

SRI SHIVA XEROX
#147, RPS Tower, Opp. JSS College,
Below UCOBank, Dr. Viswanatharao Road,
Reflex Layout, Bangalore - 560 060.
Ph: 9620099557, 7676670591.

(+) Many natural substances such as ergogenic molecules and computer metabolites can be modelled as graphs and the solution can be obtained using very efficient graph search algorithms

(e) Artificial Intelligence programs are used to solve various problems arising from medical diagnosis to daily scheduling

(5) DNA molecules can be analyzed with the help of finite state machines and context free grammars

(4) used to design interactive video games

(3) used to design vending machines (such as ATMs), semiautomatic protocols and also used in building security devices.

(2) used to design and implement modern programming languages such as C/C++/ Java/ Python etc.

(1) used to design communication protocols, HTML

and machine communication and man/machine communication.

Applications of theory of computation:

11/12 March

20-02-2014
CS 152 II

$$Z^+ = Z$$

$$\text{Note: } Z^* = Z \cup E$$

$$Z^+ = \{0, 1, 00, 01, 10, 11, 000, 001, \dots\}$$

If $Z = \{0, 1\}$, then

i.e. Z^* is not part of the Z^+ and hence $E \notin Z^*$.
String (e)

If Z is the set of words of any length except the null

$$Z^* = Z \cup Z^+ \cup Z^3 \cup Z^5 \cup \dots$$

Kleene Plus (Z^+) :- It is defined as follows -

$$Z^+ = \{ \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots \}$$

$$Z^* = \{ \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots \}$$

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

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

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

Z^* is defined as shown below:

$$\text{If } Z = \{0, 1\}, \text{ then}$$

from Z .

If Z is the set of words of any length (possibly empty or the null string) : each string is made up of symbols only

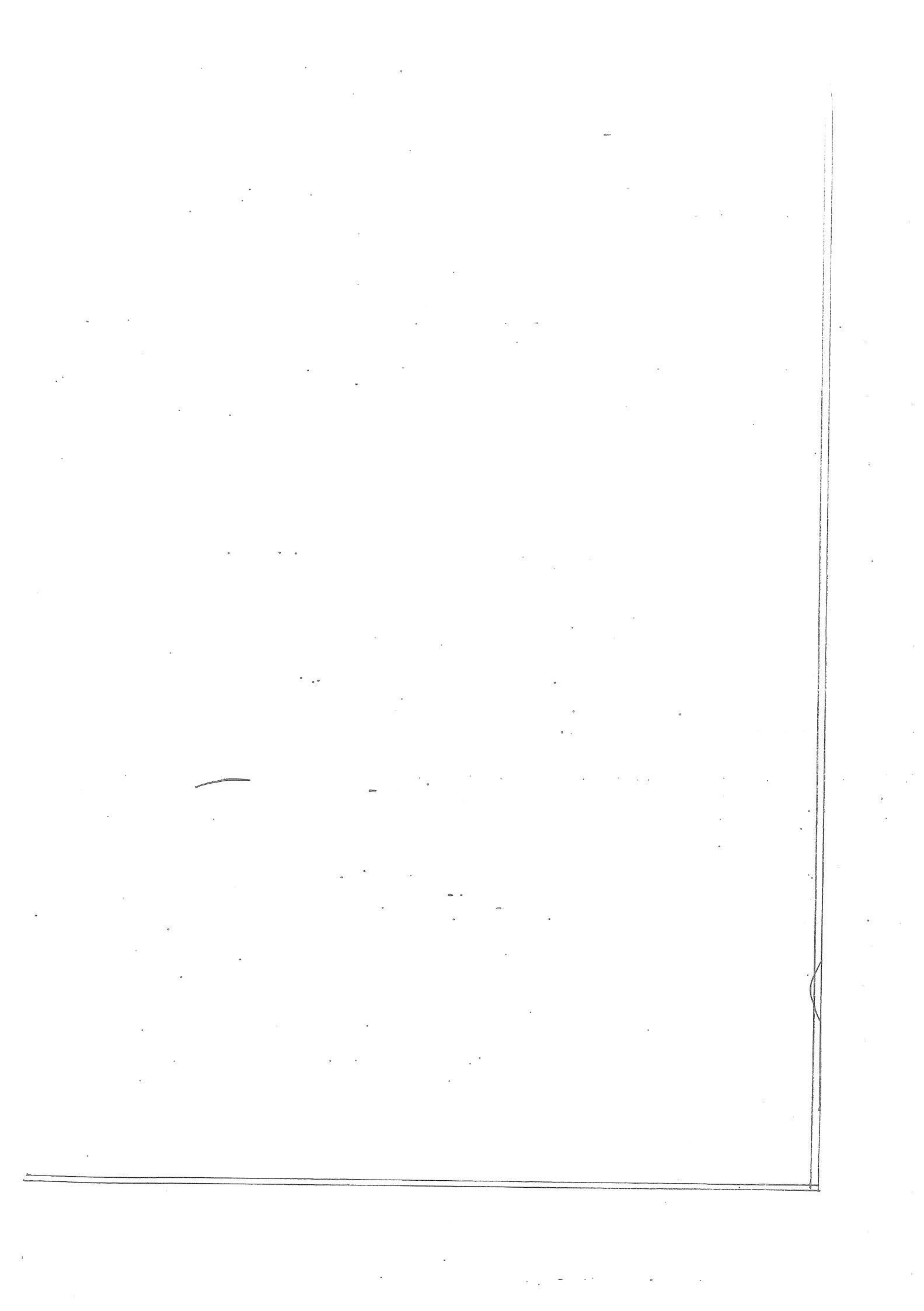
$$Z^* = Z^0 \cup Z^1 \cup Z^2 \cup \dots$$

Kleene Closure (Z^*) :- It is defined as follows -

a. Kleene Plus

b. Kleene Closure (Kleene Star)

What are the two variations of powers of an alphabet?



$L = \{00, 01, 10, 11, 011, 100, 010, 110, 111\}$ set of words of length 3

$L = \{0, 1\}$ set of words of length 1

$L = \{0\}$ set of words of length 0

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

denoted by L^*

Power of an alphabet: It is the set of words of length i .

Note: An alphabet often denoted L as a finite set.

when L :

The set of all possible strings over an alphabet L

The empty string is denoted by ϵ

For example, if $L = \{a, b\}$ then various strings that can be formed from L are $\{aa, ab, aab, bba, ba, \dots\}$. An infinite number of strings can be generated from this set.

Things: saying as finite collection of symbols from alphabet.

The other examples are sets of alphabets and more.

Here 0 and 1 are alphabets denoted by ω

The elements a, b, \dots, z are the alphabets.

$S = \{a, b, c, \dots, z\}$ English

The example of alphabet is.

We cannot define symbol formally.

Symbol: An alphabet is a finite collection of

languages and strings

$$9993 = 999$$

$$(bba)^2 = bba$$

$$a^3 = aaa$$

$$w_{H^2} = w_H$$

$$w_0 = e$$

3. Recursion: For each string w and each natural number i , the string w^i is defined as:

↳ $w^0 = \epsilon$ (empty string). So the identity for concatenation is maintained as a function defined on strings x and y such that $x = y = \epsilon$.

↳ $w^1 = w$. So if $w = h$ then $w^1 = h$ and $w^0 = \epsilon$. It is good and $w^1 = h$ is good because $|h| + |\epsilon| = |h|$.

4. Concatenation: The concatenation of two strings s and t , written $s \parallel t$ or $s t$, is the string formed by appending t to s .

$$\#(abbaba) = 4$$

Let any symbol a and strings s , we define the function $\#(s)$ to be the number of times that the symbol a occurs in s .

$$\#(101101) = 4$$

$$|e| = 0$$

5. Length: In this we are going to find the number of symbols in strings. It is represented as $|s|$.

Functions on strings:

$L_1 \cap L_2 = \{y^R : x \in L_1 \wedge y \in L_2\}$ - intersection
 $(A \cap (B \cap C)) = A \cap B \cap C$ - commutative law
 $x = ab, y = cd, (xy)^R = dcba$
 Proof: If x and y are strings then

Theorem If L_1 and L_2 are languages then $(L_1 L_2)^R = L_2^R L_1^R$

$x^R = badc$
 $y^R = dcba$
 Then define $w^R = aub$
 (i.e., the last character of w is a)
 If $|w| \geq 1$ then $\exists m \in \mathbb{Z} : m < |w|$
 $\cdot ((m=m) * z \in L \Rightarrow z^R = w^R)$
 If $|w|=0$ then $w^R = w = \epsilon$

we will write w^R , as defined as:
4. Result: For each string w , the reverse of w , which

E, a, ab are people prefixes of string abb
A string s is as a people suffix of string ab
A string s is a people suffix of a string t and s ≠ t.
E.g. 32 is a people suffix of string 123
E.g. 67 is a people suffix of string 1234567
E.g. 99 is a people suffix of string 123456789
E.g. 99 is a people suffix of string 1234567890

aa is a substring of aaabbbaaa
 aaaa is not a substring of aaabbbaaa
 A. string is a repeating substring of a string t
 It is a substring of t and is t
 Every string is a substring of itself
 The empty string, i.e., is a substring of every string
 If S is good then people substances are

Patterns on surfaces : A surface is a stretching of a surface as far as possible so as to cover completely a surface.

