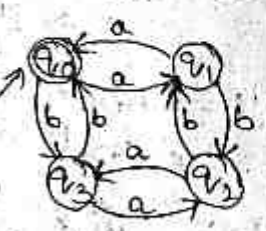
Let w be a given string and finite automata M contains Q states like $Q = \{q_0, q_1, q_2, \dots q_f\}$ where q_0 is the initial state q_f is the final state. The string w is accepted by machine M. If and only if $\delta(q_0, w) = q_f$

Ex:- check whether the following strings are accepted or not by the given finite automata.

i) aabb ii) ababa iii) aabbaa iv) abababb

Sol:-

The given finite automata



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

$Q = \{q_0, q_1, q_2, q_3\}$

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

δ: state	input a	b
→ q_0	q_1	q_3

q_1	q_0	q_2
q_2	q_3	q_1
q_3	q_2	q_0

$q_0 = \{q_0\}$

$F = \{q_0\}$

i) $w = aabb$

$\delta(q_0, aabb) = \delta(q_1, abb)$

$= \delta(q_0, bb)$

$= \delta(q_3, b)$

$= q_0 \in F$

∴ The given string $w = aabb$ is accepted by F.A M.

ii) $w = ababa$

$\delta(q_0, ababa) = \delta(q_1, baba)$

$= \delta(q_2, aba)$

$= \delta(q_3, ba)$

$= \delta(q_0, a)$

$= q_1$

∴ The given string $w = ababa$ is not accepted by F.A.

iii) $w = aabbaa$

$\delta(q_0, aabbaa) = \delta(q_1, abbaa)$

$= \delta(q_0, bbaa)$

$= \delta(q_3, baa)$

$= \delta(q_0, aa)$

$= \delta(q_1, a)$

$= q_0 \in F$

∴ The given string $w = aabbaa$ is accepted by F.A M.

iv) $w = abababb$

$\delta(q_0, abababb) = \delta(q_1, bababb)$

$= \delta(q_2, ababb)$

$= \delta(q_3, babb)$

$= \delta(q_0, abb)$

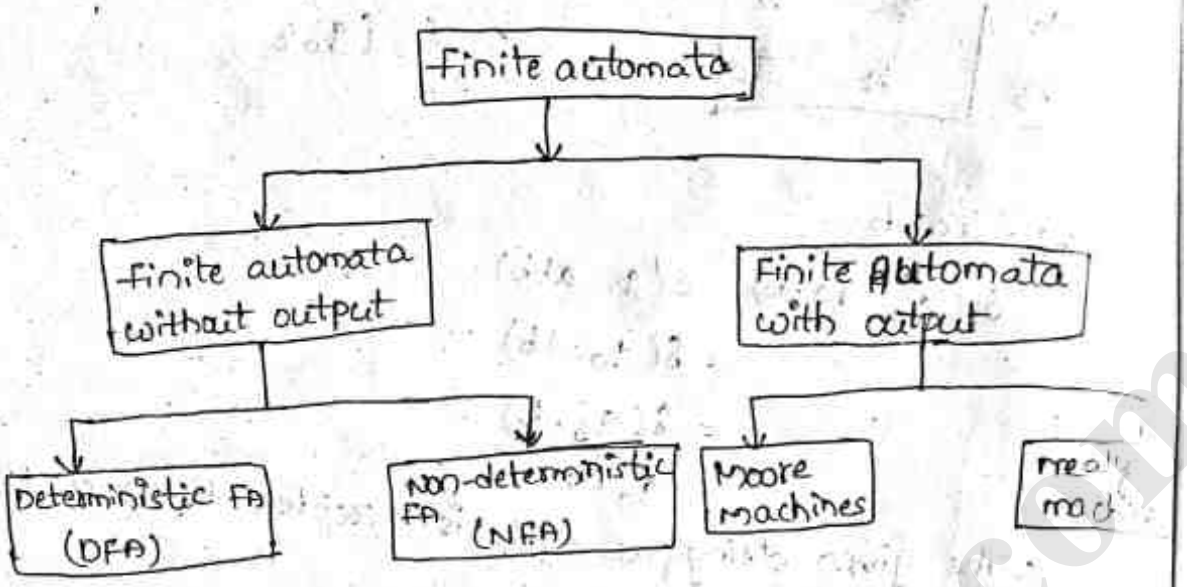
$= \delta(q_1, bb)$

$= \delta(q_2, b)$

$= q_1$

∴ The given string $w = abababb$ is not accepted by F.A.

Types of FA:-



Deterministic FA:-

Introduction:- *Introduction *Representation *language Accept by DFA

* Acceptance of DFA string by DFA

* Design of DFA.

Introduction:-

We can determine exactly what is the next state by reading a particular input symbol from a particular state then that FA is called DFA.

Definition:-

A DFA is a finite state machine where for each pair of current state and current input symbol there is a unique next state.

Representation of DFA:-

Mathematically DFA is five Tuples like

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

$Q \rightarrow$ finite and non-Empty sets of states.

$\Sigma \rightarrow$ finite and non-Empty sets of input symbols

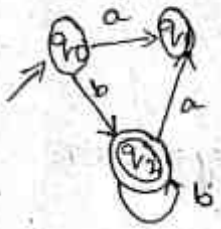
$\delta \rightarrow$ is a Transition function is defined as

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

$q_0 \rightarrow$ is the initial state.

$F \rightarrow$ is the final state.

Example of DFA:-



specification of DFA:-

1) Transition diagram 2) Transition table.

Acceptance of a string by DFA

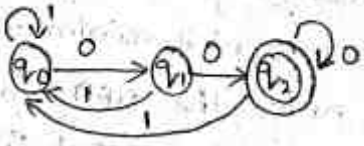
Consider the deterministic finite Automata 'M' and the string 'w' over input Alphabet Σ. Now, the string 'w' is Accepted by 'M' if and only if $L(M) = \{w | w \in \Sigma, \delta(q_0, w) = q_f\}$ methods for check whether the given string is accepted or not by DFA.

There are 3 methods.

1) Using sequence diagram
2) Using extended Transition function.
3) Using V dash function.

eg:- 1) Consider a DFA Now check whether the following strings are accepted or not by the DFA using.

1) sequence diagram 2) Transition function 3) V dash function.



1) 100 2) 1101 3) 100100.

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

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

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

δ:

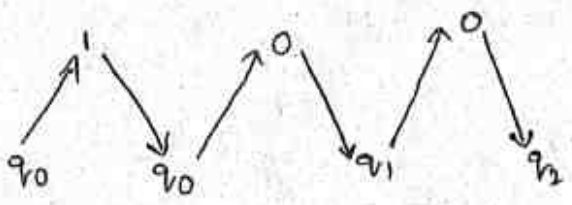
states	input	
	0	1
→ q_0	q_1	q_0
q_1	q_2	q_0
(q_2)	q_2	q_0

$q_0 = q_0$

$F = \{q_2\}$

1. $w = 100$

1) using sequence diagram



∴ The given string $w=100$ is accepted by the DFA

2) Using Extended transition function:-

$w=100$

$\delta(q_0, w) = \delta(q_0, 100)$

$= \delta(q_0, 00)$

$= \delta(q_1, 0)$

$= q_2 \in F$

∴ The given string is accepted by FA.

3) Using ⊢ dash function:-

$w=100$

$\vdash(q_0, 100) = \vdash M(q_0, 00)$

$= \vdash M(q_1, 0)$

$= q_2 \in F$

∴ The given string is accepted by FA.

** Design of DFA:-

procedure:- 1) understanding the language which is for designing a DFA.

2) Determine minimum length string in the language.

3) Draw the DFA for minimum length string.

4) Determine initial, Intermediate, dead and final states of DFA

5) Apply each input symbol on every state of DFA.

Eg:-

1) Design a DFA for the language which consits of set of all strings of 0's over $\Sigma = \{0\}$

let 'M' be a DFA with 5 tuples like

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

Given input $\Sigma = \{0\}$

$\therefore L(M) = \{\epsilon, 0, 00, 000 \ldots\}$

Minimum string is ϵ

The next minimum string is '0'.

∴ DFA

$Q = \{q_0\}$

$\Sigma = \{0\}$

δ : state	input 0
→ (q0)	q0

$q_0 = q_0$

$F = \{q_0\}$

2) Design a DFA for the language consist of set of all strings over accept empty string over $\Sigma = \{0\}$

Given input $\Sigma = \{0\}$

let M be a DFA like $M = (Q, \Sigma, \delta, q_0, F)$

$\therefore L(M) = \{0, 00, 000, 0000 \cdots\}$

Minimum string is '0'.

∴ DFA

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

$\Sigma = \{0\}$

δ: state	input 0
→ q0	q1
(q1)	q1

$q_0 = q_0$

$F = \{q_1\}$

3) construct a DFA that accepts a language over $\{0,1\}^*$

let 'M' be a DFA. like

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

The given alphabet $\Sigma = \{0,1\}^*$

$\therefore L(M) = \{\epsilon, 0, 1, 00, 01, 10, 11\}$

Minimum string = 'ε'

The Next Minimum string is '0 or 1'

∴ DFA

$q_0 = q_0$

$F = \{q_0\}$

$Q = \{q_0\}$

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

δ: state	input 0	1
→ (q0)	q0	q0

4) construct a DFA that accepts a language over set of $\{0,1\}^+$

let M be a DFA like

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

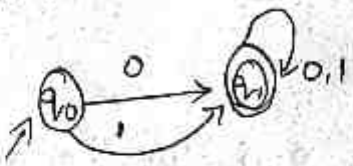
The given $\Sigma = \{0,1\}^+$

$\therefore L(M) = \{0, 1, 00, 01, 10, 11, \ldots\}$

The minimum string is 0 or 1.

$\therefore$ DFA $Q = \{q_0, q_1\}$

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



δ:

state	input 0	input 1
→ q_0	q_1	q_1
← q_1	q_1	q_1

$q_0 = q_0$

$F = \{q_1\}$

5) construct a DFA that accepts a language which contains set of all strings with even number of a's over $\Sigma = \{a\}$

⇒ Let M be a DFA like

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

The given input $\Sigma = \{a\}$

$\therefore L(M) = \{a^0, a^2, a^4, a^6, a^8, \ldots\}$

$= \{\epsilon, aa, aaaa, aaaaaa, \ldots\}$

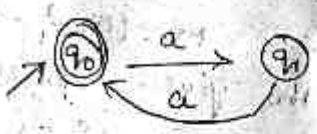
Minimum string is 'ε'

Next minimum string = aa

$\therefore$ DFA

$Q = \{q_0, q_1\}$

$\Sigma = \{a\}$



δ:

state	input a
→ q_0	q_1
q_1	q_1

$q_0 = q_0$

$F = \{q_1\}$

6) Construct a DFA that accepts a language which contains set of all strings with odd number of b's over $\Sigma = \{b\}$

⇒ let M be a DFA like

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

$\therefore$ The given $\Sigma = \{b\}$

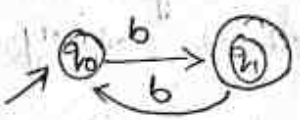
$\therefore L(M) = \{b^1, b^3, b^5, b^7, \ldots\}$

$= \{b, bbb, bbbbb, bbbbbbb, \ldots\}$

Minimum string is 'b'

$Q = \{q_0, q_1\}$

$\Sigma = \{b\}$



δ:

state	input b
→ q_0	q_1
(*) q_1	q_0

$q_0 = q_0$

$F = \{q_1\}$

** sum)

* construct a DFA for a language contains the following over $\Sigma = \{0,1\}$

i) set of all strings with even no. of 0's and even no. of 1's.

ii) set of all strings with even no. of 1's and odd no. of 0's.

iii) set of all strings with odd no. of 1's and even no. of 0's.

iv) set of all strings with odd no. of 1's and odd no. of 0's.

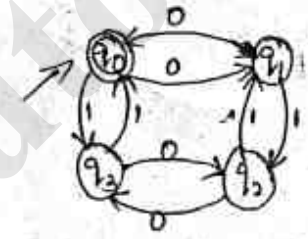
Sol:- $\Sigma = \{0,1\}$

1 - odd.

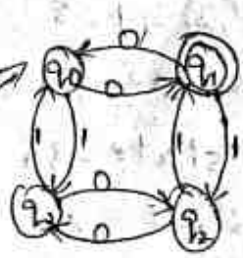
0 - even.

1	0	finalstates
0	0 = 0	q_0
0	1 = 1	q_1
1	0 = 2	q_2
1	1 = 3	q_3

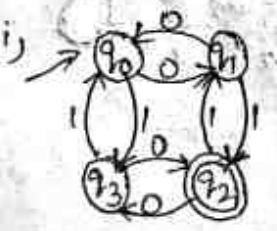
i)



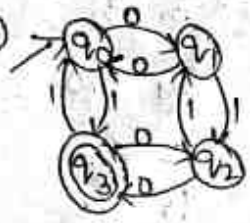
ii)



iii)



iv)



model-2

1) Design a DFA that accepts a language containing of all strings which are divisible by 3, where string is created as binary string.

Sol:- $\Sigma = \{0, 1\}$

$L(M) = \{$ All strings divisible by 3$\}$

∴ The possible remainders are

0, 1, 2

∴ The states are q_0, q_1, q_2

δ: initial state $= q_0$

final state $= q_0$

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

2) Construct a DFA for the language $L = \{w\}$ where $|w| \bmod 5$ that where string is created as ternary number.

Sol:- $\Sigma = \{0, 1, 2\}$

∴ $L(M) = \{$ All strings divisible by 5$\}$

∴ The possible remainders are

0, 1, 2, 3, 4, ∞

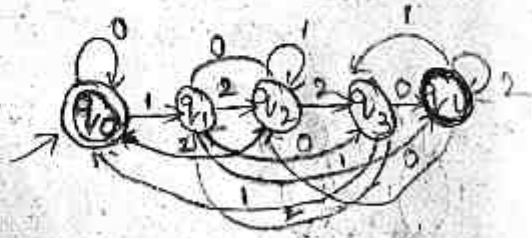
∴ The states are q_0, q_1, q_2, q_3, q_4

δ: initial state $= q_0$

final state $= q_0$

δ:

	0	1	2
→ q_0	q_0	q_1	q_2
q_1	q_3	q_4	q_0
q_2	q_1	q_2	q_3
q_3	q_4	q_0	q_1
← q_4	q_2	q_3	q_4



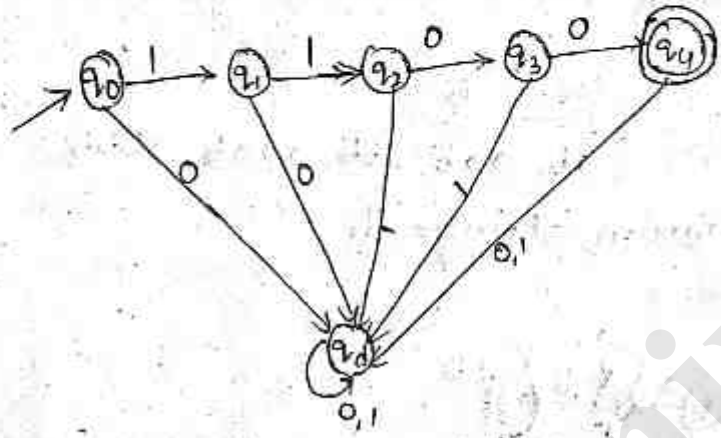
model 3

1) Construct a DFA that accepts a language which contains the string 1100 only over $\Sigma = \{0,1\}$

⇒ $\Sigma = \{0,1\}$

$L(M) = \{1100\}$

the minimum string = 1100



q_d is the invalid state

2) Design a DFA that accepts set of all strings that contains 0's and 1's and ends with 00 over $\Sigma = \{0,1\}$

Sol: Let 'M' be a DFA like

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

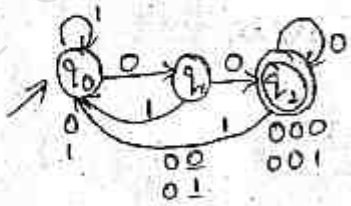
$\Sigma = \{0,1\}$

The string ends with '00'

$(0+1)^* 00$

$L(M) = \{00, 000, 100, 0000, 1100, 0100, 1000 \ldots\}$

minimum string = 00



δ:

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

3) Design a DFA that accepts a language of set of strings which are starts with 'a' and ends with 'b' over $\Sigma = \{a, b\}$

⇒ let 'M' be a DFA like

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

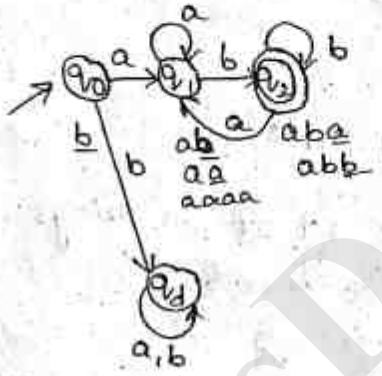
$\Sigma = \{a, b\}$

The strings starts with 'a' and ends with 'b'

$a\ \underline{(a+b)^*}\ b$

$L(M) = \{ab, aab, abb, aaab, abbb, ---\}$

minimum string = ab



δ:

	a	b
→ q_0	q_1	q_d
q_1	q_1	q_2
(*) q_2	q_1	q_2
q_d	this is a dead state	

4) Design a DFA that accepts a language of all strings which are starts with ab over $\Sigma = \{a, b\}$

Sol: Let 'M' be a DFA like

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

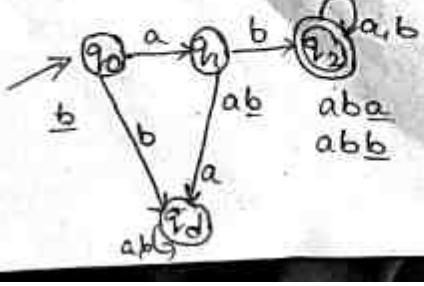
$\Sigma = \{a, b\}$

The strings starts with 'ab'

$ab\ \underline{(a+b)^*}$

$L(M) = \{ab, aba, abb, abaa, abbb, ---\}$

minimum string = ab.



δ:

	a	b
→ q_0	q_1	q_d
q_1	q_d	q_2
(*) q_2	q_2	q_2
q_d		

Model:-4

1) Design a DFA that accepts a language of set of all strings of a's and b's. which contains abb as a substring over $\Sigma = \{a, b\}$

=) Let 'M' be a DFA like

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

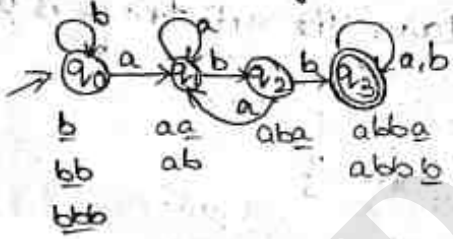
$\Sigma = \{a, b\}$

The strings with substrings as 'abb'

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

$L(M) = \{$ abb, aabbb, babb, abba, abbb, aaabb, bbabb, abbaa, abbbb, ... $\}$

Minimum string = abb.



δ:	a	b	b
$\rightarrow q_0$	q_1	q_0	q_0
q_1	q_1	q_2	q_2
q_2	q_1	q_3	q_3
(q_3)	q_3	q_3	q_3

2) Design a DFA over $\Sigma = \{a, b\}$ that contains set of strings of a's and b's Except those containing substring bba

=) Let 'M' be a DFA like

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

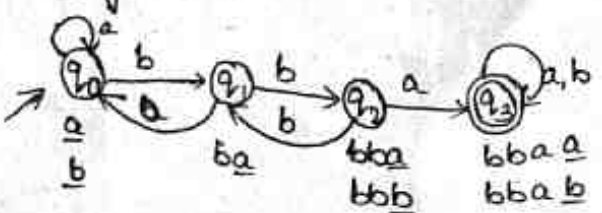
$\Sigma = \{a, b\}$

The strings with does not contains the substrings as 'bba'

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

$L(M) = \{$ bba, abba, bbba, bbaa, bbbba, ... $\}$

Minimum string = bba



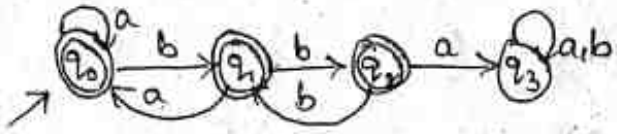
δ:	b	b	a
$\rightarrow q_0$	q_1	q_1	q_0
q_1	q_2	q_2	q_0
q_2	q_1	q_1	q_3
(q_3)	q_3	q_3	q_3

interchange

finalstate → non-final state

non-finalstate → final state

The required DFA is



3) Design a DFA over $\Sigma = \{0,1\}$ that contains set of strings are 0s and 1s and those strings does not containing the substring 001

⇒ Let M be a DFA like

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

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

Each string does not contains the substring is 001

$(0+1)^*$ 001 $(0+1)^*$

$L(M) = \{001, 0001, 1001, 0010, 0011 \cdots\}$

minimum string = 001

δ:	0	0	1
→ q_0	q_1	q_1	q_0
q_1	q_2	q_2	q_0
q_2	q_2	q_2	q_3
(*) q_3	q_3	q_3	q_3

interchange

finalstate → nonfinal state

nonfinal state → finalstate

The required DFA is

{w ∈ {a,b}* / w ≠ aa nor bb ... substr}

Non-Deterministic finite Automata:-

Introduction:

We cannot determine the next state exactly after reading an inputsymbol from a particular state then that FA is called NFA.

Definition:

NFA is a finite state machine whenever each pair of current state and particular input symbol it has more than one next state.

Elements:-

1) states 2) input symbols 3) Initial state 4) final state 5) Transitions.

Representation of NFA:-

Mathematically NFA is a five Tuple like $M=(Q, \Sigma, \delta, q_0, F)$

where Q = finite and non empty set of states.

Σ = Finite and non empty set of input symbols (or) input alphabets.

δ = It is a transition function which is defined as

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

q_0 = initial state it must be belongs to Q

F = finate state and $F \in Q$.

Description of NFA:-

It is of two ways i) Transition diagram ii) Transition table

Extended transition function:-

1. $\delta(q, \epsilon) = q$ 2. $\delta(q_i, a) = q_j$ where $q_i, q_j \in Q$

3. $\delta(q, xa) = \delta(\delta(q, x), a)$

language accepted by NFA:-

Mathematically language accepted by NFA is defined as

$L(M) = \{w / \delta(q_0, w) = q_f \in F\}$

Design of NFA:—

1) procedure ⇒ understanding the language which is for designing a NFA.
2) Determine minimum length string in the language.
3) Draw the NFA for minimum length string.
4) Determine initial, Intermediate, dead and final states of NFA.
5) Apply the each input symbol on intial and final states of NFA.

//

1) Design a NFA that accepts set of all strings over Σ that have atleast two consecutives 0's (or) 1's.

⇒ Let 'M' be a NFA like

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

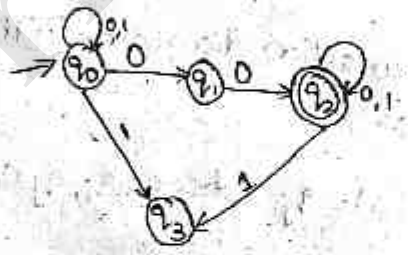
$\Sigma = \{0, 1\}$

each string has atleast two consecutives 0's or 1's

$(0+1)^*(00+11)(0+1)^*$

$L(M) = \{00, 11, 000, 01, 100, 111, \ldots\}$

minimum string = 00 (or) 11



δ:

states	input 0	input 1
→ q_0	$\{q_0, q_1\}$	$\{q_0, q_3\}$
q_1	q_2	–
q_2	q_2	q_3
q_3	–	q_2

2) Design an NFA to accept set of all strings starting with a followed by a or b and ending with a or any no. of b's over $\Sigma = \{a, b\}$

⇒ Let 'M' be NFA like

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

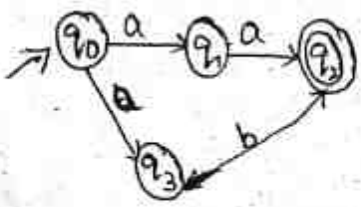
$\Sigma = \{a, b\}$

Each string starts with a and followed by 'a or b' and ending with 'a or any no. of b's.

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

$L(M) = \{aaa, aba, aa, ab, aaa, aaba, abaa, abab, \cdots\}$

Minimum string = aa or ab.



δ:

states	input a	b
→ q_0	$\{q_1, q_3\}$	—
q_1	q_2	—
(q_2)	—	q_3
q_3	—	q_2

3) Design an NFA that accepts set of all strings ending in 00 over $\Sigma = \{0, 1\}$

let – M be a NFA like

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

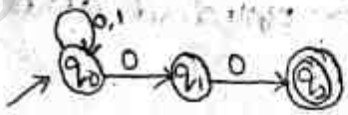
$\Sigma = \{0, 1\}$

Each string ends with 00

$(0+1)^*00$

$L(M) = \{00, 000, 100, 0000, 1100, \cdots\}$

Minimum string = 00



δ:

states	input 0	1
→ q_0	$\{q_0, q_1\}$	q_0
q_1	q_2	—
(q_2)	—	—

4) Design an NFA that accepts set of all strings ending in aba over $\Sigma = \{a, b\}$.