$L = \{w \in \{a\}^*: |w| \text{ is even}\} \leftarrow$
 Languages not in L are:
 $\{b, ba, e, bbbb, baaa\}$
 Languages not in L are:
 $L = \{a, ab, aab, abbba, \dots\}$
 \leftarrow Let $L = \{w \in \{a, b\}^*: \text{all strings begin with } a\}$
 Example: (by enumeration)
Techniques for defining languages:

Note:- Number of languages that can be defined even
 from single alphabet is infinite.
 Language $L_4 = \{a, -aaa, a\overline{a}, \overline{a}b, \dots\}$
 Language $L_3 = \{a, aaaa, aaaaaaaa, \dots\}$
 Language $L_2 = \{e, aa, aaa, aaaa, \dots\}$
 Language $L_1 = \{a, aaa, aaaa, aaaaaaa, \dots\}$
 $\therefore L = \{a\}$

language defined on alphabet L
 we will use the notation L for more than one
 set of strings over a finite alphabet L .
Languages: A language is a (finite or infinite)

L is the language that contains no strings

Let $L = \{ \} = \emptyset$

Example : The empty language

$\{ ab, ab^2, ab^3, ab^5, aaab^8, ab^9, \dots \}$

Strings not in L are :

$L = \{ e, a, a^2, a^3, a^4, \dots \}$

Let $L = \{ w \in \{a, b\}^* : \text{no prefix of } w \text{ contains } b \}$

All the strings that can be formed with $\{a, b\}^*$

Strings not in L are :

$L = \{ ba, bb, bba, \dots \}$

Let $L = \{ w \in \{a, b\}^* : \text{every prefix of } w \text{ starts with } b \}$

Example : Using some property

$\{ e, a^2, a^3, a^5, a^6, a^8, a^9, \dots \}$

Strings not in L are :

$L = \{ a, a^4, a^7, a^9, \dots \}$

Let $L = \{ w \in \{a\}^* : |w| \bmod 3 = 1 \}$

$\{ b, ab, e, bbb, \dots \}$

Strings not in L are :

$L = \{ a, aba, aabaa, bbbba, \dots \}$

Let $L = \{ w \in \{a, b\}^* : \text{all string ending in a} \}$

Example : (defining property)

P = Power set

Theorem: An uncountably infinite number of languages are countably infinite.

Proof: The elements of \mathbb{Z} can be paired with \mathbb{N} . Since there is no longest string in \mathbb{Z} , there are examples of strings of every length. Then there is an ∞ of strings of every length, enumerable than in dictionary order. Within the same enumeration is an infinite subset of strings that are all strings of length 0, then length 1, and so on. This enumeration is in bijective correspondence with the elements of \mathbb{Z} since there is no longest string in \mathbb{Z} . Since these strings are all languages over Σ as infinite sets, \mathbb{Z} is a subset of all languages over Σ .

Theorem: All languages over Σ are countably infinite.

Proof: The set of languages over Σ is defined as $P(\Sigma^*)$. If Σ is countably infinite, then Σ^* is also countably infinite by the pumping lemma. If Σ is uncountably infinite, then Σ^* is also uncountably infinite by the pumping lemma. Therefore, the set of languages over Σ is uncountably infinite.

Example: The empty language is different from \emptyset . Let $L = \{\epsilon\}$, the language that contains a single string, ϵ . Note that L is different from \emptyset because the empty string is a single string.

$$\phi = \gamma\phi = \phi\gamma$$

In general for L

$$Lg = gL$$

$$(Lg) - gL = 0$$

$$\text{Note: } Lg = gL$$

$$Lg < (axx, aayy, abxx, abyy)$$

$$Lg < (xx, yy)$$

$$Lg = (aa, bb)$$

Concateration \sqcup Languages:

$$L = (L - L)^{\sim} \quad // \text{Complement operation}$$

$$Lg = (L - L)^{\sim} \quad // \text{Complement operation}$$

$$Lg = L - L \quad // \text{Difference operation}$$

$$Lg = L - L \quad // \text{Difference operation}$$

$$Lg = \emptyset \quad // \text{Intersection operation}$$

$$L \cup Lg = L \quad // \text{Union operation}$$

$$Lg < (a_1, a_3, a_5, a_7, a_9, \dots) \quad // \text{odd no. of a's}$$

$$Lg = (e, a_2, a_4, a_6, a_8, a_{10}, a_{12}, \dots) \quad // \text{even no. of a's}$$

$$Lg \quad \text{if } L = \{a\}$$

can be applied

Like union, intersection, Difference and Complement operations are also. Therefore all set of operations

Functions in Languages:

$$a^+ = a^* - \{e\}$$

$$a^* = \{e, a, aa, aaa, aaaa, \dots\} \text{ infinite}$$

Example

How?

$$L^+ = L^* - \{e\} \leftarrow$$

$$L^+ = L^* \leftarrow$$

What is $L^+?$

Concatenation of Languages:

$$a^* = \{e, a, aa, aaa, aaaa, \dots\} \text{ infinite}$$

Kleen star operation
 $L^* = \{\text{set of all strings that can be formed by concatenating zero or more strings from } L\}$

$$\text{Note: } L^* = \{a^n \mid n \geq 0\} = \{a^m b^n \mid m \geq 0, n \geq 0\} = \{a^m b^n c^p \mid m \geq 0, n \geq 0, p \geq 0\}$$

$$\text{Note: } L^* = \{a^n \mid n \geq 0\}$$

$$\text{Note: } L^* = L(L^*) = L(L^2) = L^2(L) \leftarrow \text{associative}$$

$$L^* = \{e\}^* = \{e\}^n \leftarrow$$

In general for all L

$$L^* = L^0$$

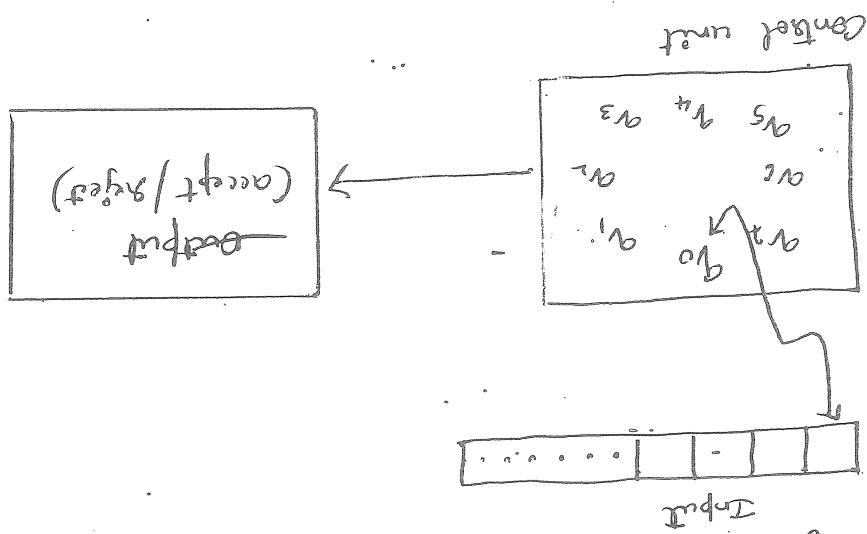
$$L^* = \{x \in A^* \mid \exists y \in A^* \text{ such that } x = yz \}$$

$$\text{Note: } L^* = \{e\}^*$$

Output: When end of input is encountered and if the string is accepted by the machine and if the machine is not in final state, the output by the machine. But, when end of input is encountered machine is in final state, the input string is discarded.

Control Unit: It consists of a number of states. The states are defined by q_0, q_1, \dots, q_n . Initially, the machine will have different some final state. Based on the current input character it changes its state and the current state of the machine, it may lead to in final state to and the machine will have different some final state. Based on the current input character it changes its state.

Input: The string to be processed is placed in input.



$$g(a_0, 1) = a_0$$

$$g(a_0, 0) = a_1$$

This is expressed as

reading a 0 at 1 end is the state a_1
that the machine is in state a_0 on
state a_0 to state a_1 . This indicates
an arrow with label 0 of 1 goes from



$$g(a_0, 0) = a_0$$

state a_0 on reading a 0, remains in same
in a_0 . This indicates the machine is in
an arrow with label 0 starts and ends



$$g(a_0, 1) = a_1$$

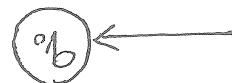
an arrow with label 1 goes from state a_0



Two codes are used to represent a
final state.