⇒ Let – M be a NFA like

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

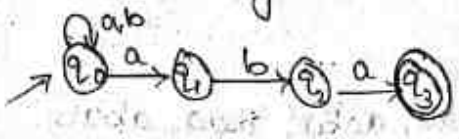
$\Sigma = \{a, b\}$

Each string ends with aba

$(a+b)^*$ aba

$L(M) = \{ aba, aaba, baba, aaaba, bbaba, \ldots \}$

Minimum string = aba



δ: states	input a	input b
→ q_0	$\{q_0, q_1\}$	q_0
q_1	—	q_2
q_2	q_3	—
(q_3)	—	—

5) Given an NFA which accepts all strings with ab over $\Sigma = \{a, b\}$

Let M be NFA like

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

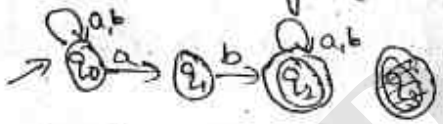
$\Sigma = \{a, b\}$

Each string have accepts all strings with ab

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

$L(M) = \{ ab, aaba, babb, \ldots \}$

minimum string = ab



δ: states	input a	input b
→ q_0	$\{q_0, q_1\}$	q_0
q_1	—	q_2
(q_2)	q_2	q_2

6) Construct an NFA that accepts all strings over $\Sigma = \{$ start with 00 and ends 01 or 10

⇒ Let M be a NFA like

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

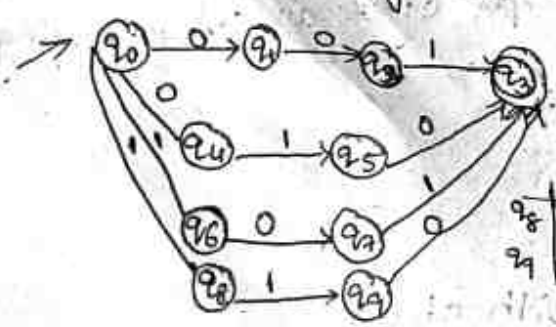
$\Sigma = \{0, 1\}$

Each string starts with 00 and ends with 01 or

$(0+1)$ $(0+1)^*$ $(01+10)$

$L(M) = \{ 001, 010, 101, 110, 0001, 0010, 0101, 0110, \ldots \}$

Minimum string = 001 or 010 or 101 or 110



states	Input 0	Input 1
→ q_0	$\{q_1, q_4\}$	$\{q_6, q_8\}$
q_1	q_2	—
q_2	—	q_3
(q_3)	—	—
q_4	—	q_5
q_5	q_3	—
q_6	q_7	—
q_7	—	q_3
q_8	—	q_9
q_9	q_3	—

7) Construct a transition system which can accept string over the alphabets a,b,c,---z containing either cat (or) Rat

⇒ Let M be a NFA like

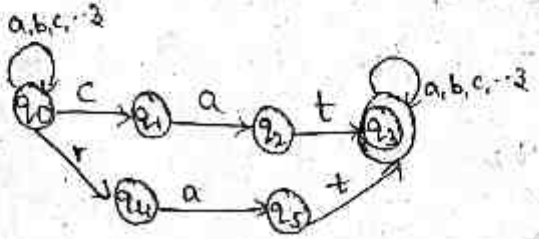
$M = (Q, \Sigma, \delta, q_0, F)$

$\Sigma = \{a,b,c,\dots z\}$

each strings containing either cat (or) RAT

$$(a+b+c+\cdots+z)^* (cat + rat) (a+b+c+\cdots+z)^*$$

Minimum string = cat or rat



δ:

states	input a	b	c	...	r	...	t	...	z
→ q_0	q_0	q_0	q_0		q_4		q_0		q_0
q_1	q_2								z
q_2							q_3		z
(*) q_3	q_3	q_3	q_3		q_3		q_3		q_3
q_4	q_5								
q_5							q_3		

Conversion of NFA to DFA:-

Algorithm:-

Let D be a DFA.

Let N be a NFA. This algorithm is called powerset (or) subset construction Algorithm. Because for NFA [N] with 'n' states then the corresponding DFA [D] can have 2^n states.

Subset construction Algorithm:-

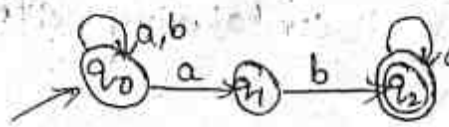
step 1:- construct the start state q_0, consisting of q_0 and all the states of NFA. that can be reached from q_0 by one or more transistions. Mark q_0 as unfinished.

2:- while there are unfinished states,

* Take an unfinished state S.
* for each $a \in \Sigma$, $\delta(s), a = t$ is either finished (or) unfinished state.
* Mark 'S' as finished.

3:- Mark all states that contain a final state from N. as final states of D.

Ex:- construct DFA from the Given NFA

⇒ 

Sol:- The given NFA N is like

$N = (Q, \Sigma, \delta, q_0, F)$ where

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

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

δ:

states	input a	input b
→ q_0	$\{q_0, q_1\}$	q_0
q_1	—	q_2
(*) q_2	q_2	q_2

$q_0 = q_0$

$F = \{q_2\}$

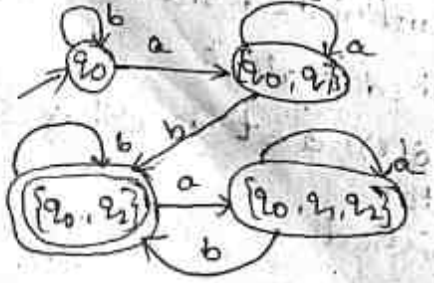
Let us consider the DFA 'D' is like

$D = (Q', \Sigma, \delta', q_0, F')$

Apply subset construction algorithm.

states	input a	input b
→ q_0	$\{q_0, q_1\}$	q_0
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
(*) $\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$
(*) $\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$

∴ The DFA is



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

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

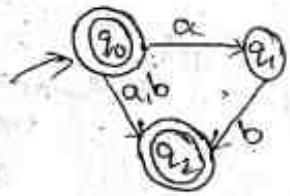
δ' =

states	inputs a	b
→ q_0	$\{q_0, q_1\}$	q_0
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$

$q_0' = q_0$

$F' = \{\{q_0, q_2\}, \{q_0, q_1, q_2\}\}$.

2) construct a DFA from the given NFA



⇒ The given NFA 'N' is like

$N = (Q, \Sigma, \delta, q_0, F)$ where

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

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

δ:

states	inputs a	b
→ q_0	$\{q_0, q_1\}$	q_2
q_1	ϕ	q_2
q_2	ϕ	ϕ

$q_0 = q_0$

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

Let us consider the DFA 'D' is like

$D = (Q', \Sigma, \delta', q_0, F')$

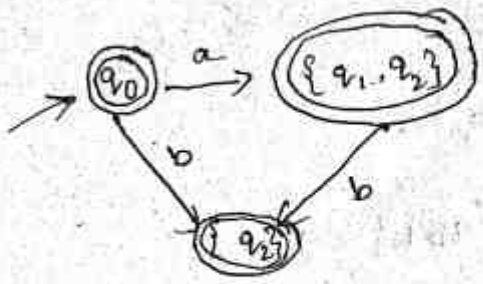
Apply subset construction algorithm

states	input a	b
→ q_0	$\{q_0, q_1\}$	q_2
$\{q_1, q_2\}$	ϕ	q_2
q_2	ϕ	ϕ

$q_0 = q_0$

$F' = \{\{q_1, q_2\}, \{q_0, q_1, q_2\}\}$

∴ The DFA is



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

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

$q_0 = q_0$

$f' = \{ \{q_1, q_2\}, q_0, q_2 \}$

δ':

state	inputs a	b
q_0	$\{q_1, q_2\}$	q_2
$\{q_1, q_2\}$	ϕ	q_2
q_2	ϕ	ϕ

NFA with ϵ-moves:

* Introduction * Representation * Elements
* External transition function * conversion of ϵ-NFA to NFA
* ϵ-closure.

Introduction:-

* A finite state machine which contains ϵ-moves is called ϵ-NFA
* ϵ-NFA is always NFA but not DFA.

Definition:-

* ϵ-NFA is a FSM where for each pair of current state and input symbol along with epsilon have more than one next state.

Elements:-

1) states 2) inputsymbols with ϵ 3) transitions
4) initialstate 5) finalstate.

Representation:-

ϵ-NFA is a five tuple like $M = (Q, \Sigma, \delta, q_0, F)$,

where ~~Q_0 is finir~~

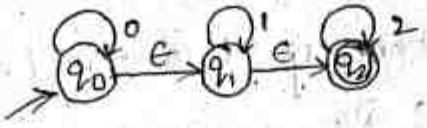
Q = is finite and non-empty set of states

Σ = finite and non-empty set of symbols.

δ = is a transition function which is defined as $\delta: Q \times \Sigma^* \to_2 Q$.

$Q \times \Sigma \cup \{\epsilon\} \to_2 Q$

Ex:-

1) Consider 

ε-closure (q_0) = $\{q_0, q_1, q_2\}$

ε-closure (q_1) = $\{q_1, q_2\}$

ε-closure (q_2) = $\{q_2\}$

***(14m) Conversion of NFA with ε-moves to NFA without ε-moves and equivalence:-

1) Converting NFA with ε-moves to NFA without ε-moves:-

In this method we remove the ε-transitions from the given NFA and obtained NFA without ε-moves

Algorithm:-

1) find out all ε-transitions from each state in Q. that will be called as ε-closure of q_i, where $q_i \in Q$.

2) δ' transitions can be obtained.

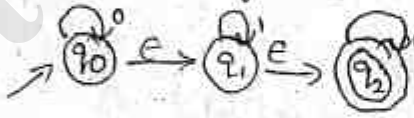
a) $\delta'(q, \epsilon)$ = ε-closure (Q)

*b) $\delta'(q, a)$ = ε-closure (δ(ε-closure(q), a))

3) Repeat step 2 for each input symbol and each state given NFA.

4) Finally the resultant states of NFA without ε-moves is obtained.

Ex:- Convert the given ε-NFA to NFA.



Sol:- Let M be a given ε-NFA like.

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

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

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

δ:- is a transition function.

states	input 0	1	2	ε
→ q_0	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
(*) q_2	ϕ	ϕ	q_2	ϕ

$q_0 = q_0$

$F = \{q_2\}$

Now find out ε-closure for all states in the given ε-NFA.

ε-closure(q_0) = $\{q_0, q_1, q_2\}$

ε-closure(q_1) = $\{q_1, q_2\}$

ε-closure(q_2) = $\{q_2\}$

Compute δ'-transitions :

$\delta'(q_0, 0)$ = ε-closure(δ(ε-closure(q_0), 0))

= ε-closure($\delta(\{q_0, q_1, q_2\}, 0)$)

= ε-closure($\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)$)

= ε-closure($q_0 \cup \phi \cup \phi$)

= ε-closure(q_0)

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

$\delta'(q_0, 1)$ = ε-closure(δ(ε-closure(q_0), 1))

= ε-closure($\delta(\{q_0, q_1, q_2\}, 1)$)

= ε-closure($\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)$)

= ε-closure($\phi \cup q_1 \cup \phi$)

= ε-closure(q_1)

= $\{q_1, q_2\}$

$\delta'(q_0, 2)$ = ε-closure(δ(ε-closure(q_0), 2))

= ε-closure($\delta(\{q_0, q_1, q_2\}, 2)$)

= ε-closure($\delta(q_0, 2) \cup \delta(q_1, 2) \cup \delta(q_2, 2)$)

= ε-closure($\phi \cup \phi \cup q_2$)

= ε-closure(q_2)

= $\{q_2\}$

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

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

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

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

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

$= \phi$

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

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

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

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

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

$= \{q_1,q_2\}$

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

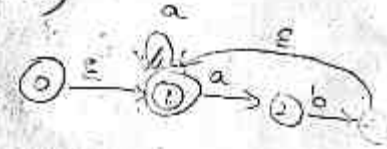
$= \epsilon\text{-closure}(\delta(\{q_1,q_2\},2))$

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

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

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

$= \{q_2\}$



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

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

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

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

$= \phi$

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

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

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

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

$= \phi$

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

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

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

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

$= \{q_2\}$

Now the NFA without ϵ-moves M' is like

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

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

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

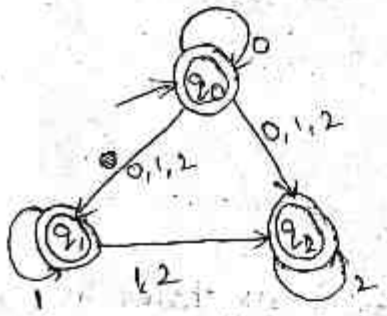
δ':

	0	1	2
→ q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$

$q_0 = q_0$

$F' = \{q_0, q_1, q_2\}$

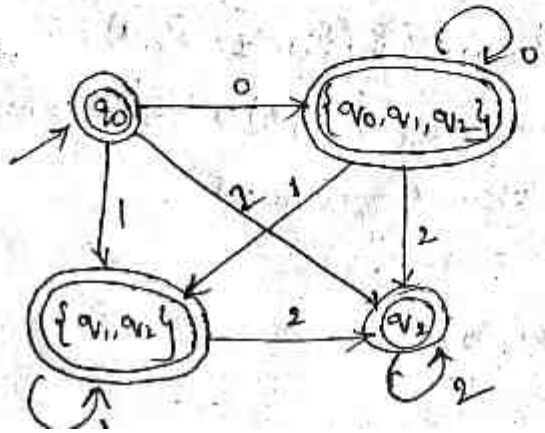
∴ NFA without ϵ-moves is



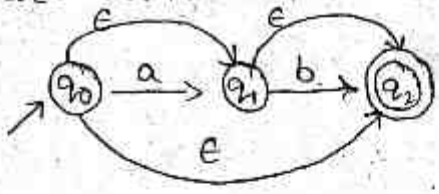
Converting NFA to DFA :-

	0	1	2
→ q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
$\{q_1, q_2\}$	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	q_2

∴ The DFA is



2) Construct DFA from the given NFA with ε-moves



→ Let M be a given NFA like.

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

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

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

δ: is a transition function

δ':

state	inputs a	b	ε
→ q_0	q_1	ϕ	$\{q_1, q_2\}$
q_1	ϕ	q_2	q_2
(*) q_2	ϕ	ϕ	ϕ

$q_0 = q_0$

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

We can find out the ε-closure of all states in the given ε-NFA.

ε-closure of $(q_0) = \{q_0, q_1, q_2\}$

ε-closure of $(q_1) = \{q_1, q_2\}$

ε-closure of $(q_2) = \{q_2\}$

Compute δ'-transitions:-

$\delta'(q_0, a) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0), a))$

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

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

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

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

$= \{q_1, q_2\}$

$\delta'(q_0, b) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0), b))$

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

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

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

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

$= \{q_2\}$

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

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

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

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

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

$= \phi$

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

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

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

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

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

$= \{q_2\}$

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

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

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

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

$= \phi$

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

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

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

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

$= \phi$.

Now the NFA without ε-moves M' is like.

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

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

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

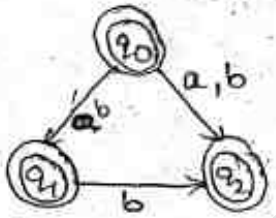
δ' :

state	inputs a	b
→ q_0	$\{q_1, q_2\}$	q_2
q_1	ϕ	q_2
q_2	ϕ	ϕ

$q_0 = q_0$

$F' = \{q_0, q_1, q_2\}$

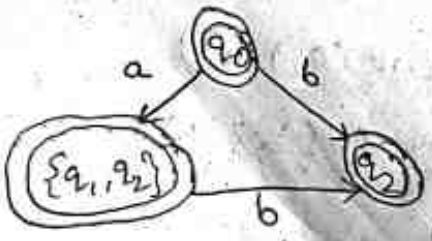
NFA without ε-moves is



Converting NFA to DFA

states	Inputs a	b
→ q_0	$\{q_1, q_2\}$	q_2
$\{q_1, q_2\}$	ϕ	q_2
q_2	ϕ	ϕ

The DFA is



Equivalence of finite state machines :-

Two finite automata's are equivalent if they accept the same set of strings over 'L'. Otherwise they are not equivalent

Algorithm :-

1) We can check the equivalence of two finite states machine by using comparison table.

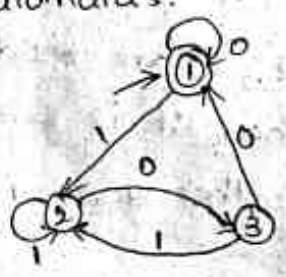
Comparison table :-

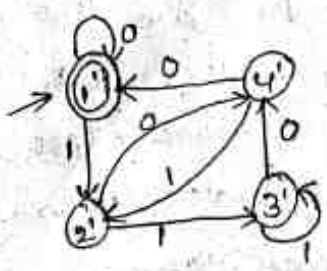
* It is similar to transition table.
* It contains (n+1) columns.
* Here, first column represents states, n column represents input symbols.
* Entries are pair of states. Here pair of states both are final states (or) both are non-final states.

Construction of Comparison table :-

* Comparison table is starts with initial states of M and M' in the first column and consider the entry as (q_0, q_0') where, $\delta(q_0, a) = q_0$ and $\delta(q_0', a) = q_0'$.
* Repeat step 1. for each input symbol on every pair of states of M and M' until no new pairs appeared.
* If you get any pair (q_0, q_0') such that q_0 is a final state and q_0' is non-final state (or) vice versa. then we terminate the process and conclude that both M and M' are not equivalent.

Ex :- check the Equivalence of the following finite automata's.





Comparison table:-

states	Input symbols	
	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')

∴ The given 2 Finite automates are Equivalence.

Minimization (or) optimization of FA :-

The process of reducing the no.of states from given FA is called minimization (or) optimization of FA.

Equivalence states:-

The two states q_1 and q_2 are Equivalent if both $\delta(q_1,a)$ and $\delta(q_2,a)$ are Final states (or) non-final states. While minimizing finite automata we first find out which two states are Equivalent then we can represent these two states by one representative state.

Minimization Algorithm:-

* we will create a set $\Pi_0 = \{\{Q_1^0\}\ \{Q_2^0\}\}$ where $\{Q_1^0\}$ is the set of final states and $\{Q_2^0\}$ is the set of non-final states

Π_0 is 0' equivalence class.

* Now we will construct Π_{K+1} from Π_K.

Let Q_i^K be any subset in Π_K

If q_1 and q_2 are two states in Q_i^K.

Find out whether $\delta(q_1,a)$ and $\delta(q_2,a)$ are residing in the same Equivalence class in Π_K. Then it is said that $q_1 \& q_2$ are K+1 Equivalent then Q_i^K is further divided into 'K+1' Equivalence classes.

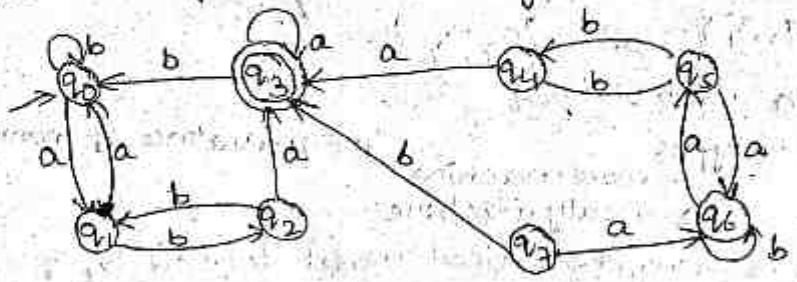
* Repeat step 2. for every Q_i^K in Π_K and obtain all elements in Π_{K+1}.

* continue the above process until $\pi_n = \pi_{n+1}$ where $n = \{1, 2, 3, \ldots\}$

* Then replace all the equivalence states in one equivalence class find representing states

this helps minimizing the given finite automata

Ex:- construct the state minimizing the Finite automata for the following transition diagram



Sol:- The given FA, M is like

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

where

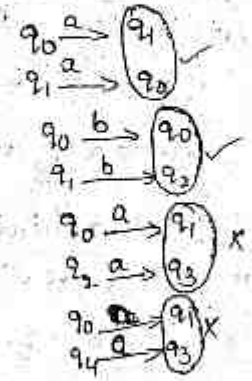
$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$

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

δ is a transition function

states	Input symbols a	b
→ q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
(q_3)	q_3	q_0
q_4	q_3	q_5
q_5	q_6	q_4
q_6	q_5	q_6
q_7	q_6	q_3

rough



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

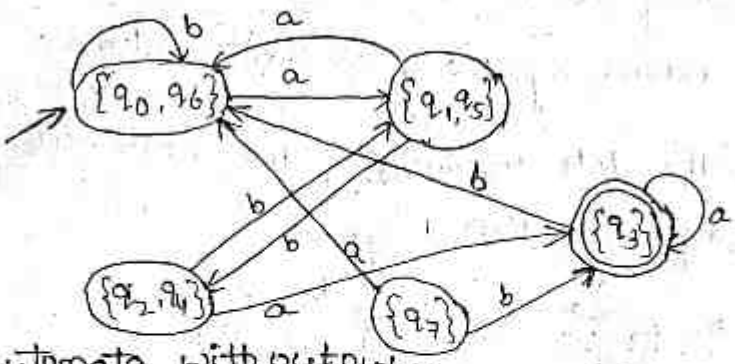
$\pi_1 = \{\{q_3\}, \{q_0, q_1, q_5, q_6\}, \{q_2, q_4, q_7\}\}$

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

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

$\therefore \Pi_3 = \Pi_2$

$\therefore$ The minimized FA



Finite automata with output :-

* Introduction * types
  1. moore machine
  2. mealy machine
* Equivalence of moore and mealy machine.

* Introduction :-

We know FA is a mathematical model defined by 5 tuples like $M = (Q, \Sigma, \delta, q_0, F)$.

without information about the output.

After reading a string if finite automata reaches to the final state then the string is accepted by FA. Else the string is registered.

FA without output is called language acceptors.

Transducers :-

* The FA with output is called Transducers.
* It doesnot contain final states.
* It cannot be used for language Acceptor.
* It provides the information about output
* It can be used for type conversion of language
* Transducers are two types.
  i) Moore machine
  ii) mealy machine.

i) Moore machine :-

moore machine is a FA that contains set of states in which the output is always depends on present state only.

present state → output.

mathematically moore machine is 6 tuples like.

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

→ where Q = finite and non empty set of states.

Σ = finite and non-empty set of Input symbol.

δ = It is a transition function defined as

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

Δ = It is a finite and non-empty set of output symbols(or) output Alphabets.

λ = It is a output function which is defined as

$$\lambda: Q \to \Delta$$

q_0 = It is a initial state and must be $q_0 \in Q$.

Representation of moore machine:-

i) Transition diagram ii) transition table

Transition diagram:-

Transition table:-

present state	Input symbols a	b	c	d	output

Ex: Design a moore machine for residue 131 for the input string is created as a binary number.

sol:- Given $\Sigma = \{0,1\}$

residue 131 means if any number is divisible by 3 then we get the possible remainders are 0,1,2.

$\therefore \Delta = \{0,1,2\}$

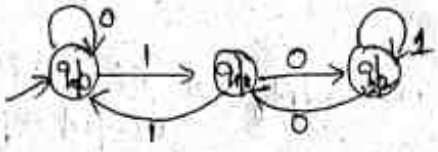
The possible states are $\{q_0, q_1, q_2\}$

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

$\therefore \delta$: Transition function is defined as

δ:	op/i/p 0	1
$\to q_0$	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

∴ The moore machine is:



Transition table :-

present state	I/p symbols		output
	0	1	
→ q_0	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_1	q_2	2

Note :- In moore machine n+1 output symbols can be produced as 'n' input symbols.

Mealy machine :-

It is a finite automata contains set of states in which "the output is always depends on present state and i/p symbol".

Mathematically mealy machine defined by 6 tuples like $M = (Q, \Sigma, \delta, \Delta, \lambda, q_0)$

where,

Q → finite and non-empty set of states.

Σ → finite and non-empty set of states and input symbols.

δ is a transition function is defined as

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

Δ is a finite and non empty set of states and output symbols.

λ is a output function is defined as,

q_0 is the Initial state must be $q_0 \in Q$.

Representation :- In two ways

1. Transition diagram 2. Transition Table

① Transition diagram

2. Transition Table

present state	Input symbols,		Input symbols 2	
	Next state	output	Next state	output

Design a mealy machine for residue mode 3 in which the input is treated as a binary machine.

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

Residue mode 3 means if any number is divisible by 3 then the possible remainders are 0, 1, 2

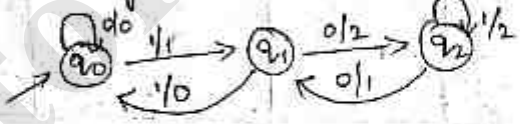
$\Delta = \{0,1,2\}$

The possible states are $\{q_0, q_1, q_2\}$

$\therefore \delta$ is transition function

δ: state	Input symbols 0	1
→ q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

The mealy machine is



Transition table

present state	input symbol, (0) Next state	output	Input symbol, (1) Next state	output
→ q_0	q_0	0	q_1	1
q_1	q_2	2	q_0	0
q_2	q_1	1	q_2	2

Note:- In mealy machine n' output symbols can be produced for 'n' input symbols.

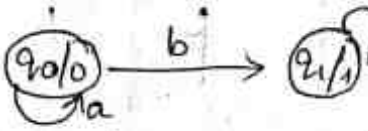
Equivalence of Mealy and Moore Machine:-

Conversion of Moore to Mealy machine

Conversion of Mealy to Moore machine

Conversion of Moore and Mealy machine:-

Convert the following Moore machine to Mealy machine.



The given Moore machine is 'M' like

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

$Q = \{q_0, q_1\}$

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

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

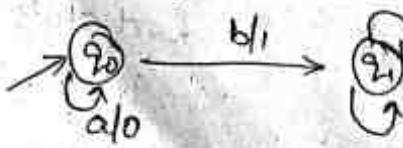
λ : output function.

present state	Input symbols		output
	a	b	
→ q_0	q_0	q_1	0
q_1	q_1	q_1	1

Now the transition table for Mealy machine is

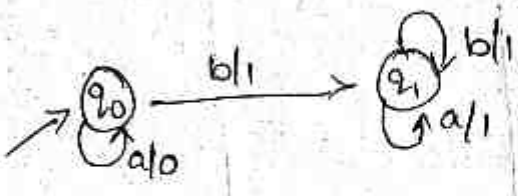
present state	Input symbol, (a)		Input symbol, (b)	
	Next state	output	Next state	output
→ q_0	q_0	0	q_1	1
q_1	q_1	1	q_1	1

Transition diagram for Mealy machine is



conversion of mealy to moore machine:-

Convert the following mealy machine to moore machine.



The given mealy machine is M like

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

$Q = \{q_0, q_1\}$

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

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

δ:

present state	a		b	
	N.S	o/p	N.S	o/p
→ q_0	q_0	0	q_1	1
q_1	q_1	1	q_1	1

δ: a b

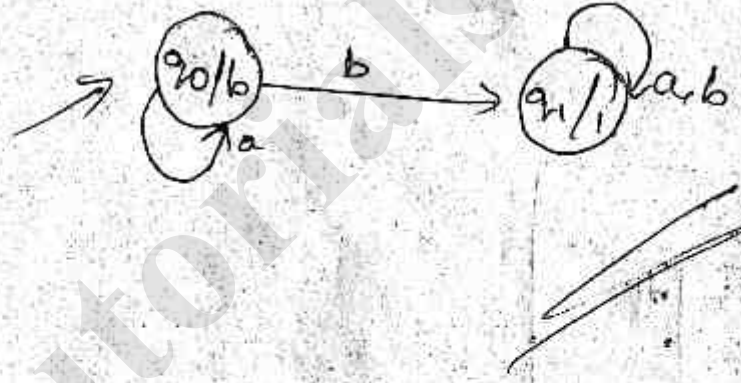
q_0 q_0 q_1

q_1 q_1 q_1

Now moore machine table.

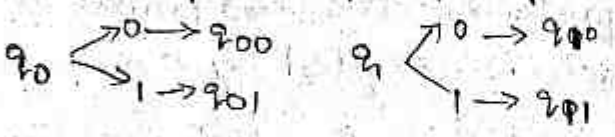
Present state	Input symbol		output
	a	b	
→ q_0	q_0	q_1	0
q_1	q_1	q_1	1

Transition diagram for moore machine is



convert the following mealy machine to moore machine

P.S	a		b	
	N.S	o/p	N.S	o/p
→q_0	q_0	0	q_1	1
q_1	q_0	1	q_1	0



Transition table for moore machine:

p.s	a	b	output
→q_{00}	q_{00}	q_{11}	0
→q_{01}	q_{00}	q_{11}	1
q_{10}	q_{01}	q_{10}	0
q_{11}	q_{01}	q_{10}	1

Applications and limitations of FA:-

limitations:-

* FA contains and i/p buffer with limited (or) finite no.of locations. i.e; it has a limited amount of memory for storing i/p data.
* It recognise a finite length of i/p string.
* It can moves its read/write head in either left to right (or) right to left direction only.

Applications:-

* FA is used to design lexical analyser.
* FA is used to create text editors.
* FA is used to spell checking.
* FA is used to design sequential circuits.

UNIT-II

Regular Expressions:

* Introduction
* Examples
* Components
* operations.

Introduction:- Regular expressions are mathematical Exp describing a language which is accepted by FA.

* R.E describing a language called Regular language.

Definition:-

let ξ be an alphabet then R.E over ξ is defined as follows.

* ϕ is a R.E then that describes an Empty set. RE=ϕ
* ϵ is a R.E then that describes "NULL" string set. RE=ϵ, RE=
* a is a R.E over ξ then that describes the set with a RE=
* Let r, and s are two R.E and L_1 and L_2 are two languages whi RE= are described by r and s then,
  - i) R+s is Equivalent to $L_1 \cup L_2$
  - ii) rs is Equivalent to $L_1 L_2$ (or) $L_1 \cdot L_2$
  - iii) r^* is Equivalent to L_1^*

Components:-

union, concatenation, Kleene closure.

Examples:-

1) Write the R.E for the language accepting all Combinati of a's over $\xi = \{a\}$

Sol:- $\xi = \{a\}$

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

$= a^*$

$\therefore RE = a^*$

2) Write a R.E for the language accepting all combinatio of a's except empty string over $\xi = \{a\}$

Sol:- $\xi = \{a\}$ $\{a, aa, aaa, aaaa, \ldots\}$

$= a^+$

$\therefore R.E = a^+$

3) Write a R.E for the language accepting any no. of a's and b's over $\xi = \{a, b\}$

Sol:- $\xi = \{a, b\}$

$L = \{\epsilon, a, b, a^2, ab, ba, b^2, \ldots\}$

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

$= (a+b)^*$

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

4) Write a R.E for the language containing all strings which are ending with 00 over $\xi = \{0, 1\}$.

Sol:- $\xi = \{0, 1\}$

$L = \{(0+1)^* 00$

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

5) Write a R.E for the language Accepting set of all string which are starting with '1' and ending with '0' over $\xi = \{0, 1\}$

Sol:- $\xi = \{0, 1\}$

$1 (0+1)^* 0$

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

6) Write a R.E for the language accepting any no. of a's followed by any no of bs, followed by any no. of c's over $\xi = \{a, b, c\}$

Sol: $\xi = \{a, b, c\}$

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

7) Write a R.E for the language accepting set of all strings which contains the third character from the right end of the string is always 'a' over $\xi = \{a, b\}$. $\therefore R.E = (a+b)^* a (a+b)(a+b)$

Sol: $\xi = \{a, b\}$

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

language Associated with R.E :-

* A language which is described by re is called Regular language.
* Let R be a R.E then the language accepted by R.E is denoted by L(R).

Ex:- If the R.E $R = (ba)^*$ then $L(R) = ?$

Sol:- $R = (ba)^*$

$L(R) = \{(ba)^0, (ba)^1, (ba)^2, (ba)^3, ---\}$

$= \{\epsilon, ba, baba, bababa, ---\}$

Properties of RE (Identity rules):

Let R, S be R.E then the following properties are true

1) $(R+S)+T = R+(S+T)$
2) $R+R = R$
3) $R+\phi = \phi+R = R$
4) $R\phi = \phi R = \phi$
5) $R+S = S+R$
6) $R\epsilon = \epsilon R = R$
7) $R(ST) = (RS)T$
8) $R(S+T) = RS+RT$
9) $(S+T)R = SR+TR$
10) $\phi^* = \epsilon^* = \epsilon$
11) $R^*R^* = R^* = (R^*)^*$
12) $RR^* = R^*R = R^+$
13) $(R+S)^* = (R^*S^*)^* = (R^*+S^*)^*$
14) $(R.S)^* = (R^*+S^*)^* = (R^*S^*)^*$

Manipulation of R.E (Basic operations)

i) union ii) concatenation iii) Kleene closure.

UNION:-

Let R & S be two R.E then union of R & S is defined as

$R \cup S = \{x \mid x \in R \text{ or } x \in S\}$

Ex:- If $R = \{ab, c\}$ and $S = \{d, ef\}$ then $R \cup S = ?$

$R \cup S = \{ab, c, def\}$

concatenation:-

Let R & S be two R.E then concatenation of R & S is defined as $RS = \{xy \mid x \in R \text{ and } y \in S\}$

Ex:- If R = {ab, c} and S = {d, ef} then RS = ?

Sol: RS = {ab, c}{d, ef}

= {abd, abef, cd, cef}

Kleene closure:-

Let R be a RE then the kleene closure of R is denoted as R^* which contains set of all strings including Null string.

Ex:- If R = {ab} then R^* = ?

Sol:- $R^* = \{(ab)^0, (ab)^1, (ab)^2, (ab)^3 - -\}$

$= \{\epsilon, ab, abab, ababab, - -\}$

Examples:

Construct a string set for the RE given below.

i) ab^*a

Sol: $\{ab^0a, ab^1a, ab^2a, - -\}$

$\{aa, aba, abba, - - -\}$

ii) 1^*0

$\{1^0, 1^1, 1^2, 1^3, 1^4, - - -\}0$

$= \{\epsilon, 1, 11, 111, 1111, - - -\}0$

$= \{0, 10, 110, 1110, 11110, - - -\}$

iii) 00^*

$= \{00^0, 00^1, 00^2, 00^3, 00^4, - -\}$

$= \{0, 00, 000, 0000, 00000, - - -\}$

$= 0^+$

iv) $(100^+)^*$

$= (10\{0^1, 0^2, 0^3, - -\})^*$ $\therefore \{\epsilon, 100, 100100, - - - 1000, 1000100, - - - 10000, 10000010000 - -\}$

$= (100, 1000, 10000, - - -)^*$

$= ((100)^*, (1000)^*, (10000)^*)$

$= (\{\epsilon, 100, 100100, 100100100, - -\}, \{\epsilon, 1000, 1000001000, 1000001000000, - - -\}, \{\epsilon, 10000, 100000010000, - -\}$

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

$= \{\epsilon, 0, 00, 000, \ldots\} \cup \{\epsilon, 1, 11, 111, 1111, \ldots\}$

$= \{\epsilon, 0, 00, 000, \ldots 1, 11, 111, 1111, \ldots\}$

vi) $(0+1)^* 011$

$= (0^* + 1^*)011$

$= ((\{\epsilon, 0, 00, 000, \ldots\} \cup \{\epsilon, 1, 11, 111, \ldots\})011)$

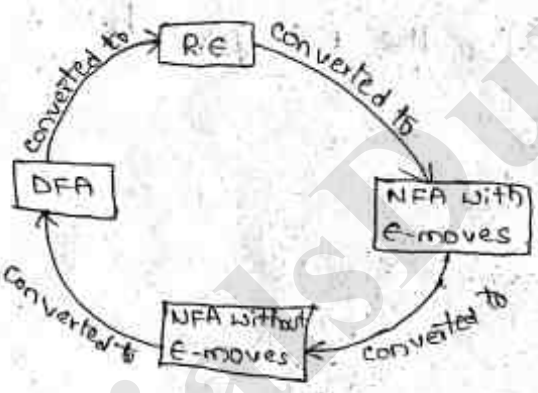
$= ((\{\epsilon, 0, 00, 000, \ldots 1, 11, 111, \ldots\})011)$

$= \{011, 0011, 00011, 000011, \ldots 1011, 11011, 111011, \ldots\}$

Equivalence of R.E and FA:-

1. Conversion of R.E to FA
2. Conversion of FA to R.E

Relationship b/w R.E and FA:



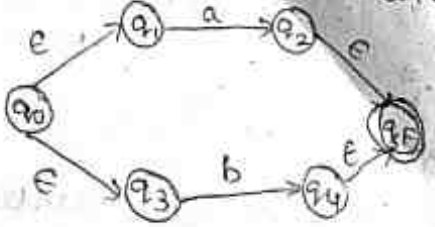
1. Conversion of R.E to FA:

Basic notations used

1. If the RE is like $r = \epsilon$ then FA is

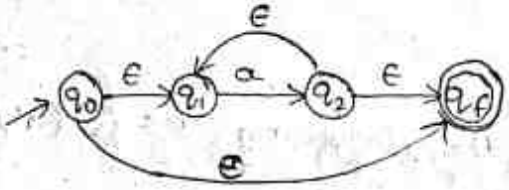
2. If the R.E is like $r = a$ then FA is

3. If the R.E is like $r = a+b$ then FA is



4. If the R.E is like $r = ab$ then FA is

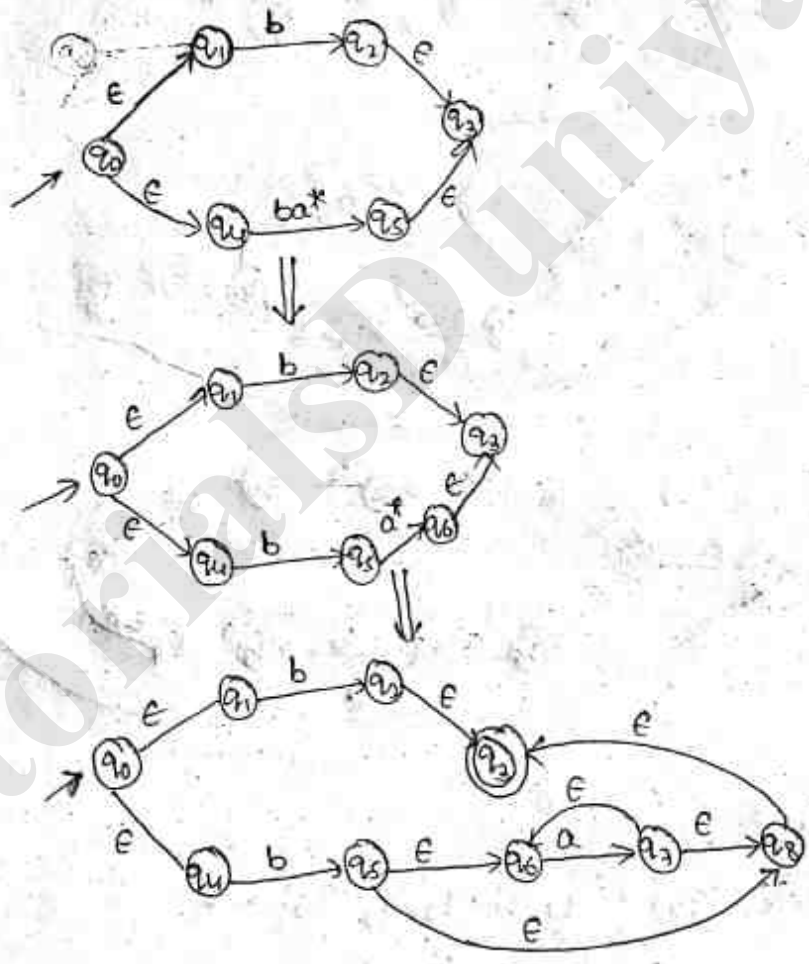
5. If the R.E is like $r = a^*$ then FA is



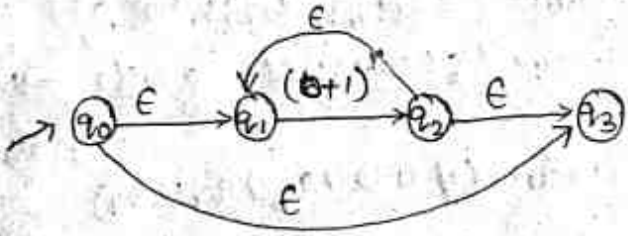
Examples:

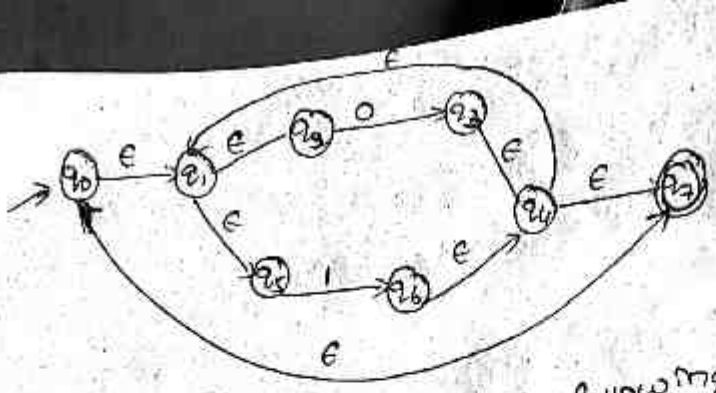
1) Construct an NFA with ε-moves for the R.E $b+ba^*$

Sol: The given RE $r = b+ba^*$



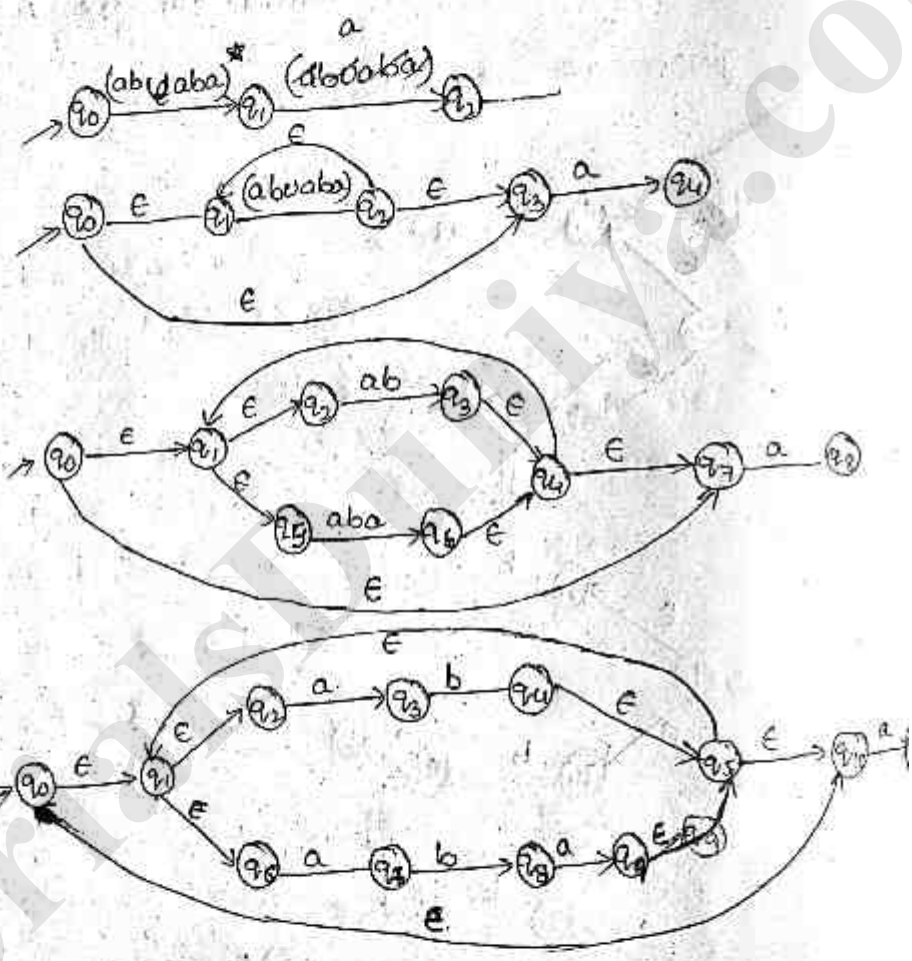
2) Construct an NFA with ε-moves for the RE is $(0+1)^*$





3) Construct a DFA for the following R.E is $(abu(aba)^*)^*a$

The given R.E is $(abu(aba)^*)^*a$



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

ϵ-closure $(q_0) = \{q_0, q_1, q_2, q_8, q_{10}\} = A$

$\delta'(A, a) = \epsilon$-closure $(\delta(A, a))$

$= \epsilon$-closure $(\delta(\{q_0, q_1, q_2, q_8, q_{10}\}, a))$

$= \epsilon$-closure $(\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_8, a) \cup \delta(q_{10}, a))$

$= \epsilon$-closure $(\phi \cup \phi \cup q_3 \cup q_7 \cup q_{11})$

$= \epsilon$-closure $(q_3) \cup \epsilon$-closure $(q_7) \cup \epsilon$-closure(

$= \{q_3\} \cup \{q_7\} \cup \{q_{11}\}$

$= \{q_3, q_7, q_{11}\} = B$

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

$= \epsilon\text{-closure}(\delta(\{q_0, q_1, q_2, q_6, q_{10}\}, b)$

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

$= \phi$

$\delta'(B,a) = \epsilon\text{-closure}(\delta, (B,a))$

$= \epsilon\text{-closure}(\delta(\{q_3, q_7, q_{11}\}, a))$

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

$= \phi$

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

$= \epsilon\text{-closure}(\delta(\{q_3, q_7, q_{11}\}, b))$

$= \epsilon\text{-closure}(q_4 \cup q_8 \cup \phi)$

$= \epsilon\text{-closure}(q_4) \cup \epsilon\text{-closure}(q_8)$

$= \epsilon\text{-closure}\{q_4, q_5, q_{10}, q_1, q_2, q_6\} \cup \{q_8\}$

$= \{q_1, q_2, q_4, q_5, q_6, q_8, q_{10}\}$

$= C.$

$\delta'(C,a) = \epsilon\text{-closure}(\delta(C,a))$

$= \epsilon\text{-closure}(\delta(\{q_1, q_2, q_4, q_5, q_6, q_8, q_{10}\}, a))$

$= \epsilon\text{-closure}(\phi \cup q_3 \cup \phi \cup \phi \cup q_7 \cup q_9 \cup q_{11})$

$= \epsilon\text{-closure}(q_3) \cup \epsilon\text{-closure}(q_7) \cup \epsilon\text{-closure}(q_9) \cup \epsilon\text{-closure}(q_{11})$

$= \{q_3\} \cup \{q_7\} \cup \{q_9, q_5, q_{10}, q_1, q_2, q_6\} \cup \{q_{11}\}$

$= \{q_1, q_2, q_3, q_5, q_6, q_7, q_9, q_{10}, q_{11}\}$

$= D$

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

$= \epsilon\text{-closure}(\delta(\{q_1, q_2, q_4, q_5, q_6, q_8, q_{10}\}, a))$

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

$= \phi$

$\delta'(D,a) = \epsilon\text{-closure}(\delta(D,a))$

$= \epsilon\text{-closure}(\phi \cup q_3 \cup \phi \cup \phi \cup q_7 \cup \phi \cup \phi \cup q_{11} \cup \phi)$

$= \epsilon\text{-closure}(q_3) \cup \epsilon\text{-closure}(q_7) \cup \epsilon\text{-closure}(q_{11})$

$= \{q_3\} \cup \{q_7\} \cup \{q_{11}\}$

$= \{q_3, q_7, q_{11}\}$

$= B.$

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

$= \epsilon\text{-closure}(\phi \cup \phi \cup q_4 \cup \phi \cup \phi \cup q_8 \cup \phi \cup \phi \cup \phi)$

$= \epsilon\text{-closure}(q_4) \cup \epsilon\text{-closure}(q_8)$

$= \{q_4, q_5, q_1, q_{10}, q_2, q_6\} \cup \{q_8\}$

$= \{q_1, q_2, q_4, q_5, q_6, q_8, q_{10}\}$

$= C.$

The previous NFA with ϵ-moves has the Initial state has q_0 and final state as q_{11}.

Now the DFA has initial state A because it contains q_0 as an element which can be initial state NFA.

The Final state of DFA is B, D because both states contain q_{11} as an element. The DFA 'M' is like,

$M = \{Q, \Sigma, \delta, B_0, F\}$ where

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

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

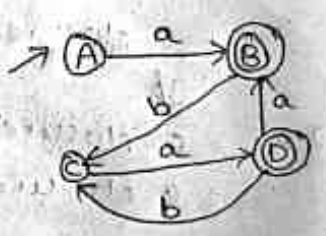
δ: is a transition function which is defined as

δ:	a	b
→ A	B	ϕ
(B)	ϕ	C
C	D	ϕ
(D)	B	C

$q_0 = a$

$f = \{b, d\}$

∴ transition diagram for DFA is.



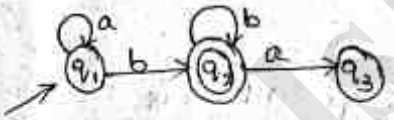
2nd method:
conversion of FA to R.E.

ARDEN'S Theorem :-
Let p and Q be two R.E. over the input alphabet Σ.
The R.E 'R' is given as $\boxed{R = Q + RP}$ which has a unique solution. $\boxed{R = QP^*}$

conversion algorithm:

* Let q_1 be a initial state.
* There are $q_2, q_3, q_4, \ldots q_n$ are no. of states. The final state may be some q_j where $j \le n$.
* Let α_{ij} be a transition from q_i to q_j state
* calculate $q_j = \alpha_{ij} q_i$. If q_j is an initial state then
  $q_j = \alpha_{ij} \cdot q_i + \epsilon$
* Similarly Compute the final state equation which gives the R.E.

Ex:- Construct R.E from given DFA.



Sol:
$q_1 = aq_1 + \epsilon \longrightarrow (1)$
$q_2 = bq_1 + bq_2 \longrightarrow (2)$
$q_3 = aq_2 \longrightarrow (3)$
$q_1 = aq_1 + \epsilon$ $\left(\because \begin{matrix} R = Q + RP \\ \Downarrow \\ R = QP^* \end{matrix} \right)$
where
$R = q_1$
$Q = \epsilon$
$P = a$
solution for above equ. is,
$R = QP^*$
$q_1 = \epsilon a^*$
$q_1 = a^* \longrightarrow (4)$
substitute equ (1) in equ (2)
$q_2 = bq_1 + bq_2$
$q_2 = ba^* + bq_2$

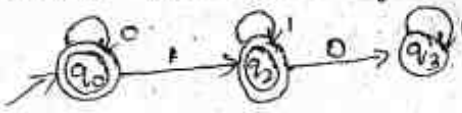
$q_2 = ba^* b^*$

$= a^* bb^*$

$= a^* b^*$

The R.E is $\boxed{a^* b^*}$

Construct R.E from given DFA



Sol: $q_1 = 0q_1 + \epsilon \longrightarrow (1)$

$q_2 = 1q_1 + 1q_2 \longrightarrow (2)$

$q_3 = 0q_3 + 0q_3 + 1q_3 \longrightarrow (3)$

$q_1 = 0q_1 + \epsilon$

$q_1 = \epsilon + 0q_1 \quad (\because R = Q + RP \Rightarrow R = QP^*)$

$q_1 = \epsilon 0^*$

$q = 0^* \longrightarrow (4)$

sub Equ (4) in Equ (2)

$q_2 = 1q_1 + 1q_2$

$q_2 = 10^* + 1q_2$

$q_2 = 10^* 1^*$

$= 0^* 11^*$

$= 0^* 1^*$

∴ The regular expression for given DFA is

$q_1 + q_2 = 0^* + 0^* 1^*$

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

$= 0^* 1^*$

Construct R.E from given DFA.

Sol: $q_1 = \epsilon \longrightarrow (1)$

$q_2 = 0q_1 + 0q_3 \longrightarrow (2)$

$q_3 = 0q_2 \longrightarrow (3)$

sub Equ (1) in Equ (2)

$q_2 = 0q_1 + 0q_3$

$= 0 \cdot \epsilon + 0 \cdot q_3$

$q_2 = 0 + 0q_3 \longrightarrow (4)$

sub Eqn ③ in Eqn ④

$q_2 = 0 + 0q_3$

$q_2 = 0 + 0 \cdot 0q_2$ $(R = Q + RP \Rightarrow R = QP^*)$

$q_2 = 0(00)^*$

The R.E for given DFA is

$0(00)^*$ //

Applications of R.E :-

* RE are used for describing a language called Regular language.
* R.E are used to implement lexical analysis in Compiler design.
* R.E are used to represent a set of strings in UNIX programming.

closure properties of Regular languages :-

* Regular languages are closed under Union.
* " " " " " Intersection
* " " " " " Concatenation
* " " " " " Kleene closure
* " " " " " Difference
* " " " " " Complement
* " " " " " Reverse
* " " " " " symetric difference
* " " " " " Homomorphism
* " " " " " Inverse Homomorphism

Regular Grammar and FA :-

* Grammar
* Regular Grammar
  1. left linear Grammar
  2. Right linear Grammar
* FA from Regular Grammar
* RG from FA
* RE from RG.