A code with an arrow which is not
overlapping from any node represents the
state of machine if no code represents a
node of machine.

meaning



Notations used during designing of FA

- What are the different types of Finite state machines?
- Deterministic Finite Automata (DFA)
- Non-deterministic Finite Automata (NFA)
- Non-deterministic Finite Automata (e-NFA)
- With e - moves (e-NFA)
- Consider the following pictorial representation of DFA:
- States: The above DFA has three states q_0 , q_1 , and q_2 . The following are the components of DFA:
- a) States: The above DFA has three states q_0 , q_1 , and q_2 .
- b) Start State: q_0 with an arrow labeled start.
- c) Final State: q_1 and q_2 with two arrows labeled final state.
- d) Input: Each edge is labelled with a symbol.
- e) Transitions: It is nothing but change of state consuming an input symbol. $g(q_i, a) = q_j$
- f) Alphabet: Consisting of all the characters which a DFA can accept.
- g) Output: Each edge is labelled with a symbol.
- h) States: $g(q_i, a) = q_j$
- i) $g(q_0, a) = q_1$
- j) $g(q_1, a) = q_2$
- k) $g(q_2, a) = q_0$
- l) $g(q_0, b) = q_2$
- m) $g(q_2, b) = q_1$
- n) $g(q_1, b) = q_0$
- o) $g(q_0, c) = q_0$
- p) $g(q_1, c) = q_1$
- q) $g(q_2, c) = q_2$
- r) $g(q_0, d) = q_1$
- s) $g(q_1, d) = q_0$
- t) $g(q_2, d) = q_2$
- u) $g(q_0, e) = q_0$
- v) $g(q_1, e) = q_1$
- w) $g(q_2, e) = q_2$
- x) $g(q_0, f) = q_0$
- y) $g(q_1, f) = q_1$
- z) $g(q_2, f) = q_2$

$f \leftarrow$ set of accepting states
 $q_0 \leftarrow$ start state
 $\delta \leftarrow$ transition function which is a mapping from $Q \times Z$ to Q
 $Z \leftarrow$ non-empty set of input alphabets
 $Q \leftarrow$ non-empty finite set of states
 $M \leftarrow$ name of the machine
 where,

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

Note: The Deterministic Finite Automaton is short
 DFA is DSm is 5-tuple of quintuple induction
 five components:

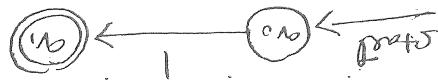
What is a DFA/DSM?
 Note: The transition function δ accepts two parameters
 namely state and input symbol and return a next state.

It means, the change of state from state q to
 $\delta(q, a) = p$

would lead as " q " is a transition function
 $Q : Q \times Z$ to Q

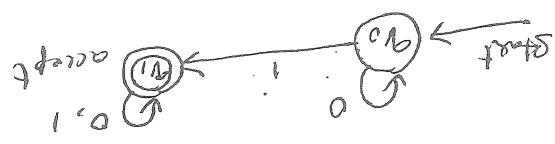
The transition function is defined as:
 what is a transition function?

transitions from state q_1
any of the input symbol
from state q_1 is not a legal
input symbol. we say
that q_1 is the transition
from q_1 to a legal
state q_2 .



shown below:

Sometimes, there can be no transition from a state
or to a state. then we say
the transition is undefined from a state
obtained from the above diagram that, those extra
symbols from the above diagram than a state in an input symbol.



Note:- The DFA can have only one transition from a
state in an input symbol. Consider the following DFA -

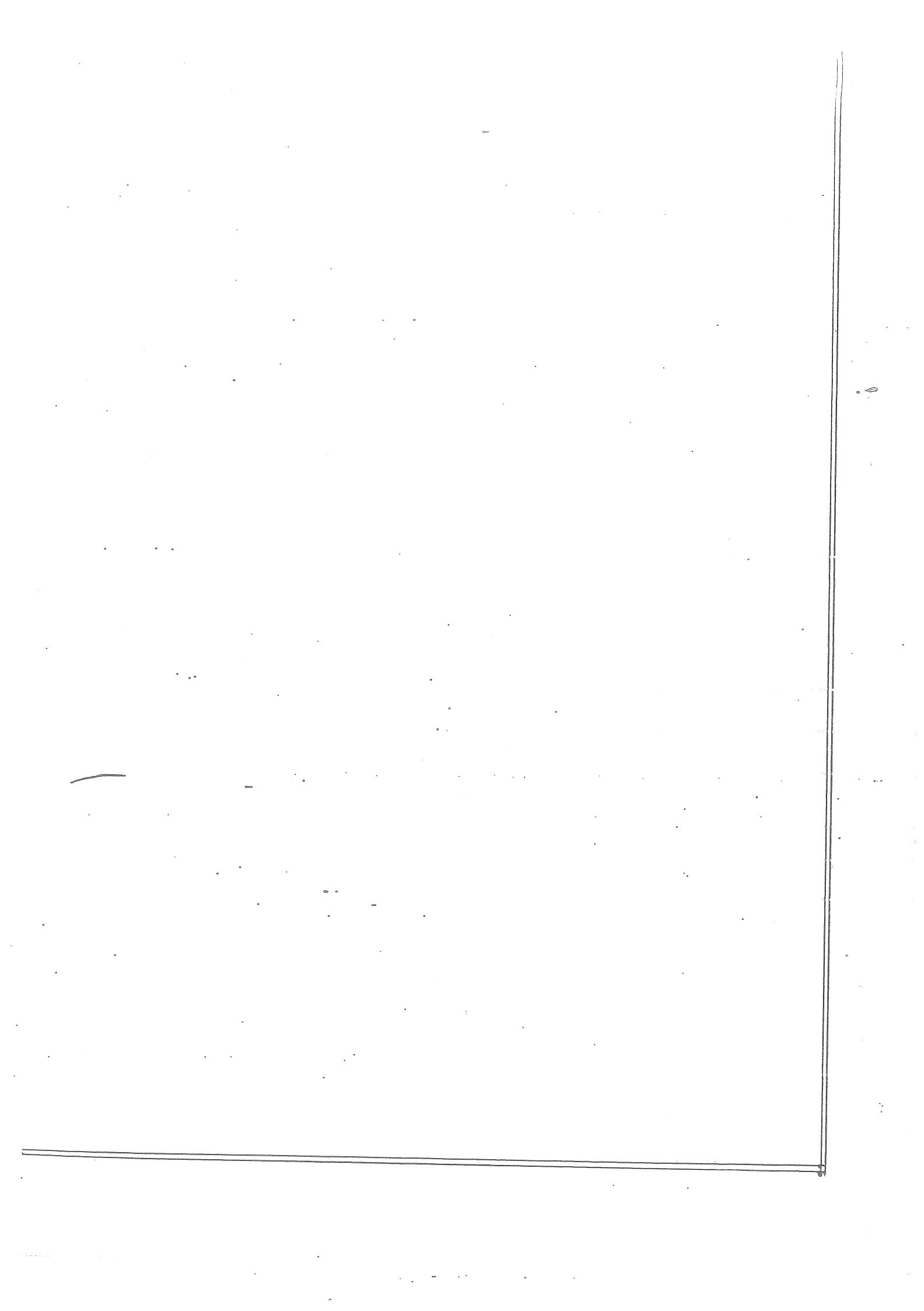
A (Automaton) - Automaton is a machine which may
accept the string or reject the string.

F (Finite) - Has finite number of states and edges.

Machine is deterministic
enters into after consuming the input symbol. So, the
possible to determine exactly to which state the machine
gets every input symbol from the state. So, it is

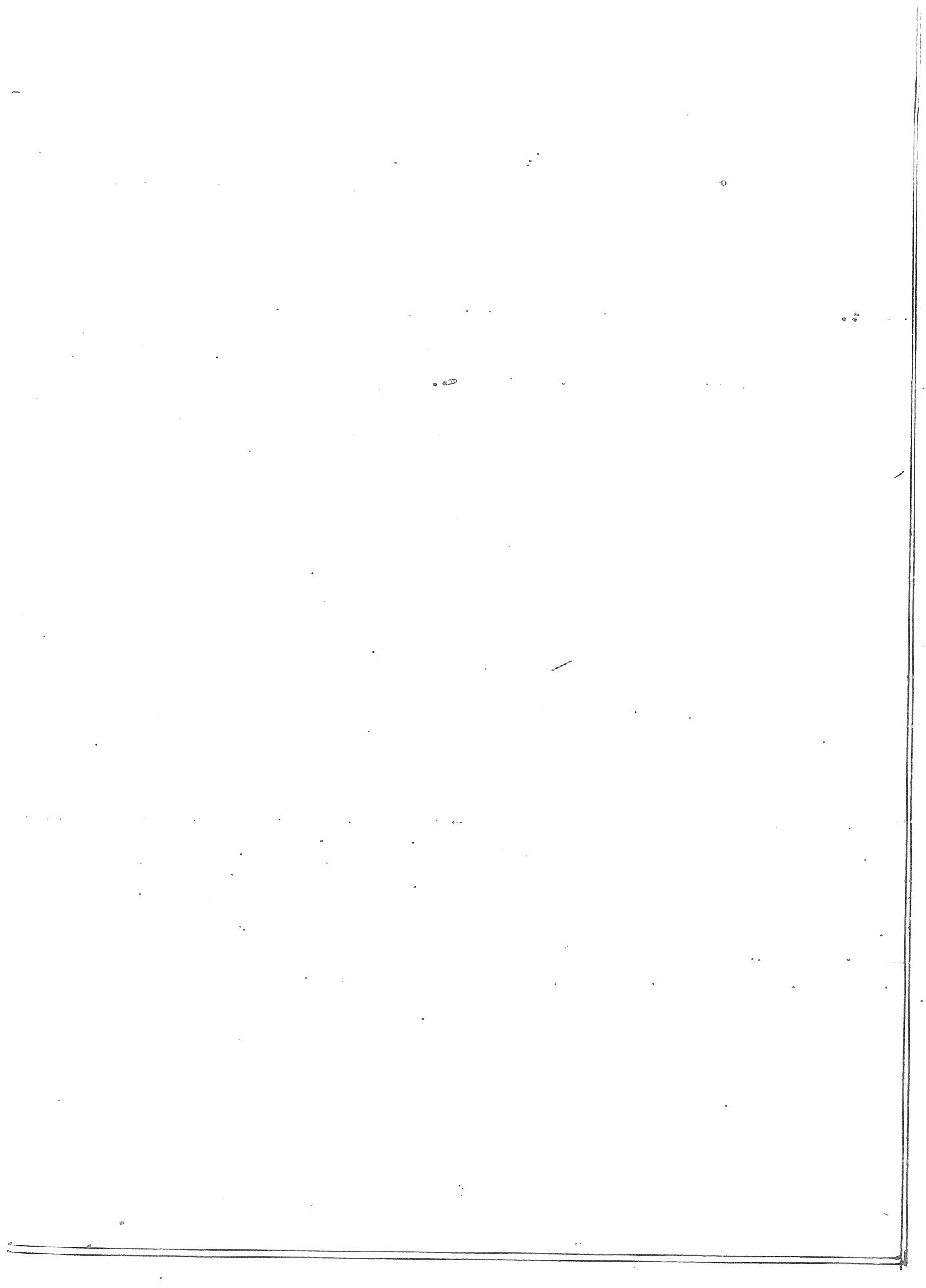
D (Deterministic) - There is exactly one transition