Grammar :-

Grammar is a set that contains four types like.

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

Regular grammar from FA:-

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a FA. It contains some no. of states like $Q = \{q_1, q_2, \ldots q_n\}$ and $\Sigma = \{a_1, a_2, a_3 \ldots a_n\}$

Therefore the Regular grammar $G = (V, T, P, S)$ is defined as

$V = \{q_1, q_2, \ldots q_n\}$

$T = \Sigma = \{a_1, a_2, \ldots a_n\}$

$S = \delta = q_1$

P = Transitions of FA.

Rules :-

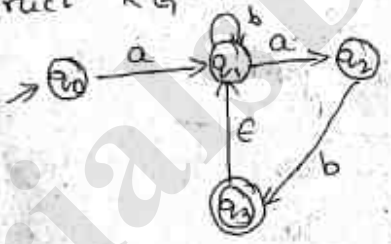
* If the transition of FA is like $q_i \xrightarrow{a} q_j$ then the production rule is $\boxed{q_i \to a q_j}$

$q_i \to a q_j$

* If there is a final state in FA is like (q_i) then the production rule is $\boxed{q_i \to \epsilon}$

* If the transition of FA is like $q_i \xrightarrow{a} (q_j)$ then the production rules are $\boxed{q_i \to a q_j \quad q_i \to a}$

Ex:- Construct RG from the given FA.



Sol:- The given FA is like $M = (Q, \Sigma, \delta, q_0, F)$

where $Q = \{q_0, q_1, q_2, q_3\}$

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

δ is a transition function is defined as.

δ: states	input symbols a	b	ε
→ q_0	q_1	ϕ	ϕ
q_1	q_2	q_1	ϕ
q_2	ϕ	q_3	ϕ
(q_3)	ϕ	ϕ	q_1

$q_0 = q_0$

$F = \{q_3\}$

Now, therefore the Equivalent regular Grammar of M is defined as $G = (V, T, P, S)$ where

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

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

P is a set of production rules defined from transitions of M is like.

P:

$\{ q_0 \to a q_1$

$q_1 \to a q_2$

$q_1 \to b q_1$

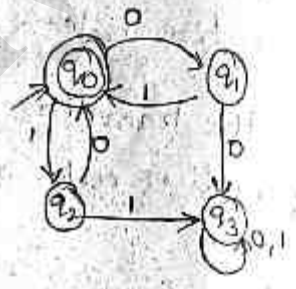
$q_2 \to b q_3$

$q_2 \to b$

$q_3 \to \epsilon q_1 \}$ //

$\therefore S = \{q_0\}$

2) Construct R.G from the given FA.



Sol:- The Given FA is like.

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

$Q = \{q_0, q_1, q_2, q_3\}$

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

δ:

states	input symbols 0	1
→ q_0	q_1	q_2
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_3	q_3

Now, therefore the Equivalent RG of M is defined as $G = (V, T, P, S)$ where

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

$T = \{0, 1\}$

P is a set of production rules defined from transitions of M is like

P: $\{ q_0 \to 0q_1$ $q_0 \to 1q_2$ $q_1 \to 0q_3$ $q_1 \to 1q_0$ $q_1 \to 1$ $q_2 \to 0q_2$ $q_2 \to 0$ $q_2 \to 1q_3$ $q_3 \to 0q_3$ $q_3 \to 1q_3 \}$

Finite automata from RG :-

Let $G = (V, T, P, S)$ be a RG. We can construct DFA 'M' whose

1. states corresponds to variables 'V'
2. starting state corresponds to starts symbol 'S'.
3. Transitions in 'M' corresponding to production Rules in 'P'.
4. Input symbols corresponding to terminals in 'T'.
5. If there is a production is of the form $q_i \to a$ then the transition is terminate at a new state called final state

Rules :-

* If the production rule is of the form $A \to \epsilon$ then 'A' is the final state. (A)
* A production rule is of the form $A_i \to a$ then there is a transition from A_i to final state labelled with a.

$A_i \to a$ = $(A_i) \xrightarrow{a} ((f))$

* A production rule is of the form $A_i \to aA_j$ then there is a transition from A_i to A_j labelled with 'a'.

$(A_i) \xrightarrow{a} (A_j)$

* If a production rule is of the form $A_i \to a_1 a_2 a_3 \cdots a_m A_j$ then there is a transition from A_i to A_j and Add intermediate states labelled by $a_1, a_2, a_3, \ldots, a_m$.

$A_i \xrightarrow{a_1} (x_1) \xrightarrow{a_2} (x_2) \xrightarrow{a_3} (x_3) \cdots (x_m) \xrightarrow{a_m} (A_j)$

Ex :- Construct a FA from the RG.

$S \to aA/B$
$A \to aaB$
$B \to bB/a$

Sol :- The Given $G = (V, T, P, S)$ where

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

$T = \{a, b\}$

$P = \{ S \to aA \quad B \to a \}$
$S \to B$
$A \to aaB$
$B \to bB$

$S = \{s\}$

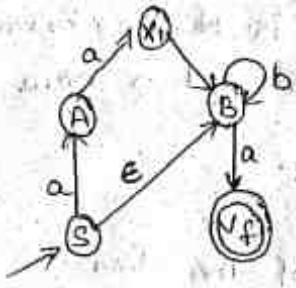
∴ The Equivalent DFA M is

$M = (Q, \xi, \delta, q_0, F)$ where

$Q = \{s, A, B, V_f\}$

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

δ :



$Q = \{s, A, B, V_f, X_1\}$

$\xi = \{a, b, \epsilon\}$

$q_0 = \{s\}$

$F = \{V_f\}$

2) Construct FA from the given RG.

$S \rightarrow aA/\epsilon$

$A \rightarrow aA/bB/\epsilon$

$B \rightarrow bB/\epsilon$

Sol:- The given $G = (V, T, P, S)$

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

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

$P = \{ S \rightarrow aA$

$S \rightarrow \epsilon$

$A \rightarrow aA$

$A \rightarrow bB$

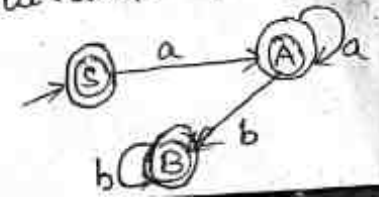
$A \rightarrow \epsilon$

$B \rightarrow bB$

$B \rightarrow \epsilon \}$

$S = \{S\}$

The Equivalent DFA M is $M = (Q, \xi, \delta, q_0, F)$



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

$\xi = \{a, b, \epsilon\}$

$q_0 = \{S\}$

Regular Expression from Regular grammar:—

Let $G = (V, T, P, S)$ be a RG. Now the RE 'R' is defined from 'G' by using the following rules.

* Replace the '$\rightarrow$' in the Grammar productions with equal symbol (=) to get the set of equations.
* convert the equation of the form $A \rightarrow aA|ab$

$$\Downarrow$$

$$A = a^*ab$$

* Repeat the step 2 until we get the R.E for the starting symbol. This gives the final R.E of given grammar G.

Ex:- obtain the Regular expression from the grammar given below.

$S \rightarrow 01B|0$

$B \rightarrow 1B|1$

Sol:- The given Regular Grammar $G = (V, T, P, S)$

where

$V = \{S, B\}$

$T = \{0, 1\}$

$P = \{ S \rightarrow 01B$
$S \rightarrow 0$
$B \rightarrow 1B$
$B \rightarrow 1 \}$

$S = \{S\}$

Replace arrow from set of productions with equal '='

$S = 01B|0$

$B = 1B|1$

$$\Downarrow$$

$B = 1^*1$

$B = 1^+$

Sub $B = 1^+$ in $S = 01B|0$

$S = 011^+|0$

The Final RE is

$S = 011^+ + 0$

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

$= 01^*$

2)

$S \rightarrow baS \mid aA$

$A \rightarrow bbA \mid bb$

Sol:- The given RG $G = (V, T, P, S)$

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

$T = \{a, b\}$

$P = \{ S \rightarrow baS$

$S \rightarrow aA$

$A \rightarrow bbA$

$A \rightarrow bb\}$

Replace arrow from set of productions with =

$S = baS \mid aA$

$A = bbA \mid bb$ → $A = bbA \mid bb$

$A = bb^* bb$

$S = baS \mid aA$

$S = ba^* aA$

Sub $A = bb^* bb$ in $S = ba^* aA$

$S = ba^* abb^* bb$

$R.E = ba^* abb^* bb$

$= ba^+ b^+ bb.$

Introduction:-

* Pumping Lemma is used for checking the given language is Regular (or) not.
* Let $M = (Q, \Sigma, \delta, q_0, F)$ be a FA with 'n' states.
* Let 'L' be a Regular language accepted by 'M'
* Let 'w' belongs to L ($w \in L$) and their Exists xyz such that $w = xyz$ and $xy^i z \in L$ for Each 'm' $i \geq 0$.

Application:-

step 1:- Assume 'L' be regular language and 'n' is the no. of states in the FA.

2:- choose the string 'w' such that $|w| \leq n$. use pumping lemma to write $w = xyz$. with the conditions.

i) $|xy| < n$

ii) $|y| > 0$

3:- find a suitable integer 'i' such that $xy^i z \notin L$. Hence 'L' is not regular.

Examples:-

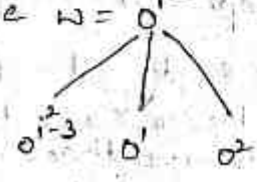
1) S.T the set $L = \{0^{i^2} / i \geq 1\}$ is not regular.

Sol:- Given $L = \{0^{i^2} / i \geq 1\}$

$$L = \{0^{1^2}, 0^{2^2}, 0^{3^2}, 0^{4^2}, \ldots\}$$

$$L = \{0^1, 0^4, 0^9, 0^{16}, \ldots\}$$

Assume $w = 0^{i^2}$



for $i = 2$

$$xy^2z = 0^{i^2-3}\, 0^2\, 0^2$$

$$= 0^{i^2+1}$$

$$= 0^{2^2+1}$$

$$= 0^5 \notin L$$

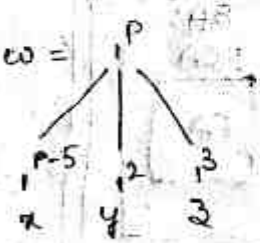
The given set is not regular.

2) S.T the set $L = \{1^p / p \text{ is prime}\}$ is not regular.

Sol:- Given $L = \{1^p / p \text{ is prime}\}$

$$L = \{1^2, 1^3, 1^5, 1^7, 1^{11}, 1^{13}, \ldots\}$$

Assume $w = 1^p$



for $i = 2$

$$xy^2z = 1^{p-5} (1^2)^2 (1^3)$$

$$= 1^{p-5}\, 1^4\, 1^3 \Rightarrow 1^{p+2} \notin L$$

$\therefore L$ is not regular.

//

UNIT-III

Formal Grammar:

* Introduction
* classification of formal Grammar
    1. chomsky Hirearchy.
    2. Types.

* **Introduction:-**

Mathematically A formal Grammar is a tuple like

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

V = finite and non empty set of non-Terminal symbols (or) Variables.

variables are represented by upper case letter

T = finite and non empty set of Terminal symbols Represented by lower case letters and some special symbols are there.

P = It is a set of production rules are of the form

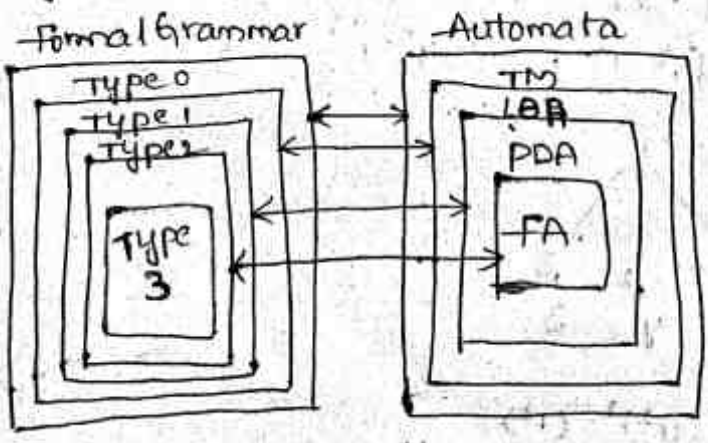
$$P \rightarrow \alpha \rightarrow \beta$$
$$\alpha \in V$$
$$\beta \in (V \cup T)^*$$

$S \rightarrow$ It is the starting symbol of the Grammar is always, a variable which is $S \in V$.

NOTE:- Grammar's are used to describe a language

* **classification of Grammar:-**

- using chomsky hierarchy.



Type 3 Grammar :—

* It is also called as Regular grammar.
* Type 3 Grammar is defined as $G = (V, T, P, S)$ where,

V → set of variables.
T → set of Terminals
P → set of production rules are of the form

$A \to Ba$
$A \to a$ — According to left linear grammar

(or)

$A \to aB$
$A \to a$ — According to Right linear grammar.

Ex :— $A \to aB$
$A \to Ba$
$A \to a$
$A \to \epsilon$

where,
$(A, B) \in V$
$a \in T^*$

* Type 3 Grammar is used to generating Regular language
* Regular languages are recognised (or) accepted by finite automata. i.e; NFA (or) DFA.

Type 2 Grammar :—

* It is also called as Context-free grammar.
* Ty context-free grammar is defined as $G = (V, T, P, S)$

where V → finite set of variables
T → finite set of Terminals
P → finite set of production rules are of the form

$\alpha \to \beta$

where $\alpha \in V$
$\beta \in (V \cup T)^*$

Ex :— $S \to aSa$
$S \to bSb$
$S \to ab$
$S \to \epsilon$

* Context-free grammars are used to generate Context-free language.

* context-free language recognised (or) Accepted by pushdown Automata.

Type 1 Grammar :—

* It is also called as context - Sensitive Grammar.
* A CSG is defined as $G=(V,T,P,S)$ where

V = finite set of variables

T = finite set of terminals

P = set of production rules are of the form $\alpha \rightarrow \beta$

where, $\alpha \in (VUT)^+$

$\beta \in (VUT)^*$

length of $|\alpha| \leq$ length of $|\beta|$

Ex:- $S \rightarrow aBb$

$bB \rightarrow aa$

$\beta \rightarrow b$

* CSG is used to generating Context - Sensitive language
* CSL recognised (or) Accepted by Linear Bounded Automata

Type 0 Grammar :—

* It is also called also Recursive - Grammar (or) Recursive Enumerable grammar (or) phrase structured Grammar.
* mathematically Recursive grammar is defined as

$G=(V,T,P,S)$ where $V \rightarrow$ finite set of variables

$T \rightarrow$ finite set of Terminals

$P \rightarrow$ set of production rules are of the form.

$\alpha \rightarrow \beta$

$\alpha \in (VUT)^{*+}$

$\beta \in (VUT)^*$

$|\alpha| \geq |\beta|$

Ex:- $S \rightarrow aAbB$

$aAbB \rightarrow aB$

$abB \rightarrow aA$

$A \rightarrow \epsilon$

* Recursive Grammars are used to generating recursive language (or) Recursive-Enumerable language (or) phrase structured language.

* Recursive languages are recognised are accepted by Turing machine.

Relationship b|w formal grammar and automata:—

1. Type 3 ⊆ Type 2 ⊆ Type 1 ⊆ Type 0
2. FA ⊆ PDA ⊆ LBA ⊆ TM

Context-Free Grammar:

* Introduction
* Design of CFL
* closure properties of CFL

*Introduction:—

Context-free Grammar is a Grammar which is defined by four tuples like $G=(V, T, P, S)$ where,

V – It is finite and non-empty set of non-terminal symbols (or) variables.

T – finite and non-empty set of Terminal symbols.

P – finite and non-empty set of production rules are of the form $\alpha \to \beta$

$\alpha \in V$

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

Ex: $S \to asa$

$S \to bSb$

$S \to aa|bb$

$S \to \epsilon$

S → It is starting symbol.

Context free language:—

Let $G=(V, T, P, S)$ be a Context-free grammar. The CFG generating a language 'L' is called Context-free language

*It is denoted by L(G).

*Context-free languages are organized by PDA.

Design of CFL :—

1) Construct a CFL for the following set. $\{\epsilon, a, aa, aaa, \cdots a^n\}$

Sol :- Given set $\{\epsilon, a, aa, aaa, aaaa, \cdots\cdots a^n\}$

minimum string = ϵ

Next minimum string = a

Maximum string = a^n

$S \rightarrow a^n$

$\Downarrow$

$a \cdot a^{n-1} \Rightarrow S \rightarrow aS$

$\Downarrow$

$a \cdot a \cdot a^{n-2}$ $\quad S \rightarrow \epsilon$

$\Downarrow$ $\quad S \rightarrow a$

$a \cdot a \cdot a \cdot a^{n-3}$

CFG :- S

$S \rightarrow aS$

$S \rightarrow \epsilon$

$S \rightarrow a$

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

2) Construct a CFL for the following set $\{\epsilon, ab, aabb, \cdots\}$

Sol :- Minimum String = ϵ

Next minimum string = ab

Maximum string = $a^n b^n$

$S \rightarrow a^n b^n$

$S \rightarrow a a^{n-1} \cdot b^{n-1} b$

$S \rightarrow aa a^{n-2} b^{n-2} bb$

$\therefore S \rightarrow aSb$

$S \rightarrow \epsilon$

$S \rightarrow ab$

$\therefore$ CFG :- $S \rightarrow aSb$

$S \rightarrow \epsilon$

$S \rightarrow ab$

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

3) construct a CFL for the following set {a, b, ab, aabb, aaabbb, ...}

sol:- Minimum string = a / b

Maximum string = $a^n b^n$

$S \rightarrow a^n b^n$

$\rightarrow a a^{n-1} b^{n-1} b$ $\Rightarrow$ $S \rightarrow aSb$

$\rightarrow aaa^{n-2} b^{n-2} bb$ $S \rightarrow a$

$S \rightarrow b$.

$\therefore$ CFG $S \rightarrow asb$

$S \rightarrow a$

$S \rightarrow b$

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

4) Construct a CFG to generate the language $L = \{a^n b^{2n} / n \geq 1\}$

sol:- Minimum string = abb

Maximum string = $a^n b^{2n}$

$S \rightarrow a^n b^{2n}$

$S \rightarrow a a^{n-1} b^{2n-2} bb$ $\Rightarrow$ $S \rightarrow aSbb$

$S \rightarrow abb$.

$\therefore$ CFG = $S \rightarrow aSbb$

$S \rightarrow abb$.

5) Construct CFG for the following CFL

$L = \{ 0^i 1^{i+1} \mid i \geq 0 \}$

sol: $L = \{ 0^i 1^{i+1} \mid i \geq 0 \}$

$= 0^i 1^i . 1$

$S \rightarrow A1$

$A \rightarrow 0^i 1^i$

$\rightarrow 0 0^{i-1} 1^{i-1} 1$

$\rightarrow 0A1$

$A \rightarrow 0A1$

$A \rightarrow \epsilon$

$A \rightarrow 01$

CFG: $S \rightarrow A1$

$A \rightarrow 0A1$

$A \rightarrow \epsilon$

$A \rightarrow 01$

6) Construct a CFL from the following Language.

$L = \{ a^m b^m c^n \mid m, n \geq 0 \}$

sol:- $\underbrace{a^m b^m}_{A} \underbrace{c^n}_{B}$

$A \to a^m b^m$

$\to a\, a^{m-1} b^{m-1} b$

$A \to aAb$

$A \to \epsilon$

$A \to ab$

$B \to c^n$

$B \to c c^{n-1}$

$B \to cB$

$B \to \epsilon$

$B \to c$

CFG :- $S \to AB$

$A \to aAb$

$A \to \epsilon$

$A \to ab$

$B \to cB$

$B \to \epsilon$

$B \to c$

Closure properties of CFL :-

- context free languages are closed under union
- " " " " " " concatenation
- " " " " " " kleene closure
- " " " " " " Reversal
- context free languages are not closed under Complement
- " " " " " " Intersection
- " " " " " " difference.

** Derivation :-

* Introduction * Types of Derivation * Derivation tree

Derivation is a process of generating a string from a given grammar.

Derivation process can be represented graphically is called Derivation tree (or)

* left most derivation * Rightmost derivation.

Left most derivation :- with example

In this, we can replace a left most Variable to obtain the given input string.

Right most derivation :-

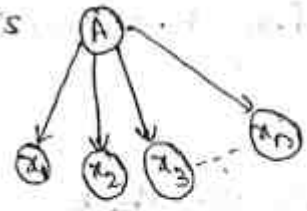
In this, we can replace a Right most variable to obtain the given input string.

Derivation Tree :—

Let $G = (V, T, P, S)$ be a CFG. Then there is a derivation tree for G. If and only if.

* the root node of the Tree is labelled with Start Symbol of G
* All leaf nodes of Tree are labelled by Terminals (or) special symbols of G.
* the interior nodes are labelled by variables of G.
* If any production rule in G. is are the form.

$A \rightarrow x_1 x_2 x_3 \cdots x_n$ then the derivation tree is



find the i) left most derivation
ii) Right most derivation
iii) parse tree for the i/p string id+id*id
from the following grammar. $E \rightarrow E+E$
$E \rightarrow E*E$
$E \rightarrow id$

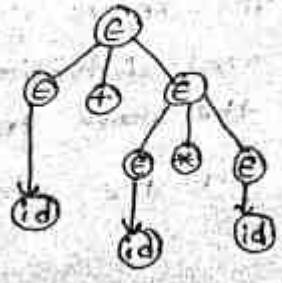
Sol:- the given grammar is $E \rightarrow E+E$
$E \rightarrow E*E$
$E \rightarrow id$

Input string id+id*id.

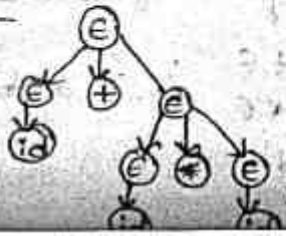
LMD:- $E \rightarrow \underline{E}+E$
$\rightarrow id+\underline{E}$
$\rightarrow id+\underline{E}*E$
$\rightarrow id+id*\underline{E}$
$\rightarrow id+id*id$.

RMD:- $E \rightarrow E+\underline{E}$
$\rightarrow E+E*\underline{E}$
$\rightarrow E+\underline{E}*id$
$\rightarrow \underline{E}+id*id$
$\rightarrow id+id*id$

Parse tree:—



Parse tree:—



Ambiguous Grammar:-

* A CFG $G=(V,T,P,S)$ which generates two (or) more parse trees for given i/p string is called Ambiguous grammar.

* that means an Ambegeous grammar has two or more left most derivations (or) right most derivation (or) parse tree.

Ex:- Prove that $S \to aSbS$, $S \to bSaS$, $S \to \epsilon$ is ambegeous for the i/p string abab

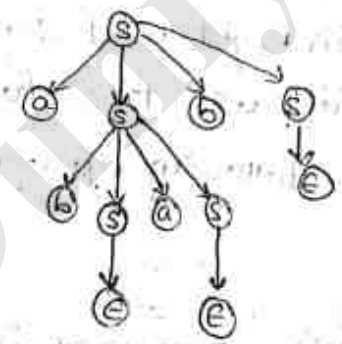
sol:- The given content free Grammar is $S \to aSbS$, $S \to bSaS$, $S \to \epsilon$

the input string is $\omega = abab$

① LMD:

$S \to a\underline{S}bS$
$\to ab\underline{S}aSbS$
$\to ab\epsilon aSbS$
$\to aba\underline{S}bS$
$\to aba\epsilon bS$
$\to abab\underline{S}$
$\to abab\epsilon$
$\to abab$

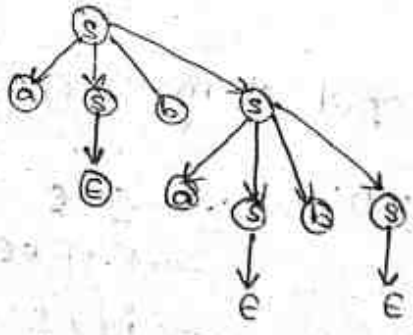
parse tree



② LMD:

$S \to a\underline{S}bS$
$\to a\epsilon bS$
$\to ab\underline{S}$
$\to aba\underline{S}bS$
$\to aba\epsilon bS$
$\to abab\underline{S}$
$\to abab\epsilon$
$\to abab$

parse tree



∴ the above grammar generates two parse trees (or) two left most derivation for the same i/p string $\omega = abab$. Hence the above grammar is ambegeous grammar.

2) P.T the grammar $E \to E+E$, $E \to E*E$, $E \to id$ is ambegeous for i/p string id+id*id.

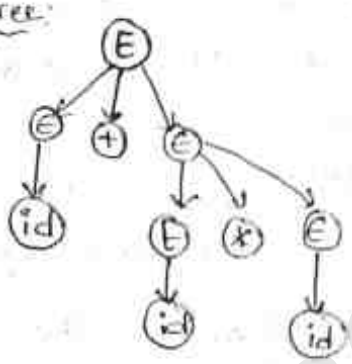
Sol:- the given context free grammar is $E \to E+E$
$E \to E*E$
$E \to id$

the input string is w= id+id*id.

① LMD:

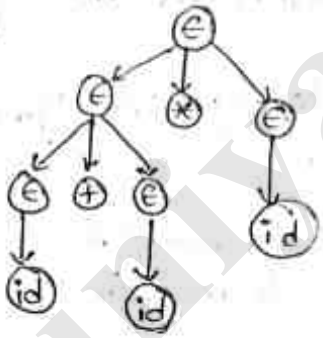
$E \to \underline{E}+E$
$\to id+\underline{E}$
$\to id+\underline{E}*E$
$\to id+id*\underline{E}$
$\to id+id*id.$

Parse tree:



② LMD:-

$E \to \underline{E}*E$
$\to \underline{E}+E*E$
$\to id+\underline{E}*E$
$\to id+id*\underline{E}$
$\to id+id*id.$



** (Jan) Simplification of CFG:

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

Introduction:-

It's means minimizing the no. of productions in the given CFG. that is reducing size of CFG. size of CFG is equal to no. of productions.