The name DFA comes from the following fact:

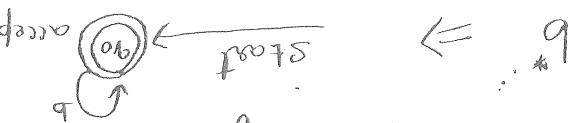


How to make a Transition diagram?

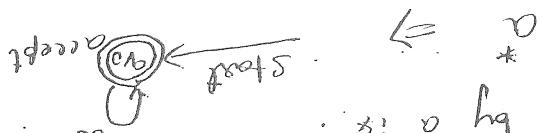
- Steps are:
- 1) Find the min. acceptable strings accepted. This decides the numbers of states in fsm in most of the cases.
 - 2) Then take longer strings and make them accepted by modifying the transitions.
 - 3) Check for minimum strings that are not to be accepted are ready not accepted as per the transition diagram.
 - 4) See that each state has transition equal to the number of alphabets present
 - 5) Two transition on the same alphabet do not go to different states.



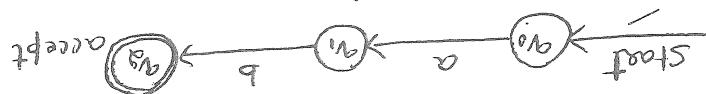
6) DFA to accept string consisting of any number of 'a's or 'b's



5) DFA to accept string consisting of any number of 'a's as well as 'b's.



Note :- To accept the symbol 'a' we require two states. In general, to accept two symbols such as 'ab', we require three states. To accept two symbols such as 'ab', we require two states.



4) DFA to accept string 'ab'



3) DFA to accept exactly one 'a'

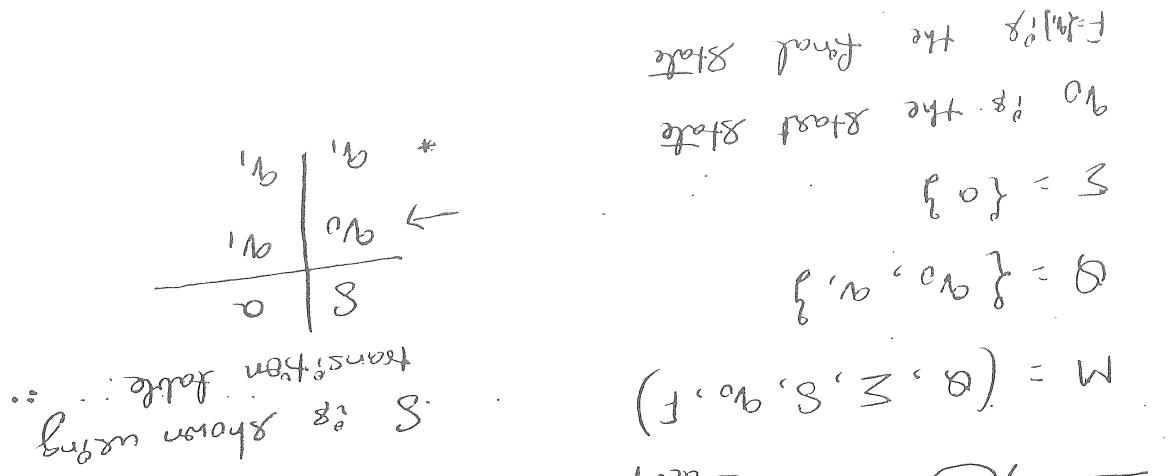


2) DFA to accept an empty string $L = \{\epsilon\}$



1) DFA to accept empty language $L = \emptyset$

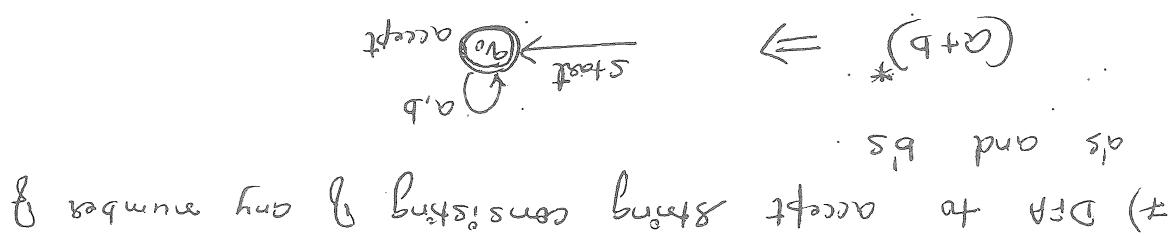
DFA for simple language :



- Pattern recognition problems:
1. Pattern recognition problems
 2. Applicable by λ of problems
 3. $|w| \leq k$ problems
 4. $|w| \leq k$ problems with additional operations
 5. $N_a(w) \leq k$ problems
 6. $N_a(w) \leq m$ if problems with logical operations
- Ex: a is one
- solution:

What are the various problems which we can construct DFA?

DFA Design techniques:



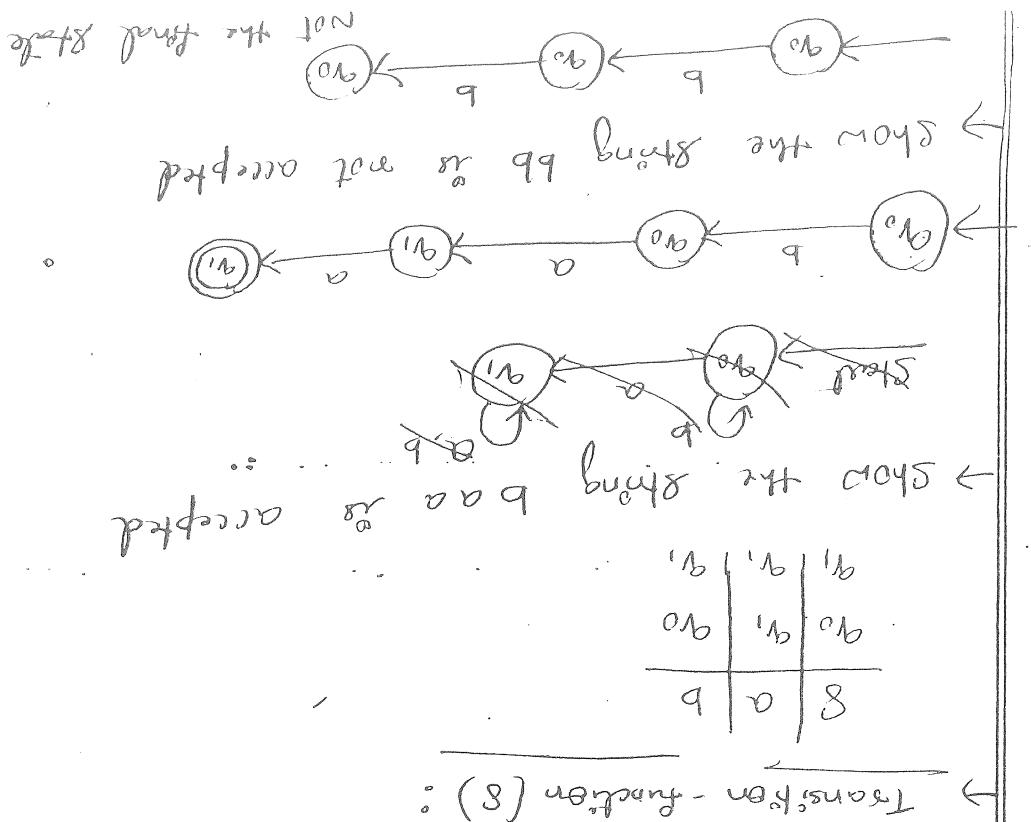
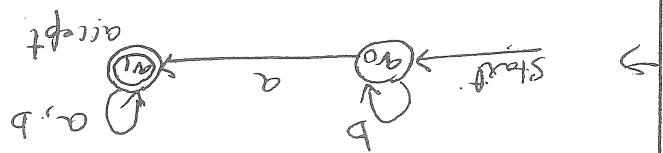


Fig. Transition diagram



To accept the char, two states are required.