Methods:- $S \to AB$
$A \to a$
$A \to aA$
$B \to SB$

Elimination of useless symbols:-

useful symbol:- A variable is said to be useful if and only if

* It generates a terminal string
* It is used in derivation of a string at least one time

useless symbol :-

* A variable is said to be useless if and only if.
  * It doesn't generates a terminal string.
* It doesn't used in derivation of a string at least one time.

Procedure :-

step 1 :- Determine useless symbols in the grammar.

step 2 :- Remove the productions which contains useless symbols in the grammar.

Ex :- eliminate useless symbols from the following grammar.

$S \rightarrow AB \mid CA$
$B \rightarrow BC \mid AB$
$A \rightarrow a$
$C \rightarrow aB \mid b$

sol :- The given CFG is

$S \rightarrow AB$
$S \rightarrow CA$
$B \rightarrow BC$
$B \rightarrow AB$
$A \rightarrow a$
$C \rightarrow aB$
$C \rightarrow b$

In the given grammar 'B' doesn't generating a terminal string.

$S \rightarrow \underline{A}B$
$S \rightarrow a\underline{B}$
$\rightarrow a\underline{B}C$
$\rightarrow a\underline{A}BC$
$\rightarrow aaB\underline{C}$
$\rightarrow aaBb$
$\rightarrow aa\underline{A}Bb$
$\rightarrow aaaBb$
:

∴ so, 'B' is useless symbol.

so, we can eliminate the productions which contains 'B'.

∴ The reduced CFG is

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

2) elimination of ∈-production :-

∈-production :- A production is of the form $A \to \epsilon$ is called ∈-production (or) NULL production.

procedure :-

step 1 :- If the grammar contains $A \to \epsilon$ then replace 'A' with ∈ in the remaining productions.

step 2 :- Remove $A \to \epsilon$ from the grammar.

ex :- Remove ∈-productions from the following grammar

$A \to 0B1 \mid 1B1$

$B \to 0B \mid 1B \mid \epsilon$

sol :- the given CFG is $A \to 0B1$

$A \to 1B1$

$B \to 0B$

$B \to 1B$

$B \to \epsilon$.

$A \to 0B1$ ∴ $A \to 0B1$
$\to 0\epsilon 1$ $A \to 01$
$\to 01$

$B \to 1B1$ ∴ $A \to 1B1$
$\to 1\epsilon 1$ $A \to 11$
$\to 11$

$B \to 0B$ ∴ $B \to 0B$
$\to 0\epsilon$ $B \to 0$
$\to 0$

$B \to 1B$ ∴ $B \to 1B$
$\to 1\epsilon$ $B \to 1$
$\to 1$

After eliminating $B \to \epsilon$ the resultant CFG is

$A \to 0B1$
$A \to 01$
$A \to 1B1$
$A \to 11$
$B \to 0B$
$B \to 0$
$B \to 1B$
$B \to 1$

14m
Normal forms :-

*Introduction

*Types of Normal forms

1. chomsky Normal form (CNF)
2. Greibach Normal form (GNF)

Introduction :-

In CFG each production of the form $\alpha \rightarrow \beta$ where $\alpha \in V$ that means β contains any no. of non-terminal symbols and any no. of terminal symbols. But, we need to have a grammar in specific form i.e; we can decide the no. of non-terminals and terminals on RHS of the grammar. This can be implemented by using "normalization of CFG".

Normalization :-

The process of Arranging the grammar with fixed no. of

non-terminals and terminals on R.H.S of CFG is called normalization.

normal forms are classified into two types

i) chomsky normal form.

ii) Greibach normal form

chomsky normal form :—

It is defined as $\alpha \rightarrow \beta$

non-terminal → Non-terminal. Non-terminal. (or) Non-terminal → Terminal.

conversion of CFG to CNF :—

procedure :—

step 1 :- simplify the CFG

step 2 :- convert the simplified CFG to CNF.

Ex:- convert the following CFG into chomsky normal form.

$S \rightarrow aaaaS$

$S \rightarrow aaaa$

Sol:- The given grammar is $S \rightarrow aaaaS$

$S \rightarrow aaaa$

consider a non-terminal $A \Rightarrow$ that derives terminal a

∴ The production rule is $A \rightarrow a$ is in CNF

$S \rightarrow aaaaS$

$S \rightarrow AAAAS$ can be replaced by P_1

$S \rightarrow AP_1$ is in CNF.

$P_1 \rightarrow AAAS$ can be replaced by P_2

$P_1 \rightarrow AP_2$ is in CNF

$P_2 \rightarrow AAS$ can be replaced by P_3

$P_2 \rightarrow AP_3$ is in CNF

$P_3 \rightarrow AS$ is in CNF

$S \rightarrow aaaa$

$S \rightarrow AAAA$ can be replaced by P_4

$S \to AP_4$ is in CNF

$P_4 \to A[AA]$ can be replaced by P_5

$P_4 \to AP_5$ is in CNF

$P_5 \to AA$ is in CNF

The resultant Grammar CNF is

$S \to AP_1$ $\quad P_5 \to AA$

$S \to AP_4$ $\quad A \to a$

$P_1 \to AP_2$

$P_2 \to AP_3$

$P_3 \to AS$

$P_4 \to AP_5$

2) Convert the given CNFG to CNF $\quad S \to aSa$

$S \to bSb$

$S \to a$

$S \to b$.

Sol:- The given grammar is $S \to aSa$

$S \to bSb$.

$S \to a$

$S \to b$

It is already in simplified form.

Consider a non-terminal A that derives a terminal a and the non-Terminal B. that derives the terminal b.

∴ the production rules is $A \to a$ $\quad B \to b$. are in CNF.

(i) $S \to aSa$.

$S \to A[SA]$ can be replaced by P_1

$S \to AP_1$ is in CNF.

$P_1 \to SA$ is in CNF.

(ii) $S \to bSb$

$S \to B[SB]$ can be Replaced by P_2

$S \to BP_2$ is in CNF

$P_2 \to SB$ is in CNF

(iii) $S \to a$ is in CNF

$S \to b$ is in CNF.

∴ the resultant grammar in CNF is

$S \to AP_1$

$S \to BP_2$

$S \to a$

$S \to b$

$P_1 \to SA$

$P_2 \to SB$

$A \to a$

$B \to b$.

Greibach Normal Form (GNF) :-

GNF is defined as

Non-terminal → Terminal. any no. of nonterminals

Non-terminal → Terminal.

Lemma 1:

Let CFG be $G = (V, T, P, S)$ and there is a production rule $A \to \alpha B$ and $B \to \beta_1 | \beta_2 | \beta_3 | \ldots | \beta_n$ then add the new production rule to GNF $A \to \alpha\beta_1 | \alpha\beta_2 | \alpha\beta_3 | \ldots | \alpha\beta_n$ to GNF.

∴ B is replaced by $B \to \beta_1 | \beta_2 | \ldots | \beta_n$

Lemma 2:

Let CFG be $G = (V, T, P, S)$ and there is production rule

$A \to A\alpha_1 | A\alpha_2 | \ldots | A\alpha_n | \beta_1 | \beta_2 | \ldots | \beta_n$ then the production rules are added to GNF.

$A \to \beta_1 | \beta_2 | \beta_3 | \ldots | \beta_n$

$A \to \beta_1 Z | \beta_2 Z | \beta_3 Z | \ldots | \beta_n Z$

$Z \to \alpha_1 | \alpha_2 | \alpha_3 | \ldots | \alpha_n$

$Z \to \alpha_1 Z | \alpha_2 Z | \alpha_3 Z | \ldots | \alpha_n Z$

Converting ~~CFG~~ CFG into GNF :-

Procedure:-

step 1:- Simplify the CFG.

step 2:- Converting simplified CFG into GNF.

Ex:- Convert the given CFG to GNF. $S \to ABA$

$A \to aA | \epsilon$

$B \to bB | \epsilon$

Sol:- The Given CFG is $S \to ABA$

$A \to aA$

$A \to \epsilon$

$B \to bB$

$B \to \epsilon$

Simplified of given CFG:-

(a) elimination of ϵ-productions:-

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

(1) $S \to ABA$
$S \to \epsilon BA$
$S \to BA$

(2) $S \to ABA$
$S \to AB\epsilon$
$S \to AB$

(3) $S \to ABA$
$S \to A\epsilon A$
$S \to AA$

(4) $S \to ABA$
$S \to \epsilon\epsilon A$
$S \to A$

(5) $S \to ABA$
$S \to \epsilon B\epsilon$
$S \to B$

$A \to aA$ $\quad$ $B \to bB$
$A \to a\epsilon$ $\quad$ $B \to b\epsilon$
$A \to a$ $\quad$ $B \to b$

∴ After eliminating $A \to \epsilon$, $B \to \epsilon$ from the grammar the resultant grammar is

$S \to ABA$ $\quad$ $A \to aA$
$S \to BA$ $\quad$ $A \to a$
$S \to AB$ $\quad$ $B \to bB$
$S \to AA$ $\quad$ $B \to b$
$S \to A$
$S \to B$

Elimination of unit productions:-

The above grammar has two unit productions like

$S \to A$ $\quad$ $S \to B$

$S \to aA$ $\quad$ $S \to bB$ $\quad$ [∵ since $A \to aA$, $A \to a$, $B \to bB$, $B \to b$]

$S \to a$ $\quad$ $S \to b$

∴ After elimination unit productions $S \to A$, $S \to B$ from the grammar. The resultant grammar is

$S \to ABA$ $\quad$ $A \to aA$
$S \to BA$ $\quad$ $A \to a$
$S \to AB$ $\quad$ $B \to bB$
$S \to AA$ $\quad$ $B \to b$
$S \to aA$
$S \to a$
$S \to bB$
$S \to b$

there is no useless production.

The simplified CFG is

$S \to ABA$ | $A \to aA$
$S \to BA$ | $A \to a$
$S \to AB$ | $B \to bB$
$S \to AA$ | $B \to b$
$S \to aA$
$S \to a$
$S \to bB$
$S \to b$

Converting simplified CFG to GNF:

i) $S \to ABA$
$S \to aABA$ ✓
$S \to aBA$ ✓

ii) $S \to BA$
$S \to bBA$ ✓
$S \to bA$ ✓

iii) $S \to AB$
$S \to aAB$
$S \to aB$ ✓

iv) $S \to AA$
$S \to aAA$
$S \to aA$

v) $S \to aA$ ✓
$S \to a$ ✓

vi) $S \to bB$ ✓
$S \to b$ ✓

$A \to aA$ ✓
$A \to a$ ✓
$B \to bB$ ✓
$B \to b$ ✓

∴ The resultant grammar is in GNF. is

$S \to aABA \mid aBA \mid bBA \mid bA \mid aAB \mid aB \mid aAA \mid aA \mid bB \mid a \mid b$
$A \to aA \mid a$
$B \to bB \mid b$.

(2) Convert the following CFG into GNF $S \to AA \mid 0$
$A \to SS \mid 1$

Sol: Given Grammar $S \to AA$
$S \to 0$
$A \to SS$
$A \to 1$

The simplified CFG is. $S \to AA$
$S \to 0$
$A \to SS$
$A \to 1$

① $S \to AA|0$
$S \to SA|0$
$S \to 0$
$S \to 0Z$
$Z \to SA$
$Z \to SAZ$
$Z \to SA$
$Z \to 0A$
$Z \to 0ZA$
$Z \to 1AA$
$Z \to 0A$

② $S \to AA|0$
$S \to 1A|0$
$S \to 1A$
$S \to 0$

$Z \to SAZ$
$Z \to 0AZ$
$Z \to 0ZAZ$
$Z \to 1AAZ$
$Z \to 0AZ$

③ $A \to 0S$
$A \to 0S$
$A \to 0ZS$
$A \to 1AS$
$A \to 0S$

∴ The resultant grammar is

$S \to 0|0Z|1A$

$Z \to 0A|0ZA|1AA|0A|0AZ|0ZAZ|1AAZ|$

$A \to 0S|0ZS|1AS|1$

③ convert the given CFG to GNF $S \to CA$
$A \to a$
$C \to aB|b$

⇒ Given CFG is not a simplified grammar

After eliminating the useless symbols the resultant CFG is. $S \to CA$
$A \to a$
$C \to b$

By applying Lemma 1 $S \to CA$
$S \to bA$

∴ The resultant GNF is $S \to bA$
$A \to a$
$C \to b$

④ Convert the given CFG to GNF $S \to SS$
$S \to 0S1|01$

The given CFG is a simplified CFG

The resultant grammar is $S \to SS$
$S \to 0S1$
$S \to 01$

Replaced 0 by A, 1 by B

then productions are $A \to 0$
$B \to 1$

S → CS

S → ASB

S → AB

Applying Lemma ①

① S → SS
S → ASBS
S → OSBS

② S → SS
S → ABS
S → OBS

② S → ASB
S → OSB
S → AB
S → OB

The resultant grammar GNF is

S → OSBS | OBS | OSB | OB

A → O

B → 1

Pumping Lemma for CFL:—

pumping Lemma is used for proving the given language is CFL (or) not

Lemma :- let 'L' be any CFL, then there is a constant 'n' which depends only a pair 'L' such that there exist a string.

$w = uvxyz$ such that 1. $|vy| \geq 1$
2. $|vxy| \leq n$
3. for $\forall\, i \geq 0$ uv^ixy^iz is in L

Then L is said to be CFL. otherwise it is not a CFL.

① Prove that $L = \{a^n b^n c^n \mid n \geq 0\}$ is not a CFL

The given language $L = \{a^n b^n c^n \mid n \geq 0\}$.

$L = \{\epsilon, abc, aabbcc, ----\}$

consider a constant n and the string $w = a^n b^n c^n$

consider a string $w \in L$

$w = abc$ for $n = 1$

$|w| = 3n$

for $i = 1$ $w = abc$

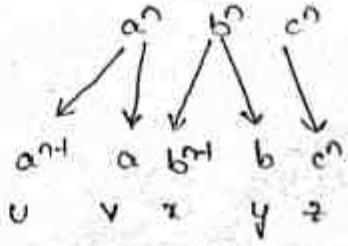
for $i = 2$

$w = uv^1xy^1z$

$w = uv^2xy^2z$

$w = a^{n-1} a^2 b^{n-1} b^2 c^n$

$w = a^{n+1} b^{n+1} c^n \notin L$



∴ The given language is not a CFL.

② show that the language $L = \{ ss^T / s \in \{a,b\}^* \}$

Given language $L = \{ ss^T / s \in \{a,b\}^* \}$

$L = \{ \epsilon,$

UNIT-IV

PUSH DOWN AUTOMATA

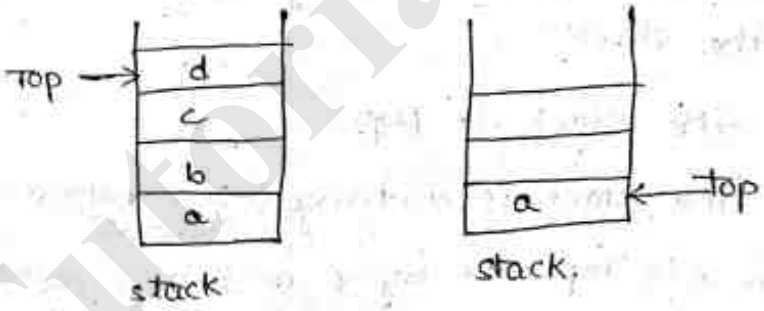
* Introduction * Basic model (Formal definition) * Graphical Notation
* Instantaneous Description (ID) * Acceptance of PDA.

Introduction:-

* A PDA is a way to implement a CFG in a similar way we can design FA for Regular Grammar.
* PDA is more powerful than finite state machine.
* FSM has a very limited memory. But a PDA has more memory.
* **PDA = FSM + stack**
* A stack is a way we arrange elements one on the top of stack.
* A stack does two basic operations.
  - i) PUSH :- A new element is added at the top of the stack.
  - ii) POP :- the top element of the stack is read and remove.

Ex:-

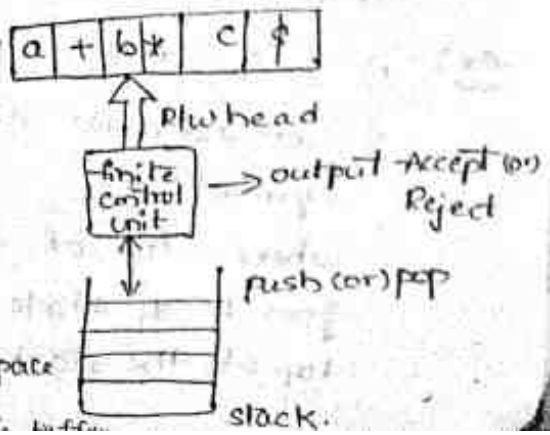
```
push(a)        pop()
push(b)        pop()
push(c)        pop()
push(d)
```



* **Basic model of PDA:-**

PDA has three Components.

i) input tape
ii) finite control unit.
iii) stack



* A stack with infinite size.
* It has unlimited amount of storage space
* Used to store data and remove the data temporarily which is read by the from

Formal definition:-

- Mathematically a PDA is defined with 7 tuples like

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where

Q → finite and non-empty set of states

Σ → finite and non-empty set of input symbols

Γ → finite and non-empty set of stack symbols.

δ → It is a transition function which is defined as

δ:

$$Q \times \{\Sigma \cup \epsilon\} \times \Gamma^* \longrightarrow Q \times \Gamma^*$$

$$Q \times \Sigma^* \times \Gamma^* \longrightarrow Q \times \Gamma^*$$

where 'δ' takes three tuples as i/p like $\delta(q, a, x)$

where i) q is a state in Q.

ii) a is either an i/p symbol in Σ (or) a is also belongs ϵ.

iii) x is a stack symbol i.e; member of Γ

iv) The o/p of δ is finite set of pairs like (p, γ)

where, p: It is a new state.

γ: It is a set of stack symbols that replace 'x' at the top of the stack.

Ex:- 1) If $\gamma = \epsilon$ then the stack is pop.

2) If $\gamma = x$ then the stack is unchange (since bypass operation)

3) If $\gamma = yz$ then x is replaced by z and 'y' is pushed on to the stack.

Ex:- 1) $\delta(q_0, a, z) = (q_1, yz)$

⇒ It indicates that from state q_0, reading i/p symbol 'a' where, top of the stack z then the finite control goes to q_1 state and adding the element 'y' to the top of the stack.

e) $\delta(q_1, a, z) = (q_2, \epsilon)$

⇒ It indicates that 'z' is removed from the stack and state is changed from q_1 to q_2

3) $\delta[q_1, a, z) = (q_2, z)$

⇒ It indicates that on reading symbol 'a' state is changing from q_1 to q_2 and there is no change in the stack (bypass operation)

q_0 → It is the initial state

$q_0 \in Q$

z_0 → It is the start stack symbol.

$z_0 \in T$

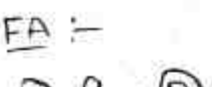
F → It is the set of final (or) accepting state and

$(F \subset Q)$.

Graphical representation :-

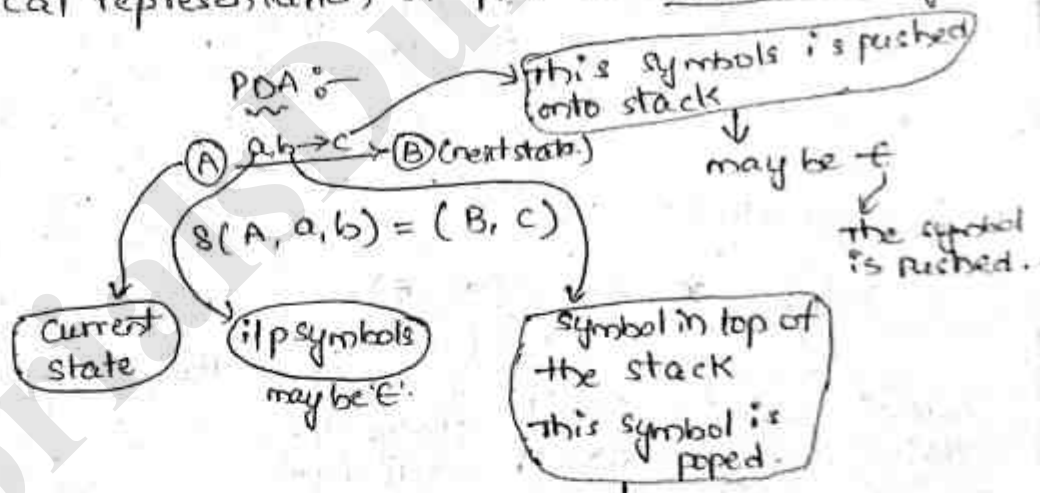
The Graphical representation of PDA is Transition diagram

FA :-



$\delta(A, a) = B$

PDA :-



Instantaneous description :-

It is used to describe the configuration of PDA at given Instance.

ID remembers the state and stack content.

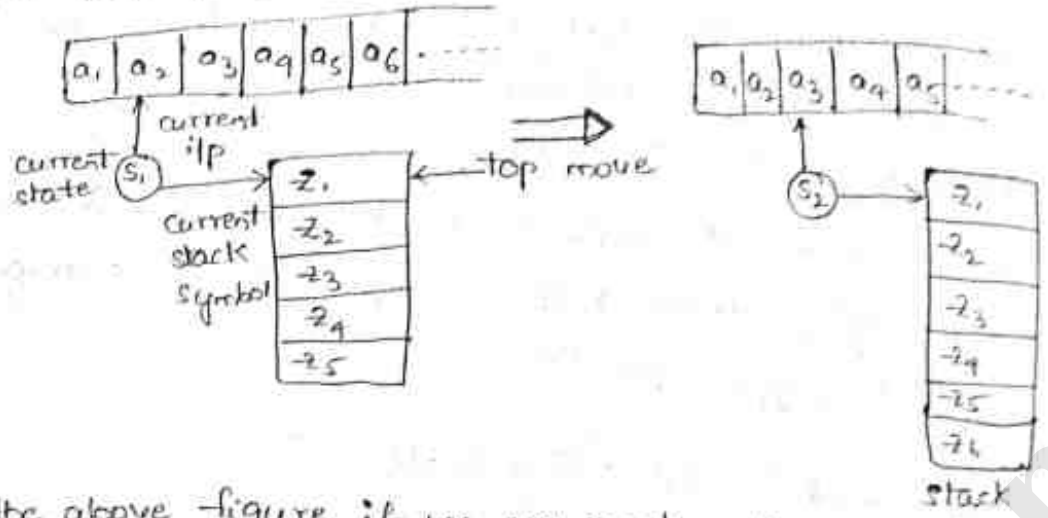
It was defined by Triple (q, w, γ) where

q → is a state.

w → input symbols of string

γ → is a string of stack symbols.

Example :- $\delta(P, a, z_0) = (q, \beta)$



From the above figure, if we are reading the current i/p symbol 'a_2' at current state 's_1' and current stack symbol 'z_1' then after a move we will reach to state s_2 and there will be some new symbol on the top of the stack. This description can be represented as.

1) push operation :—

$\delta(q_0, a_1, z_0) = (q_1, az_0)$ → push 'a' onto the stack.

- q_0 → current state (or) present state
- a_1 → current i/p symbol
- z_0 → current stack top symbol.
- q_1 → change the state from q_0 to q_1

2) pop operation :—

$\delta(q_0, a, y) = (q_1, \epsilon)$

- q_0 → current state
- a → current i/p symbol
- y → current stack top symbol.
- q_1 → change the state from q_0 to q_1
- ϵ → pop the stack (or) removing the stack.

Acceptance of PDA :—

There are two ways to accept a language by PDA they are i) Accepted by empty stack.
ii) Accepted by final state.

Accepted by empty stack :—

The given language Accepted by empty stack to be defined as $L(M) = \{\omega \mid \delta(q_0, \omega, z_0) \overset{*}{\Rightarrow} (P, \epsilon, \epsilon) \text{ for some } p \text{ in } Q\}$

that is, if stack becomes empty after scanning entire string then it is accepted by PDA - otherwise, not accepted.

Accepted by final string:-

the given language accepted by final state to be defined as

$L(M) = \{ w \mid \delta(q_0, w, z_0) \xrightarrow{*} (P, \epsilon, \gamma)$ for some $P \in F$ and $\gamma \in \Gamma^* \}$

that is, eventhough stack is not empty, after scanning i/p string. if the finite control reaches to the final state then it is accepted. otherwise, not accepted.

Design of PDA :-

Types of PDA :-

i) Deterministic PDA :- if all derivations in the design has to give only single move

ii) Non Deterministic PDA :- if derivation generates more than one move in the designing of a particular task.

1) Design a PDA that accepts equal no. of A's and B's

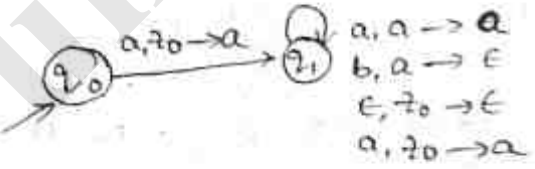
Sol:- δ: $\delta(q_0, a, z_0) = (q_1, a z_0)$

$\delta(q_1, a, a) = (q_1, aa)$

$\delta(q_1, b, a) = (q_1, \epsilon)$

$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$

$\delta(q_1, a, z_0) = (q_1, a z_0)$



∴ The PDA machine for the above language is defined as

$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ where $Q = \{q_0, q_1\}$

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

$\Gamma = \{z_0\}$

δ:

$q_0 = \{q_0\}$

$z_0 = \{z_0\}$

$F = \{\}$

Read A's → push operation, Read B's → pop operation

(ii) Consider a string $w = \{abab\}$ Read a's

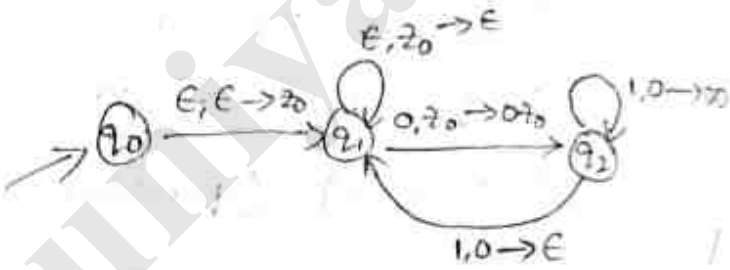
$\delta(q_0, abab, z_0) = \delta(q_1, bab, a z_0)$

③ Design a PDA for the language $L = \{0^n 1^{2n} / n \geq 1\}$

sol:- $L = \{0^n 1^{2n} \mid n \geq 1\}$

Read one 0 → push

Read two 1's → pop



$\delta(q_0, \epsilon, \epsilon) = (q_1, z_0)$

$\delta(q_1, 0, z_0) = (q_2, 0z_0)$

$\delta(q_2, 0, 0) = (q_2, 00)$

$\delta(q_2, 1, 0) = (q_2, 00)$

$\delta(q_2, 1, 0) = (q_1, \epsilon)$

$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon, \epsilon)$

$\delta(q_1, 1, 0) = (q_1, \epsilon)$

$\delta(q_1, 1, 0) = (q_1, \epsilon)$

④ Consider the string $w = \{001111\}$

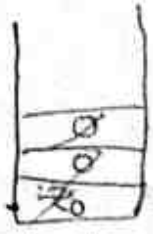
$\delta(q_1, 001111, z_0) = \delta(q_2, 01111, 0z_0)$

$= \delta(q_2, 1111, 00)$

$= \delta(q_2, 111, 00)$

$= \delta(q_2, 11, 0z_0)$

$= \delta(q_1, 1, 0z_0)$



stack

$= \delta(q_1, \epsilon, z_0)$

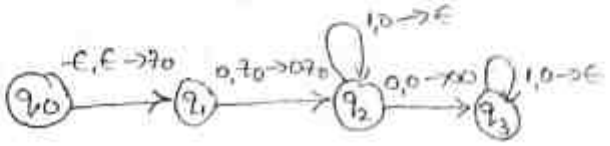
$= \delta(q_1, \epsilon, \epsilon)$

//

Design a PDA for the language $L = \{0^n 1^n \mid n \geq 1\}$

Read 0's → push

Read 1's → pop



$\delta(q_0, \epsilon, \epsilon) = (q_1, z_0)$

$\delta(q_1, 0, z_0) = (q_2, 0z_0)$

$\delta(q_2, 0, 0) = (q_2, 00)$

$\delta(q_2, 1, 0) = (q_3, \epsilon)$

$\delta(q_3, 1, 0) = (q_3, \epsilon)$

$\delta(q_3, \epsilon, z_0) = (q_3, \epsilon, \epsilon)$

* Design a PDA for the language $L = \{ww^R \mid w \in (a+b)^*\}$

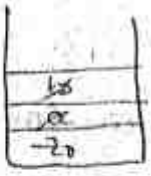
ii) $L = \{w c w^R \mid w \in (a+b)^*\}$

sol) i) $L = \{ww^R \mid w \in (a+b)^*\}$

In this language contains palindrome string. i.e;

if $w = ab$, $w^R = ba$ then $ww^R = abba$ is a palindrome

* we can read no. of a's and b's and pushed them into stack until we can reach the mid position of i/p string.
* In the mid position we can't read any i/p and can't push onto stack.
* After mid position when we read a (or) b then pop them from the stack. This process is repeated until stack, is empty.



$\delta(q_0, \epsilon, \epsilon) = (q_1, z_0)$

$\delta(q_1, a, z_0) = (q_1, az_0)$

$\delta(q_1, b, z_0) = (q_1, bz_0)$

$\delta(q_1, \epsilon, \epsilon) = (q_2, z_0)$

$\delta(q_2, a, a) = (q_3, \epsilon)$

$\delta(q_2, b, b) = (q_3, \epsilon)$

$\delta(q_3, \epsilon, z_0) = (q_4, \epsilon, \epsilon)$

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

$w = ab$

$w^R = ba$

$wcw^R = abcba$

$\delta(q_0, \epsilon, \epsilon) = (q_1, Z_0)$

$\delta(q_1, a, z) = (q_1, aZ_0)$

$\delta(q_1, b, Z_0) = (q_1, bZ_0)$

$\delta(q_1, a, a) = (q_1, aa)$

$\delta(q_1, a, b) = (q_1, ab)$

$\delta(q_1, b, a) = (q_1, ba)$

$\delta(q_1, b, b) = (q_1, bb)$

$\delta(q_1, c, Z_0) = (q_2, Z_0)$

$\delta(q_1, c, a) = (q_2, a)$

$\delta(q_1, c, b) = (q_2, b)$

$\delta(q_2, a, a) = (q_3, \epsilon)$

$\delta(q_3, b, b) = (q_3, \epsilon)$

$\delta(q_3, \epsilon, Z_0) = (q_4, \epsilon, \epsilon)$

Deterministic pushDown Automata:-

→ A DPDA is 7tuple like $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

where Q is finite and non empty set of states

Σ is finite and nonempty set of ilp Alphabet

Γ is finite set of stack symbols

δ is a mapping function used for mapping (or) moving from current state to next state. is defined as $\delta(q_0, x, Z_0) = (q, x\beta)$ where

q_0 is current state

x is current ilp symbol

Z_0 is current stack symbol

q is next state

$x\beta$ shows top of the stack

if δ denotes a unique transition for each ilp then PDA is said to be deterministic PDA

Ex:- i) $L = \{ a^n b^n \mid n \geq 1 \}$

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

Non deterministic PDA:-

It is a 7tuple like $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where

Q is finite and non empty set of states

Σ is finite and non empty set of ilp Alphabet

Γ is finite set of stack symbols.

δ is a mapping function used for moving from current state to next state and is defined as $\delta(q_0, x, z_0) = (q, \beta)$

q_0 is current state
x is current i/p symbol
z_0 is stack symbol
q is next state
$x\beta$ is top of the stack

if δ denotes more than one transition for a particular i/p symbol then the PDA is said to be non-deterministic PDA.

Ex:- $L = \{ ww^R \mid w = (a+b)^* \}$

Context free grammar and push down automata:-

Conversion of CFG to PDA

conversion of PDA to CFG.

i) Conversion of CFG to PDA:-

* For constructing a PDA from given CFG it is necessary to convert this CFG to some Normal form like GNF

* For converting given CFG to PDA, By this method the necessary condition is that the first symbol on RHS of production rule must be a terminal symbol. This rule that can be used to obtain PDA from CFG.

Algorithm:-

Rule 1:- for non-terminal symbols, add following rule

$\delta(q, \epsilon, A) = (q, \alpha)$ where the production rule is $A \to \alpha$.

Rule 2:- for each terminal symbols, add following rule

$\delta(q, a, a) = (q, \epsilon)$ for every terminal symbol 'a' in given CFG.

Ex:- construct a PDA for the given CFG $S \to 0BB$
$B \to 0S$
$B \to 1S$
$B \to 0$

Sol:- The given CFG $G = (V, T, P, S)$ where V = non-terminals $\{S, B\}$

$T = \{0, 1\}$

P :- $S \to OBB$
$B \to 0S$
$B \to 1S$
$B \to 0$

$S = \{S\}$

Rule 1 :- $A \to \alpha$
$\delta(q, \epsilon, A) = (q, \alpha)$

$S \to OBB$
$\delta(q, \epsilon, S) = (q, OBB)$

$B \to 0S$
$\delta(q, \epsilon, B) = (q, 0S)$

$B \to 1S$
$\delta(q, \epsilon, B) = (q, 1S)$

$B \to 0$
$\delta(q, \epsilon, B) = (q, 0)$

Rule :- 2

Terminals
$\delta(q, a, a) = (q, \epsilon)$
$T = \{0, 1\}$
$\delta(q, 0, 0) = (q, \epsilon)$
$\delta(q, 1, 1) = (q, \epsilon)$

∴ The corresponding PDA for the given CFG is defined as

$M = (Q, \Sigma, \tau, \delta, q_0, z_0, F)$

$Q = \{q\}$

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

$\tau = \{S, B, 0, 1\}$

δ = it is a transition symbol defined as

$\delta(q, \epsilon, S) = (q, OBB)$
$\delta(q, \epsilon, B) = (q, 0S)$
$\delta(q, \epsilon, B) = (q, 1S)$
$\delta(q, \epsilon, B) = (q, 0)$
$\delta(q, 0, 0) = (q, \epsilon)$
$\delta(q, 1, 1) = (q, \epsilon)$

$q_0 = \{q\}$

$z_0 = \{z_0\}$

$F = \{\}$

2) construct a PDA for the following CFG S→0S1
S→A
A→1A0|S|ϵ.

Sol: The given CFG is S→0S1
S→A
A→1AD
A→S
A→ϵ

elimination of ϵ-production:-

A→ϵ ×
S→A
S→ϵ ×
S→0S1
S→0ϵ1
S→01

A→1AD
A→1ϵ0
A→10
A→S
A→ϵ ×

∴ S→0S1|01
S→A
A→1A0|10
A→S.

elimination of unit productions:-

S→A ×
S→1A0|10

A→S ×
A→0S1|01

∴ The resultant CFG is.

S→1A0
S→10
A→0S1
A→01

∴ The Simplified CFG is.

S→1AD
S→1D
A→0S1|1A0|ϵ10
A→01
S→0S1|01

Method-1
P→1
Q→0

S→1A0 S→1D A→0S1 A→1D S→0S1
S→1AQ✓ S→1Q✓ A→0SP✓ A→1Q✓ S→0SP✓

A→1AD A→01 S→01
A→1AQ✓ A→0P✓ S→0P✓

∴ The simplified CFG in GNF is S→1AQ A→0SP
S→1Q A→1Q
S→0SP A→1AQ
S→0P A→0P
P→1
Q→0

Rule-1 ∴ The PDA is

S→1AQ.
δ(q, ϵ, S) = (q, 1AQ)

S→0SP
δ(q, ϵ, S) = (q, 0SP)

A→0SP
δ(q, ϵ, A) = (q, 0SP)

S→1Q
δ(q, ϵ, S) = (q, 1Q)

S→0P
δ(q, ϵ, S) = (q, 0P)

A→1Q
δ(q, ϵ, A) = (q, 1Q)

A → 1AQ

$\delta(q, \epsilon, A) = (q, 1AQ)$

A → 0P

$\delta(q, \epsilon, A) = (q, 0P)$

P → 1

$\delta(q, \epsilon, P) = (q, 1)$

Q → 0

$\delta(q, \epsilon, Q) = (q, 0)$

Rule 2:- Terminals

$\delta(q, a, a) = (q, \epsilon)$

$T = \{0, 1\}$

$\delta(q, 0, 0) = (q, \epsilon)$

$\delta(q, 1, 1) = (q, \epsilon)$

method:2

The Given CFG is S → 0S1

S → A

A → 1A0

A → S

A → ∈

The resultant PDA is S → 0S1

$\delta(q, \epsilon, S) = (q, 0S1)$

S → A

$\delta(q, \epsilon, S) = (q, A)$

A → 1A0

$\delta(q, \epsilon, A) = (q, 1A0)$

A → S

$\delta(q, \epsilon, A) = \delta(q, S)$

A → ∈

$\delta(q, \epsilon, A) = (q, \epsilon)$

Construct PDA for the following CFG S → aABB | aAA

A → aBB | a

B → bBB | A

sol:- The Given CFG is S → aABB

S → aAA

A → aBB

A → a

B → bBB

B → A

elimination of unit production :-

B → A ×

B → aBB

B → a

∴ After eliminating unit production B → A. The resultant CFG in GNF is S → aABB

S → aAA

A → aBB

A → a

B → bBB

B → aBB

B → a

The PDA is:

$S \to aABB$

$\delta(q, \epsilon, S) = (q, aABB)$

$S \to aAA$

$\delta(q, \epsilon, S) = (q, aAA)$

$A \to aBB$

$\delta(q, \epsilon, A) = (q, aBB)$

$A \to a$

$\delta(q, \epsilon, A) = (q, a)$

$B \to bBB$

$\delta(q, \epsilon, B) = (q, bBB)$

$B \to aBB$

$\delta(q, \epsilon, B) = (q, aBB)$

$B \to a$

$\delta(q, \epsilon, B) = (q, a)$

Conversion of PDA to CFG :-

* If $M = (Q, \Sigma, T, \delta, q_0, Z_0, F)$ is a PDA. then their exists CFG 'G' which is accepted by PDA (M).

Let 'G' be a CFG which is generated by PDA. The 'G' can be defined as $G = (V, T, P, S)$ where 'S' is the start symbol and the set of non-terminals $V = \{S, q, q', Z_0\}$ where $S, q, q' \in Q$ and $Z_0 \in T$.

Now, we get set of production rules using the following algorithm.

Algorithm :-

Rule 1 :- The start symbol production rule can be $S \to [q, Z_0, q']$

where q indicates present state

q' indicates next state.

Z_0 is the stack symbol.

Rule 2 :- If there exists a move of PDA as then the production rule can be return as $\delta(q, a, Z_0) = (q', \epsilon)$

$[q, Z_0, q'] \to a$

3) If there exists a move of PDA as $\delta(q, a, Z_0) = (q', Z_1 Z_2 Z_3 \ldots)$ then the production rules can be written as

$$[q, Z_0, q'] \rightarrow a[q, Z_1, q_1][q_1, Z_2, q_2][q_2, Z_3, q_3] \ldots [q_{n-1}, Z_n, q_n]$$

Q: construct a CFG from the following PDA $M = (\{q_0, q_1\}, \{0, 1\}, \{S, A\}, \delta, q_0, S, \phi)$ and

δ:

$\delta(q_0, 1, S) = (q_0, AS)$

$\delta(q_0, \epsilon, S) = (q_0, \epsilon)$

$\delta(q_0, 1, A) = (q_0, AA)$

$\delta(q_0, 0, A) = (q_1, A)$

$\delta(q_1, 1, A) = (q_1, \epsilon)$

$\delta(q_1, 0, S) = (q_0, S)$

Sol:- Let we will construct a CFG $G = (V, T, P, S)$ where

$T = \{0, 1\}$

$V = \{S, U, [q_0, S, q_0], [q_0, S, q_1], [q_1, S, q_0], [q_1, S, q_1], [q_0, A, q_0], [q_0, A, q_1], [q_1, A, q_0], [q_1, A, q_1]\}$

Now, Let us build the production rules as.

using rule ① the production rules for start symbol is

P_1: $S \rightarrow [q_0, S, q_0]$

P_2: $S \rightarrow [q_0, S, q_1]$

using Rule ③ of the Algorithm for the $\delta(q_0, 1, S) = (q_0, AS)$

$q_0 < q_0 q_0$ P_3: $[q_0, S, q_0] \rightarrow 1[q_0, A, q_0][q_0, S, q_0]$

$q_0 < q_0 q_1$ P_4: $[q_0, S, q_0] \rightarrow 1[q_0, A, q_1][q_1, S, q_0]$

P_5: $[q_0, S, q_1] \rightarrow 1[q_0, A, q_0][q_0, S, q_1]$

P_6: $[q_0, S, q_1] \rightarrow 1[q_0, A, q_1][q_1, S, q_1]$

Now, for $\delta(q_0, \epsilon, S) = (q_0, \epsilon)$ using Rule ② of Algorithm we get.

P_7: $[q_0, S, q_0] \rightarrow \epsilon$

P_8: ~~$[q_0, S, A] \rightarrow q_0 A$~~

Now for $\delta(q_0, 1, A) = (q_0, AA)$ using Rule ③ of Algorithm

P_8: $[q_0, A, q_0] \rightarrow 1[q_0, A, q_0][q_0, A, q_0]$

Next for $\delta(q_0, 0, A) = (q_1, A)$ using Rule ③ of Algorithm

P_9: $[q_0, A, q_0] \rightarrow 0\,[q_0, A, q_1]\,[q_1, A, q_0]$

P_{10}: $[q_0, A, q_1] \rightarrow 1\,[q_0, A, q_0]\,[q_0, A, q_1]$

P_{11}: $[q_0, A, q_1] \rightarrow 1\,[q_0, A, q_1]\,[q_1, A, q_1]$

Now, for $\delta(q_0, 0, A) = (q_1, A)$ using

$q_0 \begin{cases} q_0 \\ q_1 \end{cases}$ $q_1 \begin{cases} q_0 \\ q_1 \end{cases}$

P_{12}: $[q_0, A, q_0] \rightarrow 0\,[q_1, A, q_0]$

P_{13}: $[q_0, A, q_1] \rightarrow 0\,[q_1, A, q_1]$

Now, for $\delta(q_1, 1, A) = (q_1, \epsilon)$

P_{14}: $[q_1, A, q_1] \rightarrow 1$

Now, for $\delta(q_1, 0, S) = (q_0, S)$

$q_1 \begin{cases} q_0 \\ q_1 \end{cases}$ $q_0 \begin{cases} q_0 \\ q_1 \end{cases}$

P_{15}: $[q_1, S, q_0] \rightarrow 0\,[q_0, S, q_0]$

P_{16}: $[q_1, S, q_1] \rightarrow 0\,[q_0, S, q_1]$

PDA with two stacks :—

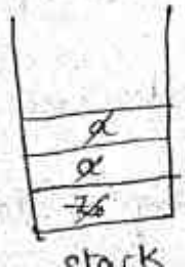
ⓐ PDA with one stack :

① $L = \{a^n b^n \mid n \geq 1\}$

Consider the string $w = aabb$

read a's → push

read b's → pop



a a b b $

when stack is empty then the string aabb is accepted

① $L = \{a^n b^n c^n / n \geq 1\}$

consider string $w = aabbcc$

read a's → push

read b's → pop

read c's → no change

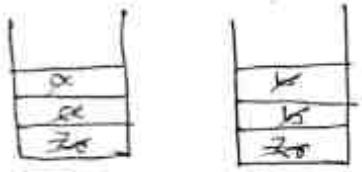
stack

a a b b c c $

when read c there is no change in stack without completion of reading string 'w' the stack is empty. so, string is not accepted.

② PDA with two stacks :—

$L = \{a^n b^n c^n / n \geq 1\}$



stack 1 stack 2

w = a a b b c c $

read a's → push (on $stack_1$)

read b's → push (on $stack_2$)

read c's → pop (a from $stack_1$ and 'b' from $stack_2$)

when two stacks are empty then string 'w' is accepted.

∴ The PDA with two stacks is more powerful than a ~~stack~~ PDA with one stack.

FA + 0-stack = NFA or DFA

FA + 1-stack = PDA.

FA + 2-stack = PDA with two stacks.

Applications of PDA :—

* used for deriving a string from the grammar.
* used for designing top-down parser and bottom-up parser in compiler design.
* It works on regular grammar and context-free grammars
* It accepts regular language and CFL.
* It has remembering capability by maintaining a stack.
* It is more powerful than FA.

3/3/18

UNIT-5

TURING MACHINE

Turing machine contains 7 tuples $TM = (Q, \Sigma, \Gamma, \delta, q_0, F, B)$

where Q = Set of states

Σ = set of input symbols

Γ = set of input tape symbols

δ = Transition function

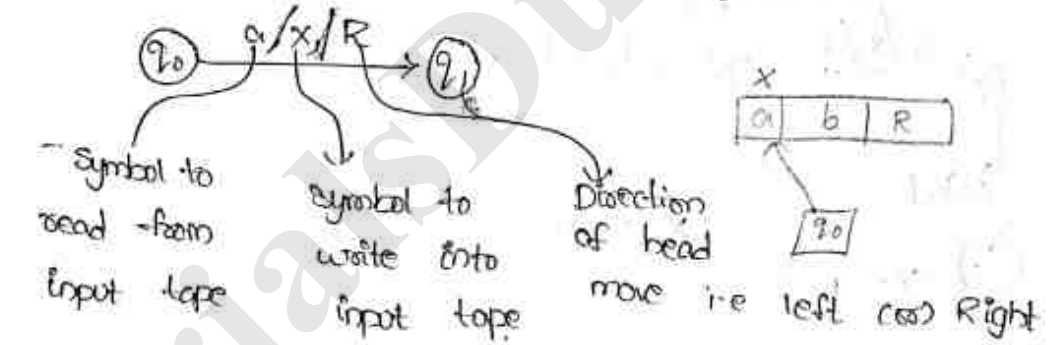
q_0 = initial state

F = final state

B = Blank symbol.

where δ can be defined as $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times (L/R)$

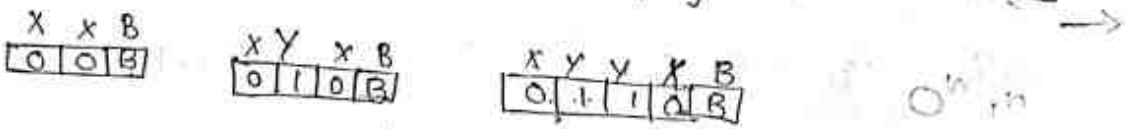
The Transition Diagram:- the transition function can be represented in the form of graphical notation.

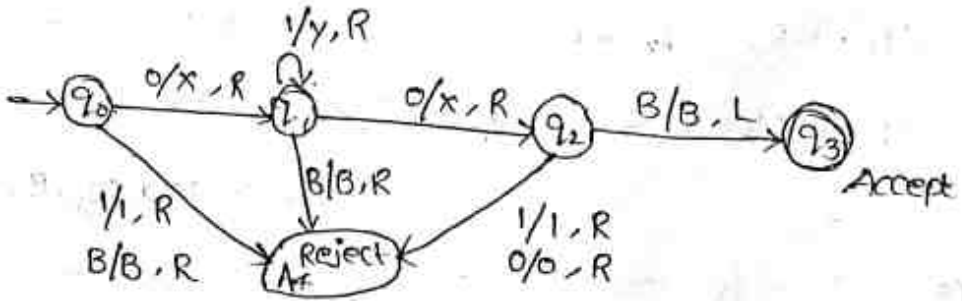


1) Design a turing machine for $L = 01^*0$

$L = 01^*0$

$L = \{00, 010, 0110, 01110, \cdots\}$



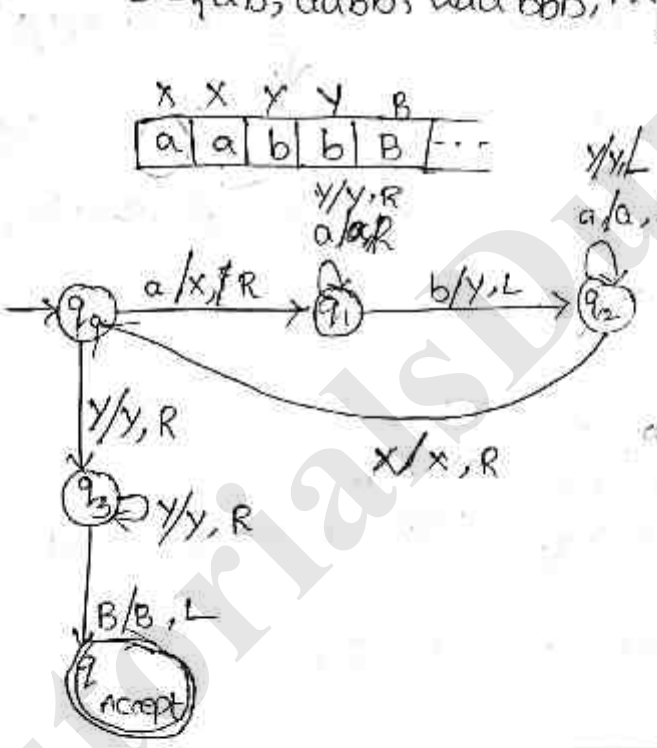


Tape symbol / State	0	1	X	Y	B
q_0	$\langle q_1, X, R\rangle$	$\langle q_4, 1, R\rangle$	-	-	$\langle q_4, B, R\rangle$
q_1	$\langle q_2, X, R\rangle$	$\langle q_1, Y, R\rangle$	-	-	$\langle q_4, B, R\rangle$
q_2	$\langle q_4, 0, R\rangle$	$\langle q_4, 1, R\rangle$	-	-	$\langle q_3, B, L\rangle$
q_3	-	-	-	-	-
q_4	-	-	-	-	-

*
**

2) Design a Turing Machine for language $L = \{a^n b^n / n \geq 1\}$

$L = \{ab, aabb, aaabbb, \ldots\}$



each 'a' is connected to 'b'

Tape symbol / State	a	b	X	Y	B
$\rightarrow q_0$	$\langle q_1, X, R\rangle$	—	—	$\langle q_3, Y, R\rangle$	—
q_1	$\langle q_1, a, R\rangle$	$\langle q_2, Y, L\rangle$	-	$\langle q_1, Y, R\rangle$	—
q_2	$\langle q_2, a, L\rangle$	—	$\langle q_0, X, R\rangle$	$\langle q_2, Y, L\rangle$	-
q_3	—	-	—	$\langle q_3, Y, R\rangle$	$\langle q_4, B, L\rangle$
* q_{Accept}	-	-	-	-	-