Minimum string is a

In final state.

Whenever a string from L is input, it should land in final state.

Whenver a string from L is input, it should land in final state, if can be in any other state.

In final state.

Final state, it can be in any other state.

To accept the char, two states are required.

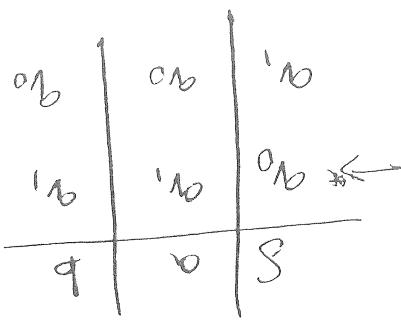
$$L = \{b, bb, bbb, \dots\}$$

The strings which are not accepted by the DFA is:

Sol: All strings in L should end any state (accepting).

$$L = \{w \in \{a, b\}^* \mid w \text{ contains at least one } a\}$$

Q) Write a PSM to accept L, where the



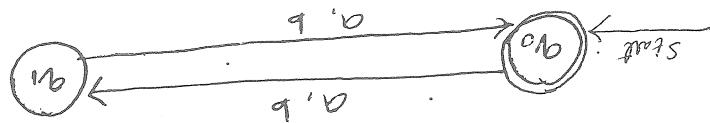
not found state.



Show string bba is not accepted



Show string bb is not accepted
Step 4 :- Transition function (δ)



Step 3 :- Write Transition diagram

So, two states are required.

Step 2 :- Min. strings are $\{e, aa\}$

$$L = \{a, b, aba, abb, aaa, bbb, \dots\}$$

Note:- C is even, because its length is 0

$$L = \{e, aa, ab, ba, bb, aaa, bbb, \dots\}$$

Step 1 :- Write the strings accepted by L

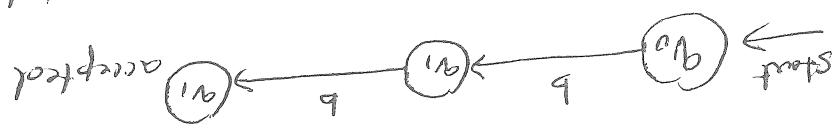
So:-

$$L = \{w \in \{a, b\}^* \mid w \text{ is even length}\}$$

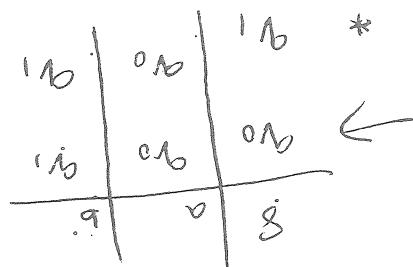
3) Write a DFA to accept the language.



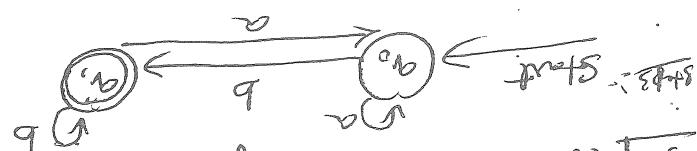
To show string ba is not accepted



Step 5 :- To show string bb is accepted



Step 6 :- Transition function (S)



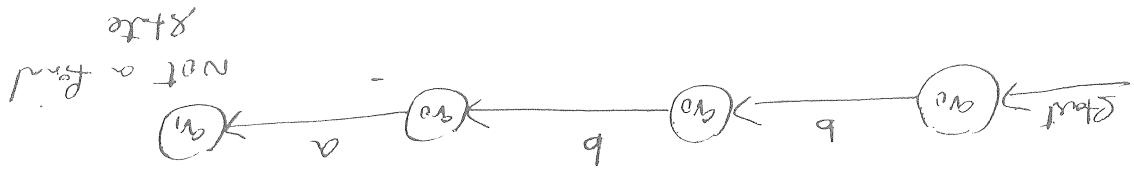
Step 7 :- Min. string is ba . Both states are reached.

$L = \{a, aa, baa, bba, \dots\}$

Sol :- Step 1 :- $L = \{b, ab, aba, abb, ba, \dots\}$

$L = \{w \in \{a, b\}^* \mid \text{every } w \text{ ends in } b\}$

(4) Write a DPM to accept the language.

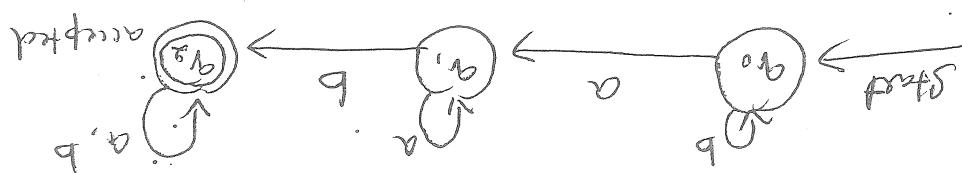


short strong bba is not accepted



~~84451~~ - shows shifting ab as reflected

Step 4: Transliteration



Step 3: Write Transliteration diagram:

5463 - New shell is fab'y, so 3 shells are required.

~L = { baa, a, b, aa, bb, bba; bbb, ... }

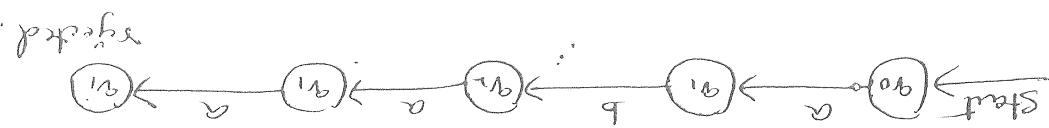
$$L = \{ab, aba^*, aab^*, abaaa^*, abbbb^*, bbaabb^*, \dots\}$$

Step 2 :- Write the changes accepted by

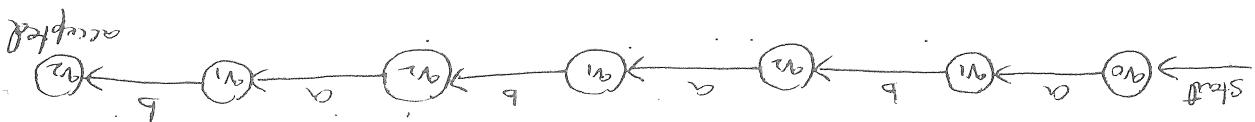
- १० -

$$L = \{w \in \{a, b\}^* \mid ab \text{ is a substring of } w\}$$

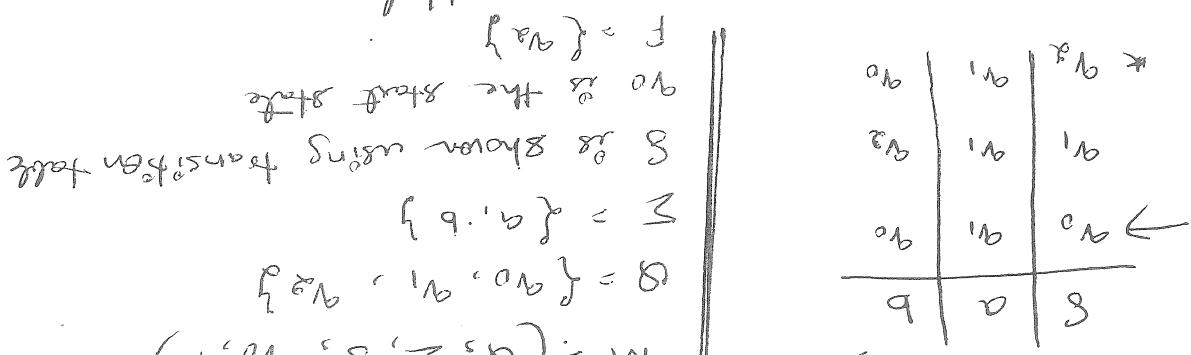
5) Write a DFSM to accept the language



To show string abaa is not accepted.



Steps :- To show string abab is accepted

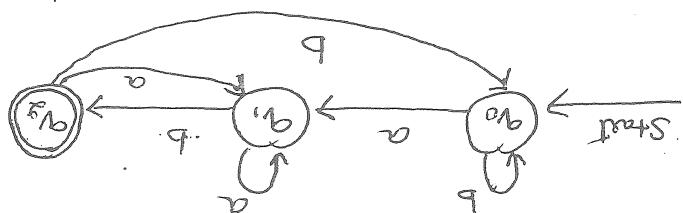


$$Q = \{q_0, q_1, q_2\}$$

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

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

Finally, DFA can be written as:



Step 3: Transition diagram

Step 2: So 3 states are required

Step 2:- Min string accepted is ab

$$L = \{aba, abb, aaa, bbb, a, b, \dots\}$$

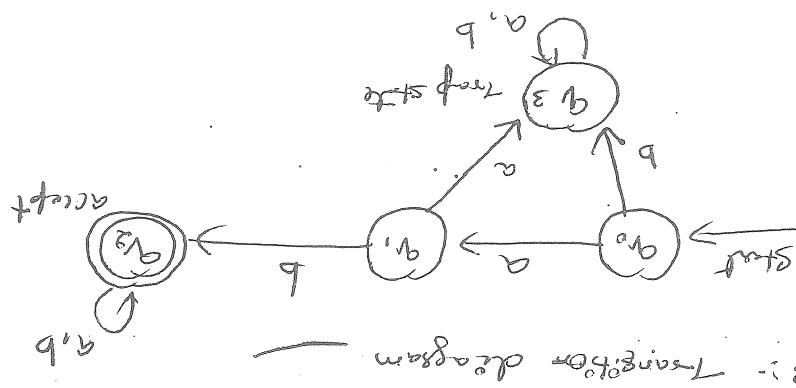
$$L = \{aab, ab, ba, bba, aab, \dots\}$$

- With the string ab and give the formal definition.
- 6) Design a DFA to accept strings of a's and b's ending

→ All external characters will be in trap state
 ← Only incoming edges and there are no outgoing edge.
 Note:- Trap state:

a_3	a_3	a_3
a_2	a_2	a_2
a_1	a_1	a_1
a_0	a_0	a_0
b	a	b

Step 4 :- Transition Table



Step 3 :- Transition diagram

So three states are required.

Step 2 :- Now string accepted by the machine is ab.

$L = \{aaa, aa, a, b, ba, bbb, bab, baa, \dots\}$

$\therefore L = \{abab, aba, abb, ababa, abaa, \dots\}$

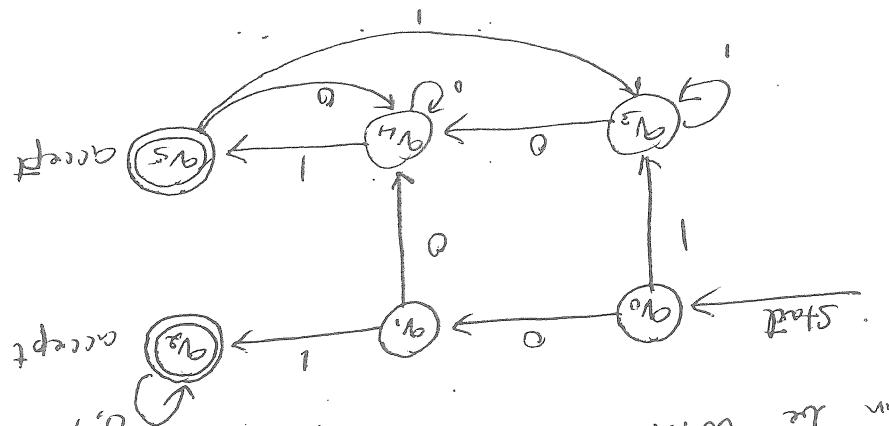
Step 1 :- $L = \{ab, abab, aba, abb, ababa, abaa, \dots\}$

Q1 :-

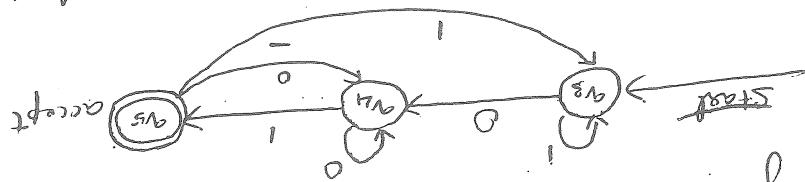
With the string ab.

∴ Design a DFA to accept strings of ab as substrings

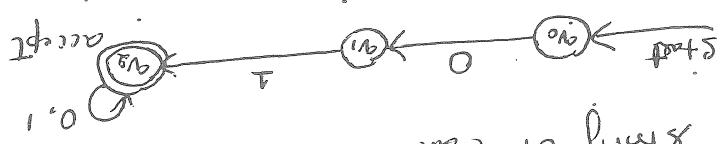
$S = \{q_0, q_1, q_2, q_3, q_4, q_5\}$	$\delta = \{(q_0, q_1, 0), (q_1, q_2, 1), (q_2, q_3, 0), (q_3, q_4, 1), (q_4, q_5, 0), (q_5, q_0, 1)\}$	q_0	q_1	q_2	q_3	q_4	q_5
S is shown using the transition table and transition diagram.							
Formally, DFA can be defined as:							
$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$							
δ = { $(q_0, 0, q_1)$, $(q_1, 1, q_2)$, $(q_2, 0, q_3)$, $(q_3, 1, q_4)$, $(q_4, 0, q_5)$, $(q_5, 1, q_0)$ }							
q_0 is the start state and q_5 is the final state.							



The above two DFAs can be joined to accept strings of 0s and 1s ending with 01 or both 0s and 1s beginning with 01, either as shown below:



The DFA to accept strings of 0s and 1s ending with 01 can be written as:

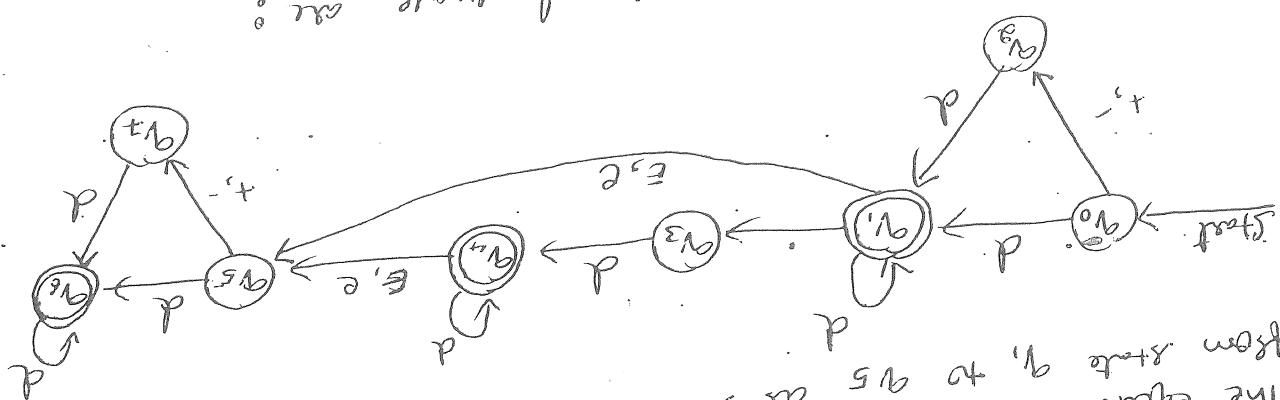


So, The DFA to accept strings of 0s and 1s starting with 01 ends as Both with the substring 01.

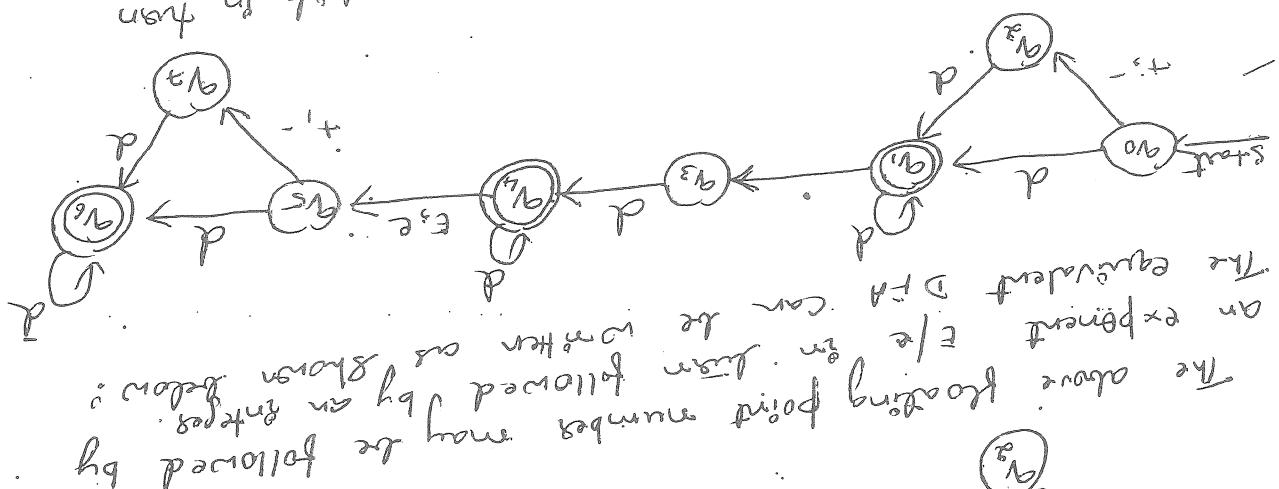
the alphabet $S = \{0, 1\}$ that either begins with 01 or ends with 01 can be written as:

8) Draw a DFA to accept all strings in

$L = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, \cdot, /\}$
 $L = \{3, 0, +3, 0, 33, 1, +33, 11, 0, 3E1, 0, 3E+1,$
 Note: string accepted by this language are:



But, an integer followed by E/e is also a floating point number.
 followed by an integer is also a floating point having an edge
 from state q_1 to q_5 as shown below:
 The equivalent DFA can be obtained as follows:



Solution:- An integer may be followed by a dot followed
 by an integer. The DFA can be written as shown below:
 option B is:

g) Draw a DFA to accept a floating point number with

- 1) Draw a DFA to accept strings with atleast one 'a' and one 'b'.
- 2) Draw a DFA to accept strings of 'a's and 'b's showing a pulse train - abab.
- 3) Design a DFA to accept strings of 'a's and 'b's showing no switching - aba.
- 4) Draw a DFA to accept strings of 'a's and 'b's ending with ab or ba.
- 5) Draw a DFA to accept strings of 'a's and 'b's showing three consecutive 'a's.
- 6) Draw a DFA to accept strings of 'a's showing no 3 consecutive 'a's.
- 7) Draw a DFA to accept strings of 'a's and 'b's showing atleast four 'a's.
- 8) Draw a DFA to accept strings of 'a's and 'b's showing not more than 3 'a's.
- 9) Draw a DFA to accept strings of 'a's and 'b's showing exactly three 'a's.
- 10) Draw a DFA to accept strings of 'a's and 'b's such that no two consecutive characters are same.
- 11) Draw a DFA to accept strings with atleast one 'a' and one 'b' showing a pulse train - ababa.