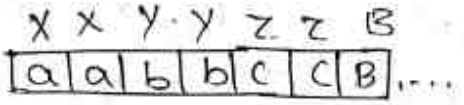
Transaction functions

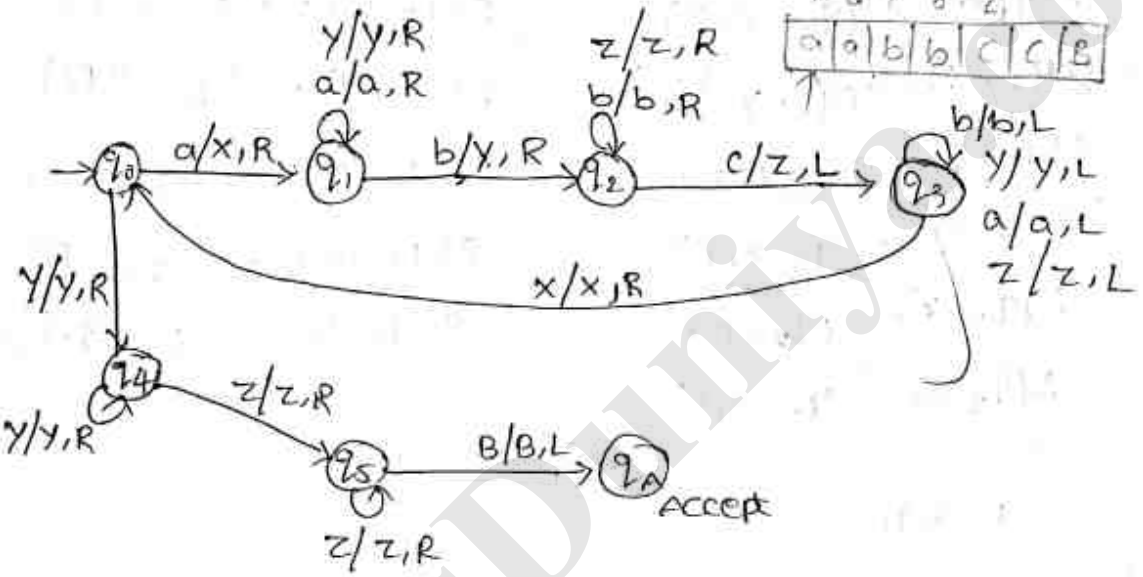
8

Design Turing machine for $L = \{a^n b^n c^n / n \geq 1\}$

$L = \{abc, aabbcc, aaabbbccc, \ldots\}$

$TM = \{Q, \Sigma, \delta, \Gamma, F, q_0, B\}$





Tape symbol / State	a	b	c	X	Y	Z	B
→ q_0	$\langle q_1, X, R\rangle$	–	–	–	$\langle q_4, Y, R\rangle$	–	–
q_1	$\langle q_1, a, R\rangle$	$\langle q_2, Y, R\rangle$	–	–	$\langle q_1, Y, R\rangle$	–	–
q_2	–	$\langle q_2, b, R\rangle$	$\langle q_3, Z, L\rangle$	–	–	$\langle q_2, Z, R\rangle$	–
q_3	$\langle q_3, a, L\rangle$	$\langle q_3, b, L\rangle$	–	$\langle q_0, X, R\rangle$	$\langle q_3, Y, L\rangle$	$\langle q_3, Z, L\rangle$	–
q_4	–	–	–	–	$\langle q_4, Y, R\rangle$	$\langle q_5, Z, R\rangle$	–
q_5	–	–	–	–	–	$\langle q_5, Z, R\rangle$	$\langle q_A, B, L\rangle$
q_A	–	–	–	–	–	–	–

TM: $M = \{Q, \Sigma, \Gamma, \delta, q_0, B, F\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_A\}$

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

$\Gamma = \{a, b, c, x, y, z, B\}$

$\delta(q_0, a) = (q_1, x, R)$

$\delta(q_0, y) = (q_4, y, R)$

$\delta(q_1, a) = (q_1, a, R)$

$\delta(q_1, b) = (q_2, y, R)$

$\delta(q_1, y) = (q_1, y, R)$

$\delta(q_2, b) = (q_2, b, R)$

$\delta(q_2, c) = (q_3, z, L)$

$\delta(q_2, z) = (q_2, z, R)$

$\delta(q_3, a) = (q_3, a, L)$

$\delta(q_3, b) = (q_3, b, L)$

$\delta(q_3, x) = (q_0, x, R)$

$\delta(q_3, y) = (q_3, y, L)$

$\delta(q_3, z) = (q_3, z, L)$

$\delta(q_4, y) = (q_4, y, R)$

$\delta(q_4, z) = (q_5, z, R)$

$\delta(q_5, z) = (q_5, z, R)$

$\delta(q_5, B) = (q_A, B, L)$

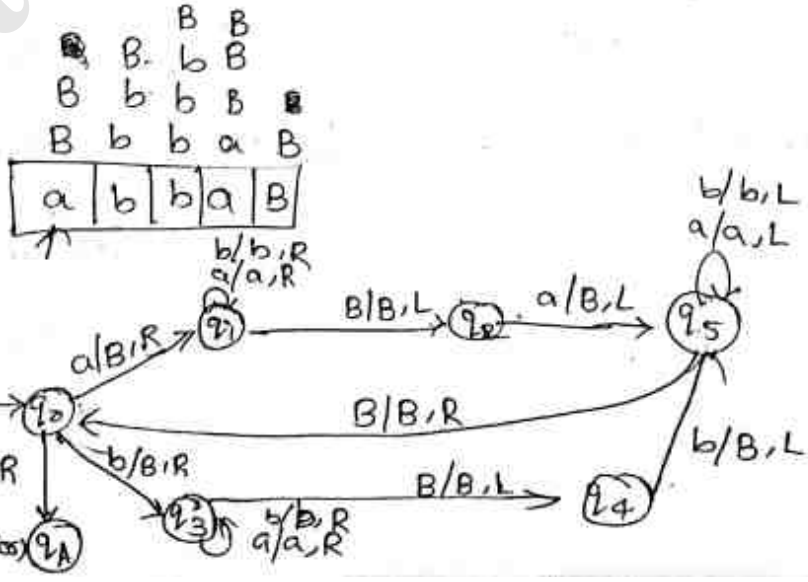
$F = \{q_A\}$

4) Design Turing Machine for $L = \{ww^R / w \in (a,b)^*\}$

Given $L = \{ww^R / w \in (a,b)^*\}$

It is a even length palindrome

$L = \{aa, bb, abba, baab, abbbba, abaaba, \ldots\}$



Tape Symbols / State	a	b	B
$\rightarrow q_0$	$\langle q_1, B, R\rangle$	$\langle q_3, B, R\rangle$	$\langle q_A, B, R\rangle$
q_1	$\langle q_1, a, R\rangle$	$\langle q_1, b, R\rangle$	$\langle q_2, B, L\rangle$
q_2	$\langle q_5, B, L\rangle$	-	-
q_3	$\langle q_3, a, R\rangle$	$\langle q_3, b, R\rangle$	$\langle q_4, B, L\rangle$
q_4	-	$\langle q_5, B, L\rangle$	-
q_5	$\langle q_5, a, L\rangle$	$\langle q_5, b, L\rangle$	$\langle q_0, B, R\rangle$
* q_A	-	-	-

(1) abba

q_0 a b b a B

$\vdash$ B q_1 b b a B

$\vdash$ B b q_1 b a B

$\vdash$ B b b q_1 a B

$\vdash$ B b b a q_1 B

$\vdash$ B b b q_2 a B

$\vdash$ B b q_5 b B B

$\vdash$ B q_5 b b B B

$\vdash$ B q_5 B b b B B

$\vdash$ B q_0 b b B B

$\vdash$ B B q_3 b B B

$\vdash$ B B b q_3 B B

$\vdash$ B B q_4 b B B

$\vdash$ B q_5 B B B B

$\vdash$ B B q_0 B B B

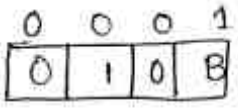
$\vdash$ B B B q_A B B

Accept

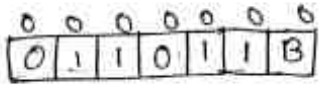
5) Design turing Machine for 'parity counter' that outputs '0' (or) '1' depending (or) on w

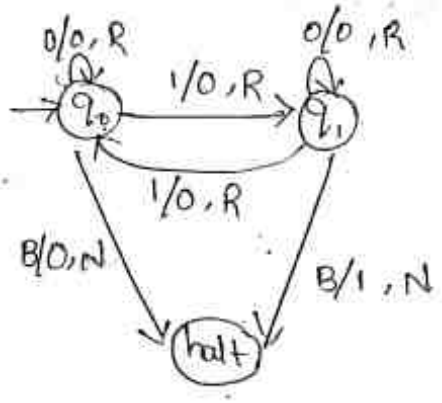
1's odd - 1

1's even - 0



Transition Table





odd 1's 010

$\vdash q_0 010$ B

$\vdash 0 q_0 10$ B

$\vdash 00 q_1 0$ B

$\vdash 000 q_1$ B

$\vdash 0001$ halt

even 1's 1010

$\vdash q_0 1010$ B

$\vdash 0 q_1 010$ B

$\vdash 00 q_1 10$ B

$\vdash 000 q_0 0$ B

$\vdash 0000 q_0$ B

$\vdash 00000$ halt

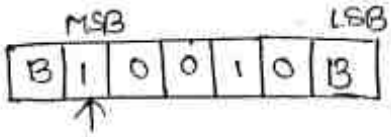
Design TM for 2's complement

I/P 10010

1's 01101

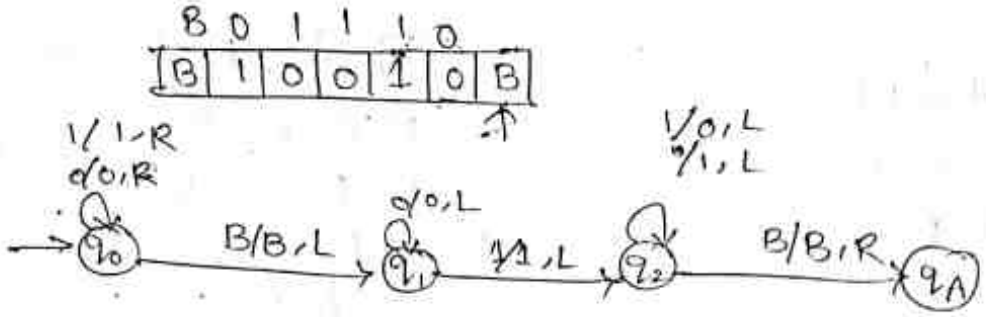
+1 1

2's 01110



accept
halt

First of all we have to move LSB then upto



ID:- B1 0 0 1 0 B

$\vdash B q_0 1 0 0 1 0 B$

$\vdash B 1 q_0 0 0 1 0 B$

$\vdash B 1 0 q_0 0 1 0 B$

$\vdash B 1 0 0 q_0 1 0 B$

$\vdash B 1 0 0 1 q_0 0 B$

$\vdash B 1 0 0 1 0 q_0 B$

$\vdash B 1 0 0 1 q_1 0 B$

$\vdash B 1 0 0 q_1 1 0 B$

$\vdash B 1 0 q_2 0 1 0 B$

$\vdash B 1 q_2 0 1 1 0 B$

$\vdash B q_2 1 1 1 1 0 B$

$\vdash B B 0 1 1 1 0 B$

$\vdash q_2 B 0 1 1 1 0 B$

$\vdash B q_2 0 1 1 1 0 B$

* (Addition of two Integers. Design TM)*

Design turing Machine for to accept set of all the palindromes.

Sol:

a	b	a	B

B b a B

B b B B

B B B

halt

b	a	b	B

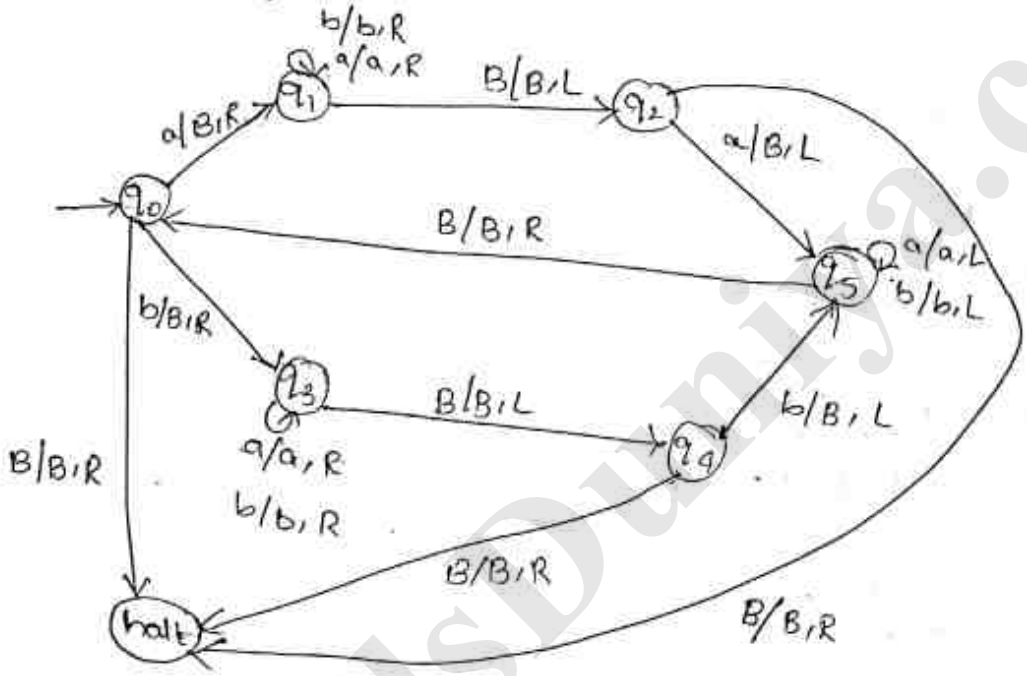
B a b B

B a B B

B B B

B B

halt



bab

⊢ q_0 bab B

⊢ B q_3 ab B

⊢ B a q_3 b B

⊢ B a b q_3 B

⊢ B a B q_4 b B

⊢ B q_5 a B B

⊢ q_5 B a B B

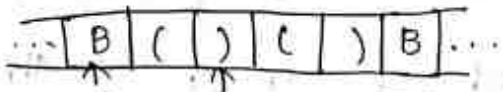
⊢ B q_0 a b B

⊢ B B q_1 B B

⊢ B q_2 B B B

⊢ B B q_A B B

Design Turing Machine for parenthesis checking



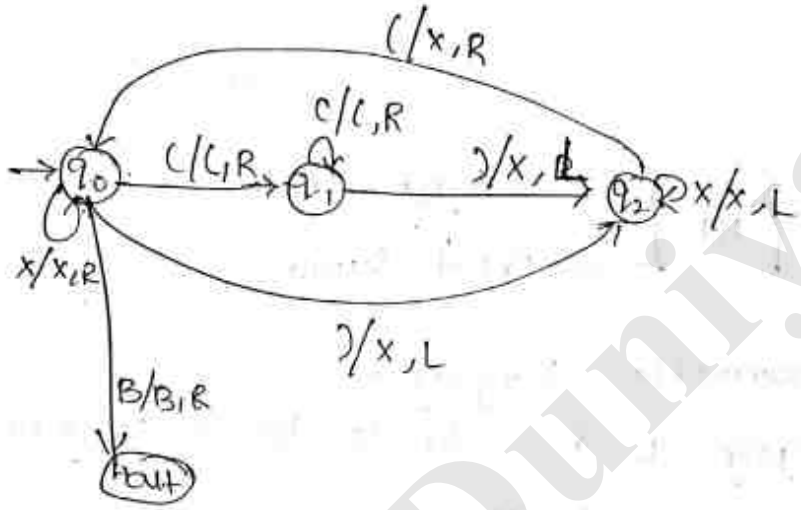
B (X ( ) B

B X X ( ) B

B X X (X B

B X X X X B

↑ B



⊢ q_0 (( )) B

(q_1 ( )) B

((q_1)) B

((q_2 X) B

(q_2 (X) B

(X q_0 X) B

(X X q_0) B

(X q_0 X X B

(q_2 X X X B

q_2 (X X X B

X q_1 X X X B

X X q_0 X X B

X X X q_0 X B

X X X X q_0 B

X X X X B ⊣

Types of Grammars - Chomsky Hierarchy:

Linguist Noam Chomsky defined a hierarchy of languages in terms of complexity. This four level hierarchy, called the Chomsky hierarchy, corresponds to four classes of machines.

The Chomsky hierarchy classifies grammars according to form of their productions into the following four levels.

(1) Type 0 grammars - unrestricted grammars
(2) Type 1 grammars - Context sensitive grammars
(3) Type 2 grammars - Context free grammars
(4) Type 3 grammars - regular grammars.

(1) Type-0 grammars - Unrestricted Grammars: (URG)

These grammars include all formal grammars. In URGs, all the productions are of the form $\alpha \rightarrow \beta$, where α and β may have any number of terminals and non-terminals. ie, no restrictions on either side of productions.

Every grammar is included in it if it has atleast one non-terminal on the left hand side.

Ex:- $aA \rightarrow abCB$
$aA \rightarrow bAA$
$bA \rightarrow a$
$S \rightarrow aAb \mid \epsilon$

They generate exactly all languages that can be recognized by a turing machine. The language that is recognized by a Turing machine is defined as set of all the strings on which it halts. These Languages

are also known as the recursively enumerable languages. (2)

(2) Type 1 grammar – Context Sensitive Grammars: (CSG)

These grammars define the context-sensitive languages. In Context Sensitive grammar, all the productions of the form $\alpha \to \beta$, where length of α is less than or equal to the length β. i.e $|\alpha| \leq |\beta|$. α and β may have any number of terminals and non-terminals. These languages are exactly all the languages that can be recognized by linear bound automata.

Ex:- $|\alpha| \leq |\beta|$ $\alpha \to \beta$

$aAbcD \to abcDbcD$

(3) Type 2 Grammar - Context-free Grammar (CFG)

These grammars define the context-free languages. These are defined by rules of the form $\alpha \to \beta$ with $|\alpha| \leq |\beta|$ where $|\alpha| = 1$ and α is non-terminal and β is a string of terminals and non-terminals. we can replace α by β regardless of where it appears. Hence the name context free grammar. These languages are exactly those languages that can be recognized by a pushdown automaton. CFG's context free languages defines the syntax of all programming languages.

Ex:- i) $S \to aS \mid Sa \mid a$

ii) $S \to aAA \mid bBB \mid \epsilon$.

(4) Type 3 grammars - regular grammars:

These grammars generate the regular languages. Such a grammar restricts its rules to a single non-terminal on the LHS. the RHS consists of either a single terminal or string of terminals with single non-terminal on left or right end.

$\alpha \to \beta$, $\alpha = \{V\}$
$\beta = V\{T\}^* \text{ or } \{T\}^* V \text{ or } T^*$

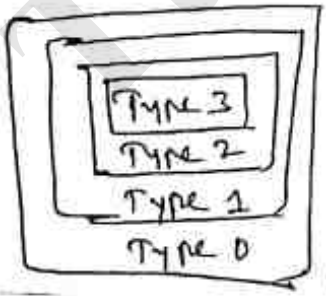
Eg: $A \to aB|a$ — right linear grammar

$A \to Aa|a$ — left linear grammar.

\# Every regular language is context free, every context-free language is context-sensitive and every context-sensitive language is recursively enumerable.

Table: Chomsky's hierarchy

Grammar	Language	Automaton	production rules
Type 0	Recursively enumerable	Turing machine	$\alpha \to \beta$ no restrictions on β, α α should have at least one non-terminal
Type 1	Context-sensitive	Linear bounded automata	$\alpha \to \beta$ $\lvert\alpha\rvert \le \lvert\beta\rvert$
Type 2	Context-free	push down automata	$\alpha \to \beta$ $\lvert\alpha\rvert = 1$
Type 3	Regular	Finite state automatons	$\alpha \to \beta$, $\alpha = \{V\}$ $\beta = \{V\}T^*$ or $= T^*\{V\}$ $= T^*$



chomsky hierarchy of grammars

(3)

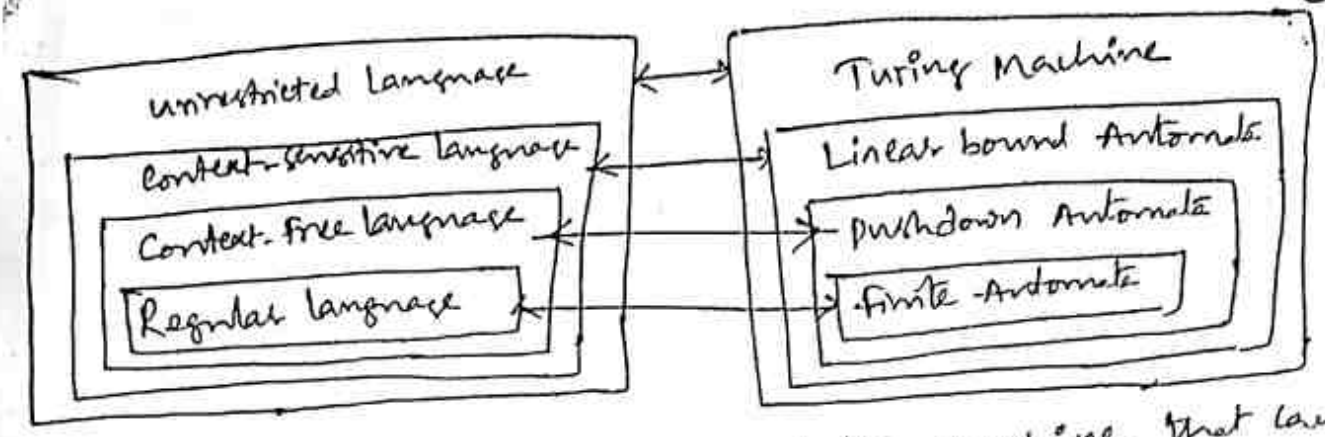


fig: The hierarchy of languages and the machine that can recognize the same as shown above fig.

- Every RG is context free, every CFG is context-sensitive and every CSG is unrestricted. So the family of regular language can be recognized by any machine.
- CFL's are recognized by pushdown automata, linear bounded automata and Turing machines.
- CSLs are recognized by Linear bounded automata and Turing machines
- Unrestricted languages are recognized by only Turing machine

(4)

Push Down Automata (PDA)

(68)

PDA – FA + Stack
memory element

PDA

DPDA NDPDA

PDA = $(Q, \Sigma, \delta, q_0, Z_0, F, \Gamma)$ (7 Tuple)

Q = finite set of states

Σ = input symbol

δ = Transition function

q_0 = Initial state

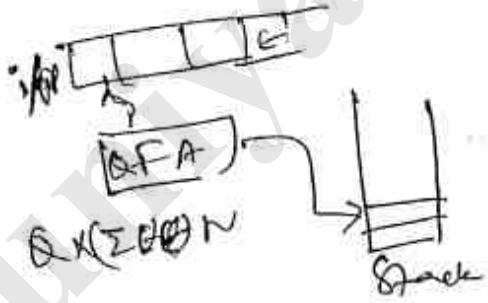
Z_0 = Bottom of the stack

F = set of final states

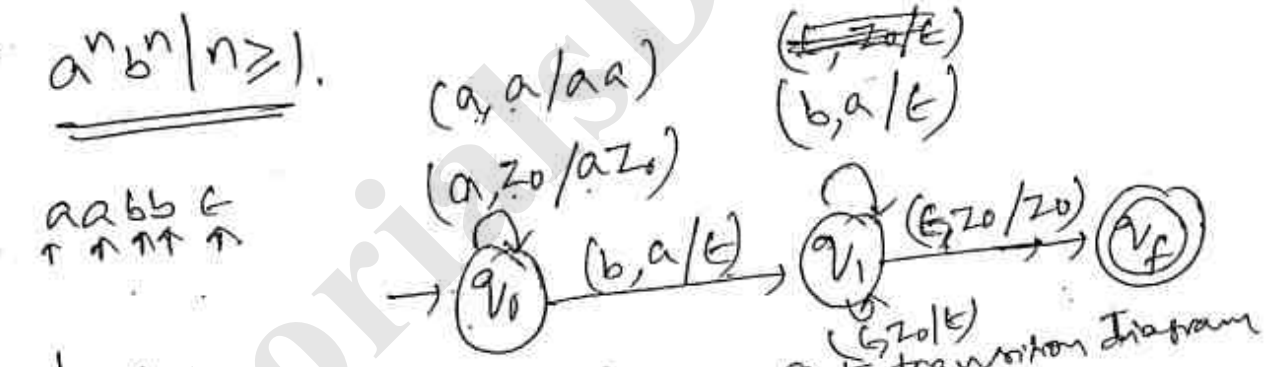
Γ = stack alphabet.

DPDA: $\delta : Q \times \{\Sigma \cup \epsilon\} \times \Gamma \to Q \times \Gamma$
(State, Input, Top)

NDPDA $\delta : Q \times \{\Sigma \cup \epsilon\} \times \Gamma \to 2^{(Q \times \Gamma)^*}$
(State, Input, Top)



Ex: $a^n b^n \mid n \geq 1$.



State transition Diagram

DPDA

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

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

$\delta(q_0, b, a) = (q_1, \epsilon)$

$\delta(q_1, b, a) = (q_1, \epsilon)$

$\delta(q_1, \epsilon, Z_0) = (q_f, Z_0)$ or (q_1, ϵ)

accepting final state. Acceptance by empty stack

transition function

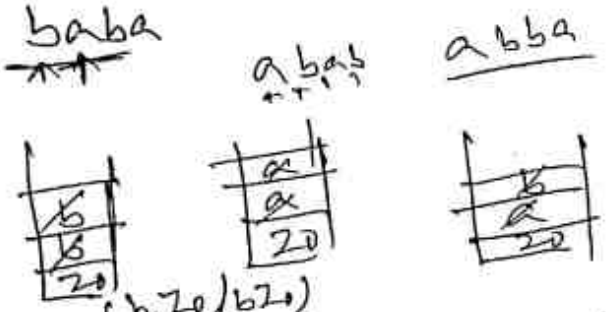
PDA — final state
Empty stack.

$\{w | n_a(w) = n_b(w)\}$ [number of a's must be equal].

DPDA

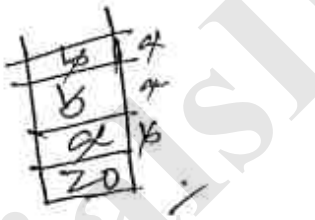
Ex: a b bbaa
aabb baba
abab

baba



abba

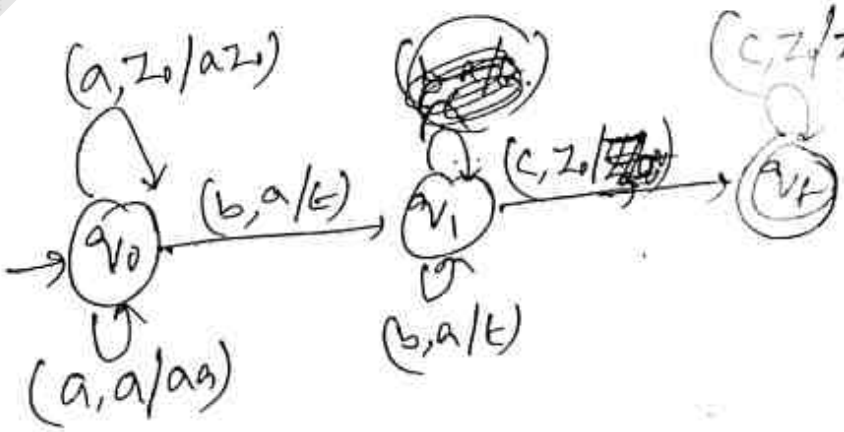
abbbaa



$a^n b^n c^m \mid n, m \geq 1$.

PDA

a^n → push
b^m → pop
c^m → no stack use

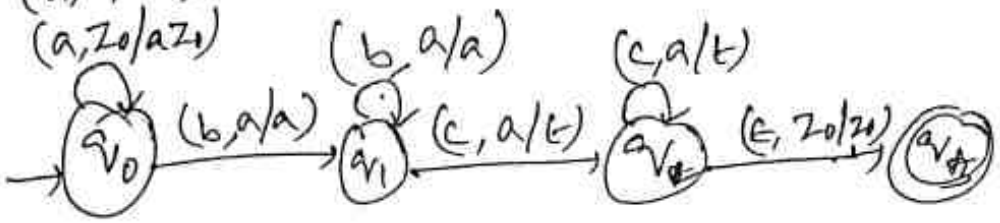


$a^n b^m c^n \mid n, m \geq 1$

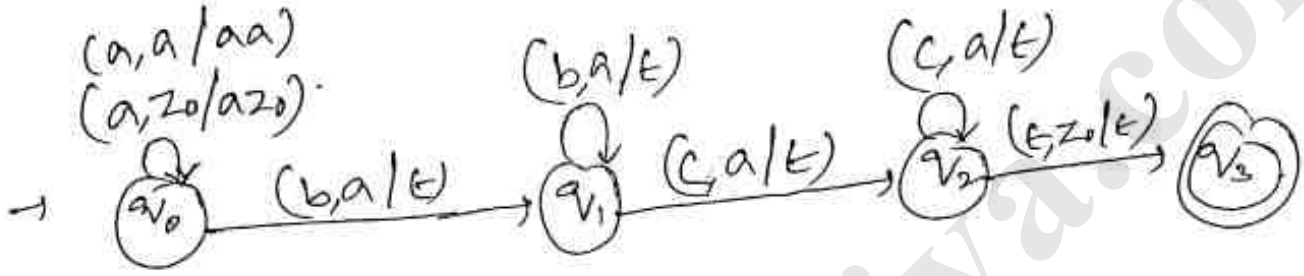
dont want count b

aaabbbccc

Sol:



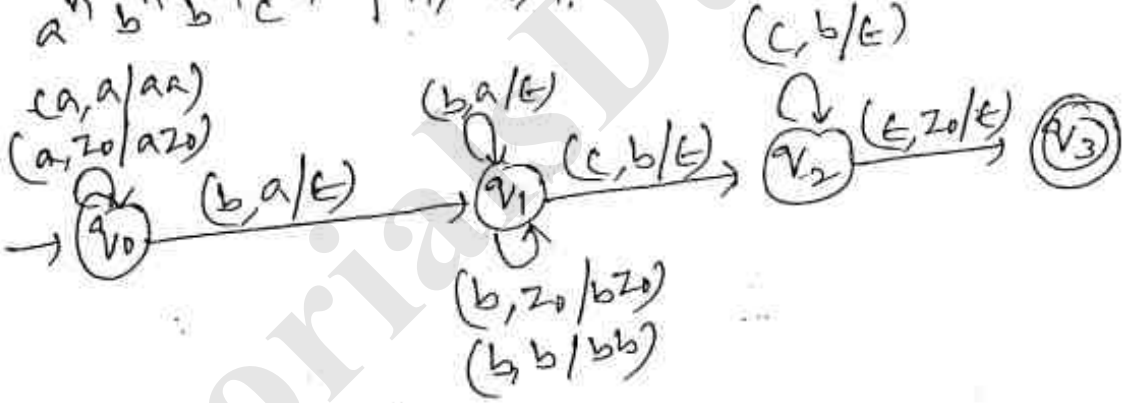
$a^{m+n} b^m c^n \mid m, n \geq 1$



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

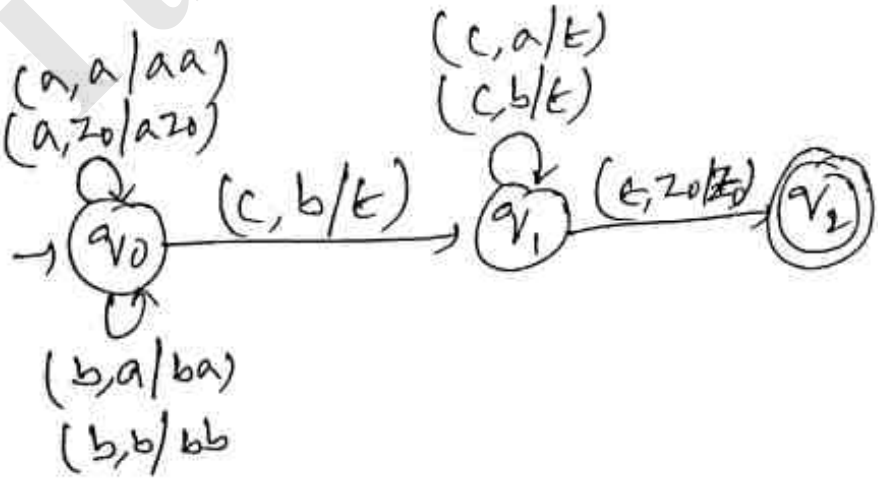
Sol:

$a^n b^n b^m c^m \mid n, m \geq 1.$



$a^n b^m c^{n+m} \mid n, m \geq 1.$

Sol:



$a^n b^n c^m d^m \mid n, m \geq 1$

$a^n b^m c^m d^n \mid n, m \geq 1$

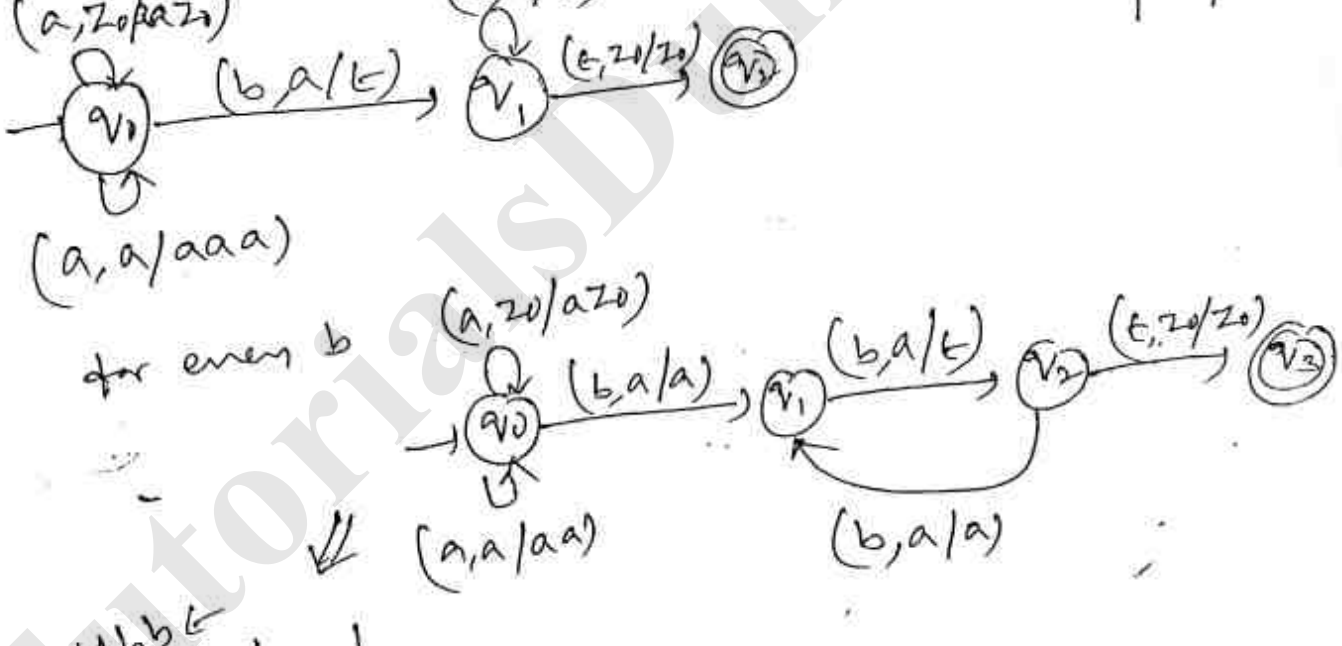
$a^n b^m c^n d^m \mid n, m \geq 1$ ✗ is not CFL not PDA

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

1) abb, aabbbb, aaabbbbbb, ...

two solutions → for every a – push two a's.
→ for every b – pop two b's

0



aabbbb

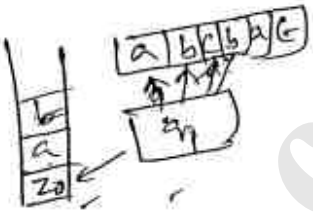
Ex: $a^n b^n c^n \mid n \geq 1$ ✗ not a PDA

one possibility

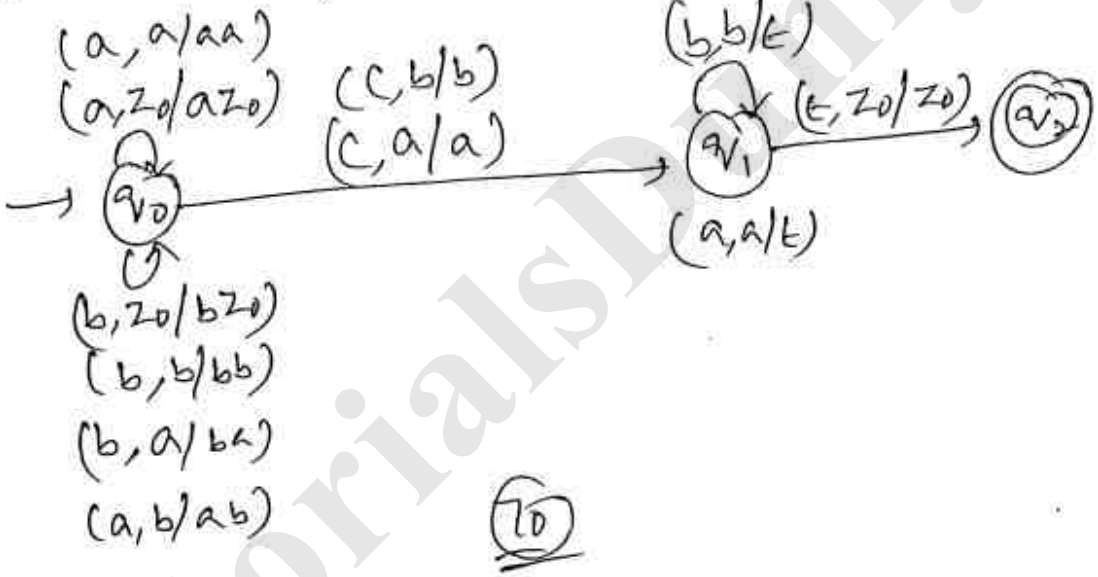
✗ aa bb cc

for every a - two ab pairs

Ex: $wcw^R \mid w \in (a,b)^+$



Ex: abcba, abbcbba

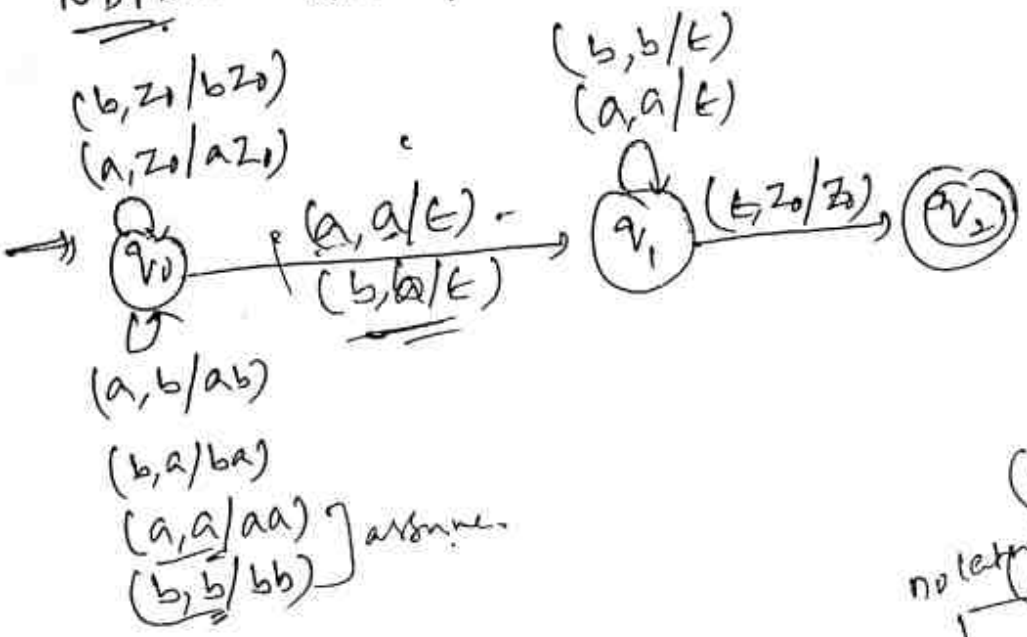


10

Ex: $ww^R \mid w \in (a+b)^+$ → whenever

aba aba
w w^R

NDPDA $ww^R \mid w \in (a+b)^+$



abba

Instantaneous Descriptions

aaaa

(q0, aaaa, Z0)

(q0, aaa, aZ0)

no centre | centre

(q0, aa, aaZ0) (q1, aa, Z0) ×

no centre | centre

(q0, a, aaaZ0) (q1, a, aZ0)

no centre | centre

(q0, ε, aaaaZ0) (q0, ε, aaZ0) ×

(q1, ε, Z0)

(q2, Z0)

aabaab

baa aab

aab baa

w w

aba aba

w wR

Centre

NDPDA

Centre has come pop

Centre not Come push

q0

Change → getting → centre

ab | ba.

UNIT-VI

Language decidability

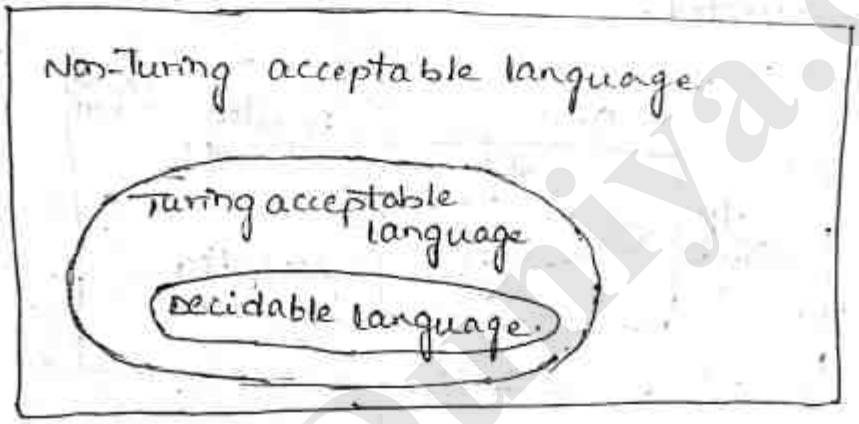
* Introduction
* Examples

Introduction:-

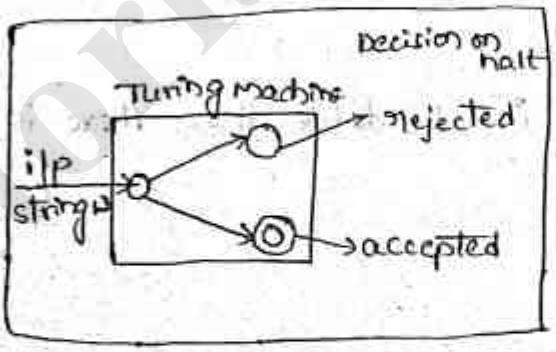
- Decidable problem:-

* A language is called Decidable (or) recursive if there is a turing machine which accepts and halts on every ilp string "w"

* Every decidable language is a turing acceptable



* A decision problem 'p' is decidable if the language 'L' of all "yes" instances to 'p' is decidable.
* for a decidable language, for each ilp string, the turing machine halts either at the accept (or) the reject state



Examples:-

1) find out whether the following problem is decidable (or) not.

Is a number 'm' prime?

Sol:- Prime numbers = {2, 3, 5, 7, 11, 13, 17, 19, ...}

divide the number 'm' by all the numbers b/w

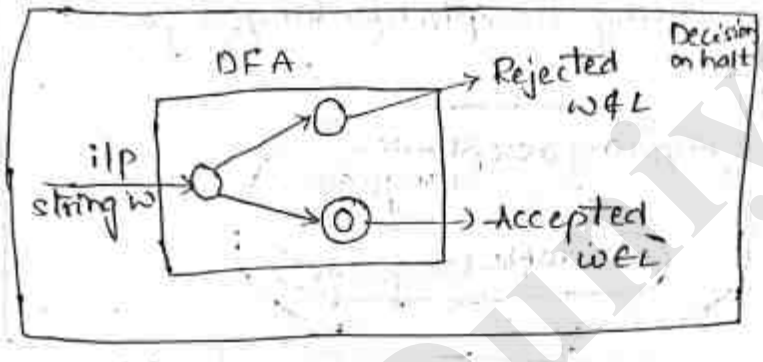
2 and m/2 · starting from 2.

If any of these numbers produce a remainder 0. then it goes to the rejected state. otherwise it goes to the accepted state. so, here the answer could be made by YES (or) NO.

Hence, it is a decidable problem.

2) Given a Regular language 'L' and string 'w', how can we check if $w \in L$.

sol:- Take the DFA that accepts 'L' and check if 'w' is accepted.



Some more decision problems are

i) Does DFA accept the empty language

ii) Is $L_1 \cap L_2 = \phi$ for regular sets.

iii) If a language L is decidable then its complement L' is also decidable.

iv) If a language is decidable then there is a turing machine for it.

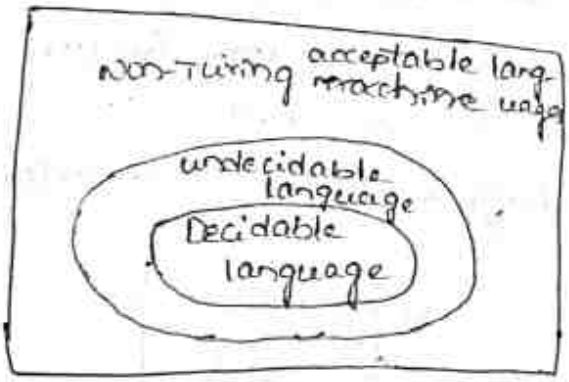
Undecidable problems:-

Introduction:-

* for an undecidable language there is no "TM" which accepts the language and makes a decision for every ilp string 'w'.

* A decision problem 'p' is called undecidable if the language 'L' of all 'YES' instances to "p" is not decidable.

undecidable languages are not recursive languages but, sometimes they may be recursive enumerable languages.



examples:—

i) the halting problem of turing machine.

ii) the mortality problem.

iii) the mortal matrix problem.

iv) the post correspondence problem [pcp]

i) The halting problem:

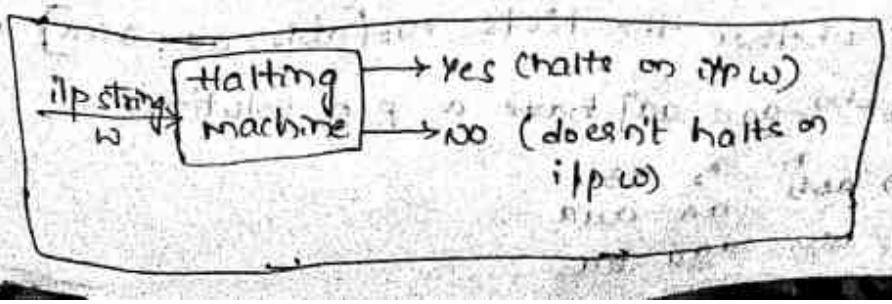
The halting problem i/p: a turing machine and the i/p string w.

problem: Does the turing machine finish computing of the string 'w' in a finite no. of steps? The answer must be either yes (or) NO.

Proof:— At first, we will assume that a turing machine exists to solve, the problem. we will show and then it is contradicting it self.

We will call this turing machine as a halting machine that produces a yes (or) NO. in a finite amount of time.

If the halting machine finishes in a finite amount of time then the o/p comes as yes. otherwise, as NO

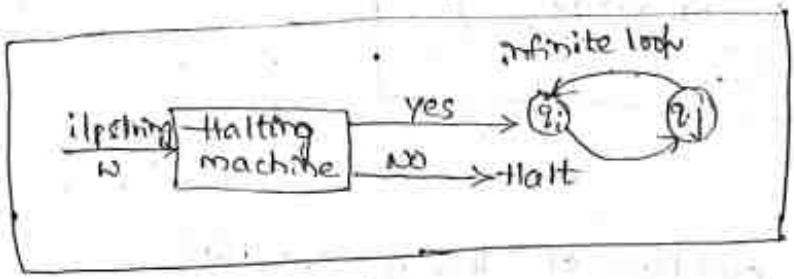


Now, we will design an inverted halting machine as

i) If HM returns yes then loop forever.

ii) If HM returns NO then halt.

The following block diagram shows the inverted halting machine.



Further, a machine "HM" which ilp itself is constructed as follows.

i) If HM halts on ilp loop forever.

ii) Else Halt

∴ Here, we have got a contradiction. Hence, the halting problem is undecidable.

Post Correspondence problem (PCP):—

- It was introduced by "Emil post" in 1946 is as undecidable decision problem.

The pcp problem over an ilp alphabet "Σ" is stated as follows.

Given the following two lists, M, and N. of non-empty strings over Σ $M=(x_1, x_2, x_3 \cdots x_n)$

$N=(y_1, y_2, y_3 \cdots y_n)$

We can say that there is a pcp solution if for some $i_1, i_2, i_3 \cdots i_K$ where $1 \le i_K \le n$, The condition.

$$x_{i_1} x_{i_2} x_{i_3} \cdots x_{i_K} = y_{i_1} y_{i_2} y_{i_3} \cdots y_{i_K}$$ satisfies

Ex:- Find whether the lists $M=(abb, aa, aaa)$ and $N=[bba, aaa, aa]$ have a PCP solution.

Sol:-

	x_1	x_2	x_3
M	abb	aa	aaa
N	bba	aaa	aa

Here $x_2 x_1 x_3 = aaabbaaa$

$y_2 y_1 y_3 = aaabbaaa$

we can see that $x_2 x_1 x_3 = y_2 y_1 y_3$

Hence, the solution is $i=2, j=1, k=3$.

2) find whether the list $M = [ab, bab, bbaaa]$ and $N = [a, ba, bab]$ have a pcp solution

	x_1	x_2	x_3
M	ab	bab	bbaaa
N	a	ba	bab

sol:- In this case, there is no solution because

$|x_2 x_1 x_3| \neq |y_2 y_1 y_3|$ length's are not same.

Hence, it can be said that this pcp is a undecidable problem.

Modified post correspondence problem:-

Given two lists $M = x_1, x_2, x_3 \cdots x_n$ and $N = y_1, y_2, y_3 \cdots y_n$

Given a set of pairs of strings $\{(x_1, y_1), (x_2, y_2), \cdots (x_n, y_n)\}$

then the solution is an instance such that,

$$x_1 x_{i_1} x_{i_2} \cdots x_{i_n} = y_1 y_{i_1} y_{i_2} \cdots y_{i_n}$$

that means the pair (x_1, y_1) is forced to be at the begining of the strings.

Ex:-

	x_1	x_2	x_3
M	11	100	111
N	111	001	11

sol:- ∴Then the solution is $x_1 x_2 x_3 = y_1 y_2 y_3$.

$x_1 x_2 x_3 = 11100111$

$y_1 y_2 y_3 = 11100111$

That means it is essential to have x_1 & y_1 at the beginning of list

P and Np Classes :-

* P-problems
* Np-problems
* P vs Np

- **P-problems:-**

* P is the class of problems that can be solved by deterministic algorithm in a polynomial type time $p(n^k)$ where 'n' is the size of i/p string.
* P-problem consist of a language accepted by deterministic Turing machine that runs in polynomial amount of time.

Ex:- 1) shortest path problem
2) Equivalence of NFA and DFA.
3) shortest cycle in a graph.
4) sorting algorithms

- **Np-problems:-**

* Np-problem is a class of problems that can be solved by Non-deterministic algorithms in a polynomial time $p(n^k)$ where 'n' is the size of i/p string.
* Np-problems consists of a language accepted by Non-deterministic turing machine that runs in a polynomial amount of time.

Ex:- 1) Travelling salesman problem.
2) subgraph isomorphism

Np-problem classified into two types.

i) Np-hard problem.
ii) Np- complete problem.

- **Np-hard problem:-**

If there is a language x such that every language y in Np can be polynomially reduceable to x. and we cannot prove that x is in Np. then x is said to be Np-hard problem.

Ex:- Turing machine halting problem.

- **Np- Complete problem:-**

If there is a language x such that every language

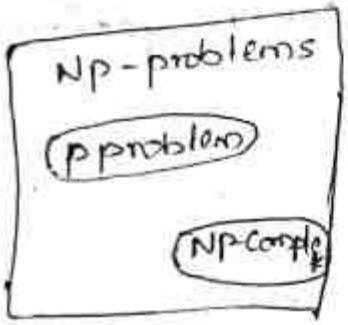
y in NP can be polynomially reducable to x and we can prove that x is in NP then x is said to be NP-complete problems.

Eg:- 1) Travelling salesman problem.
2) subgraph isomorphism.

P Vs NP :-

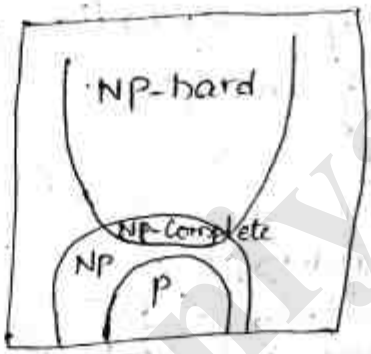
1. Ladner's theorem:

(a) $P \neq NP$



2. Euler's theorem:-

(a) $P \neq NP$



(b) $P = NP$