Pattern Recognition Problems :-

Assignment - I

- (12) Obtain a DFA to accept strings of 0's and 1's such that a's beginning with a 0, followed by odd number of 1's and ending with a 2.
- (13) Obtain a DFA to accept strings of 0's and 1's with almost two consecutive 0's.
- (14) Obtain a DFA to accept strings of 0's and 1's starting with atleast two 0's and ending with atleast two 1's.
- (15) Obtain a DFA to accept the language $L = \{wba^l w \in \{a, b\}^*\}$

Because it uses the ten digits from 0 to 9.
 For the decimal/denary system the radix is ten.
 To represent numbers in a positional numerical system.
 The number of unique digits including the digit zero would
 In digital numerical systems, the radix is base 10 .

The base of a system of numeration.

Note: What is radix?

Step 4: Convert the DFA using the transition diagram of DFA

Step 3: Find the transitions using $S(q_i; a) = q_j$ where
 $j = (q_i + a) \text{ mod } k$

Step 2: Compute the possible remainders. These remainder
 Step 1: Identify the radix, input alphabets and the digits k.

Figure of problem is shown below:

The step to be followed to find FA for this

where $j = (q_i + a) \text{ mod } k$

$S(q_i; a) = q_j$

where a is the radix of input. For binary $a = 2$

d represents digit. For binary $d = \{0, 1\}$

i is the remainder obtained after dividing by k .

k is the radix of DFA

where $j = (q_i + d) \text{ mod } k$

For these figures if problems, the transitions can be obtained using the following relation:

We are constructing DFA that divide a number by k .

Divisible by k problems:

Transitions of DFA				
$t_1 = (q_1, 1)$	$g = 4 \mod 5 = 4$			
$s_1 = (q_1, 0)$	$(q+4+0) \mod 5 = 3$	0		$q = 4$
$t_2 = (q_2, 1)$	$g = 5 \mod (1 + 3 * 8)$	1		
$s_2 = (q_2, 0)$	$(q+3+0) \mod 5 = 1$	0		$s = 1$
$t_3 = (q_3, 1)$	$g = 5 \mod (1 + 8 * 8)$	1		
$s_3 = (q_3, 0)$	$(q+8+0) \mod 5 = 0$	0		$p = 0$
$t_4 = (q_4, 1)$	$g = 5 \mod (1 + 8 * 8)$	1		
$s_4 = (q_4, 0)$	$(q+8+0) \mod 5 = 0$	0		$p = 0$
$t_5 = (q_5, 1)$	$g = 5 \mod (1 + 1 * 8)$	1		
$s_5 = (q_5, 0)$	$(q+1+0) \mod 5 = 0$	0		$l = 0$
$t_6 = (q_6, 1)$	$g = 5 \mod (1 + 0 * 8)$	1		
$s_6 = (q_6, 0)$	$(q+0+0) \mod 5 = 0$	0		$o = 0$

$$t_6 = (p+q+d) \mod 5$$

$$g = (q+1+0) \mod 5$$

$$g \mod 5 = (q+1+0) \mod 5 = 0$$

$$g = q + 1 + 0 = q + 1$$

$$g = q + 1 \text{ using the following reduction:}$$

$$g = q + 1 \text{ as } g = q + 1 \text{ with } q = 4$$

Step 3 :- Compute transitions : The transitions can be computed

the possible boundaries are ; $g = 0, 1, 2, 3, 4$

Step 3 :- Compute the possible boundaries : After dividing by 8

$$g = 0, 1, 2, 3, 4 \text{ In this case, } e = 2, d = 0, 1, 3, f = 5$$

Step 1 :- Identify the radix, input alphabet and the divisor k.

So :- The DFA can be obtained as shown below :

accept

Final example : 0000, 0101, 1010, 1111 should be

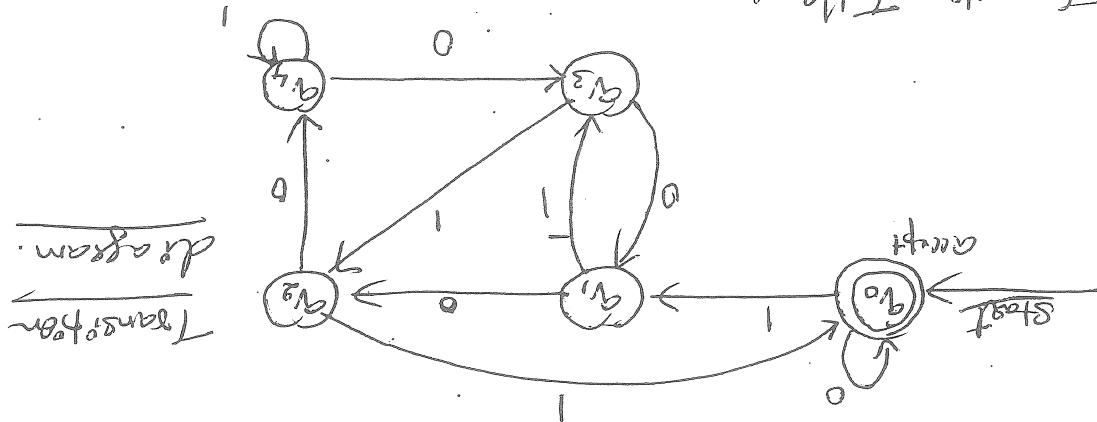
accepted. The should be accepted.

A binary number is only if the string representing some is where the value of each string is represented as a boundary number. So the following representation goes

3) Construct a DFA which accepts strings of 0's and

In general, for modulo k , number of states of DFA will be k .
 Note:- Observe that for modulo 5, number of states of DFA will be 5.

q_4	q_{14}
q_2	q_1
q_0	q_3
q_3	q_4
q_1	q_2
q_0	q_0^*
0	1
1	0
0	8



8 is shown below using the transition diagram and transition table.

$$F = \{q_0\}$$

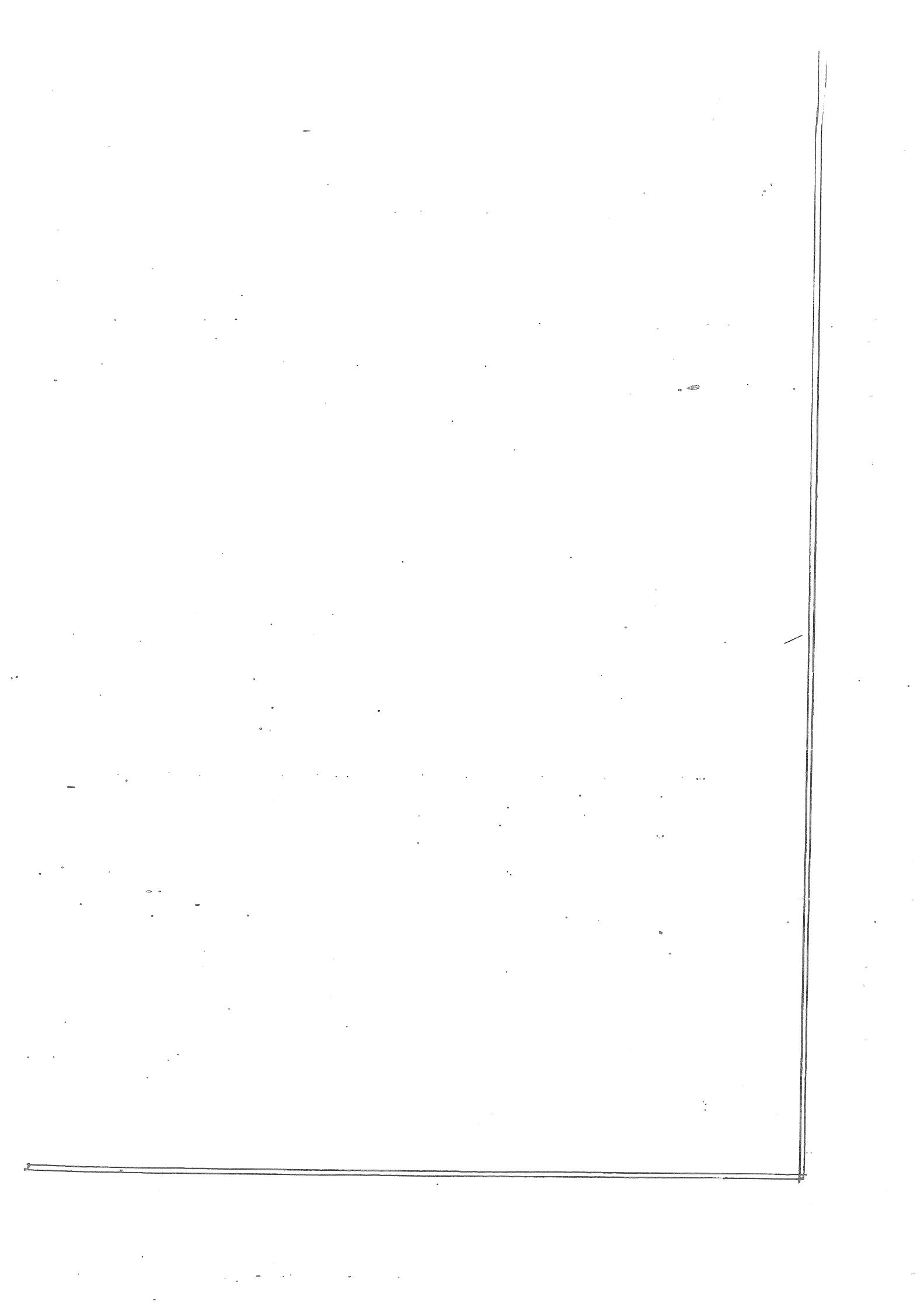
q_0 is the start state

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

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

where,

Step 4: The DFA can be defined as $M = (Q, Z, S, q_0, F)$



$\{8, 5, 8\}$ can be written as $\{8\}$
 $\{8, 4, 3\}$ can be written as $\{8\}$
 $\{0, 3, 6, 9\}$ can be written as $\{8\}$
 $\{8, 5, 8\}$ with a as the remainder
 $\{1, 4, 3\}$ with 1 as the remainder
 $\{0, 3, 6, 9\}$ with 0 as the remainder

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

$$d = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

Note:- Let us group the digits from 0 to 9 based on the remainder we get after dividing by 3 for the sake of convenience

$$s = (a_i + d) \bmod 3$$

$$s(a_i, a) = a_i \text{ where } s = (a_i + d) \bmod 3$$

using the following addition:

Step 3: Compute transitions: The transitions can be computed

using DFA

$s = 0, 1, 2$ which implies a_0, a_1, a_2 and a_3 are the

remainders:

decimal number by 3, results in following states
Step 2: Compute the possible remainders: After dividing any

$$k = 3$$

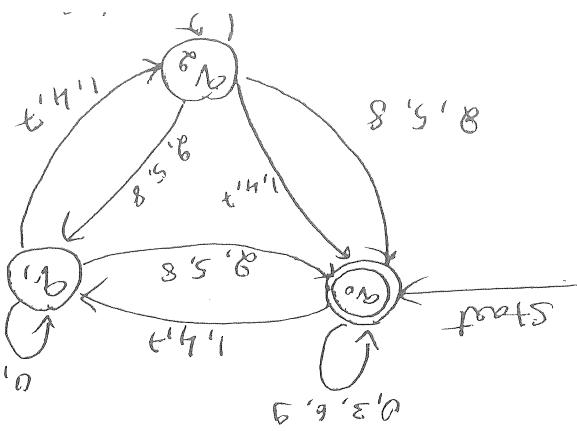
In this case, $a = 10$, $d = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$,

Step 1: Identify the states, input alphabet and the division k

Solution:- The DFA can be obtained as shown below:

Q) Draw a DFA to accept decimal strings divisible by 3

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_1
a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_1	a_2
a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_1	a_2	a_3



Q 8 shows using the transition diagram.

$$E = \{q_0\}$$

Q 9 the start state

$$Z = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

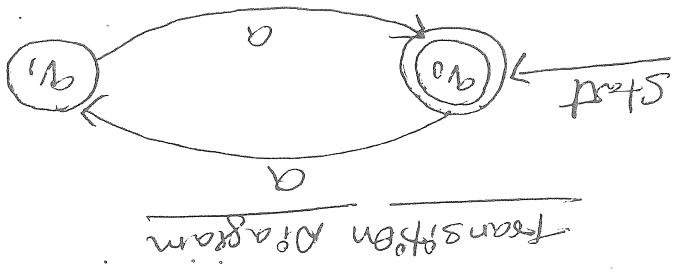
$$Q = \{q_0, q_1, q_2\}$$

$$M = (Q, Z, S, q_0, E) \text{ where}$$

Q 4.1 - The DFA can be defined as -

Remainder	$(10x + 2) \bmod 3 = 1$	$(10x + 1) \bmod 3 = 0$	$(10x + 0) \bmod 3 = 0$	$E = \{2\}$
$a_1 < \{8, 5, 8\} = q_0$	$g(q_0, \{8, 5, 8\}) = 1$	$g(q_0, \{1, 4, 7\}) = 0$	$g(q_0, \{0, 3, 6, 9\}) = 0$	$1 = ?$
$a_2 < \{1, 4, 7\} = q_1$	$g(q_1, \{8, 5, 8\}) = 0$	$g(q_1, \{1, 4, 7\}) = 1$	$g(q_1, \{0, 3, 6, 9\}) = 0$	$0 = ?$
$a_3 < \{0, 3, 6, 9\} = q_2$	$g(q_2, \{8, 5, 8\}) = 0$	$g(q_2, \{1, 4, 7\}) = 1$	$g(q_2, \{0, 3, 6, 9\}) = 1$	$1 = ?$
\vdots	\vdots	\vdots	\vdots	$0 = ?$

String length mod k problems :



Transitions Diagram

$$\begin{array}{c} q_1 = (s \text{ mod } 1+1)_k = (s + a) \text{ mod } k \\ q_0 = (s \text{ mod } 1+0)_k = (s - a) \text{ mod } k \\ \hline q_0 = (s \text{ mod } 1+1)_k = (s + a) \text{ mod } k \end{array}$$

$$S(q_1, a) = q_1^{(e+1) \bmod k}$$

The transitions can be observed as shown below:

$$q = 0, 1$$

$$a = 0, 1, 2, \dots$$

The possible remainders are 0, 1, 0, 1, 0, 1, 0, 1, ...

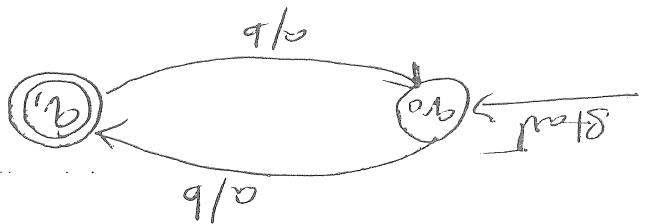
To as final state

$$L = \{ w : |w| \bmod a = 0 \} \text{ with } q_0 \text{ as the start state and } q_1 \text{ as final state}$$

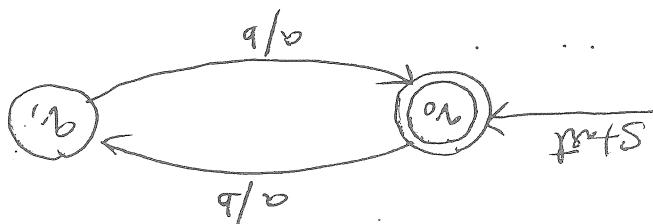
shown below:

So, the language accepted by the DFA can be written as

obtain a DFA to accept strings of even numbers of a's.



Note:- By changing the final state to q_1 and non-final state to q_0 , we get strings showing odd number of symbols and non-final state to final state, we get strings showing odd number of symbols.



$$\begin{array}{c|c}
 S(a_1, a/b) = a_1 & ! \\
 \hline
 S(a_0, a/b) = a_0 & 0 \\
 \hline
 S(a_1, a/b) = a_1 & ? \\
 \hline
 \text{Transitions :-} &
 \end{array}$$

$$i = 0, 1$$

$$k = 2$$

as shown below:
 Q1:- The language accepted by the DFA can be written showing even numbers of symbols.
 2) Obtain a DFA to accept strings of a^* and b^* and a^* and b^* as final state with $L = \{ w : |w| \bmod 2 = 0 \}$ as the start state and q_0 as final state.

α_1	α_2	α_3
α_2	α_0	α_1
α_3	α_1	α_0
α_0	α_1	α_2
α_1	α_2	α_3

8 is shown using the
earliest form of
earliest written
form of the
earliest written

$$\{ab\} = +$$

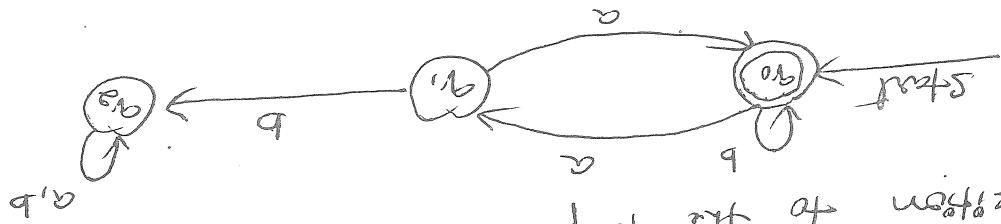
$g_0 = \text{start_site}$

$$\{q^{\prime }v\}=\emptyset$$

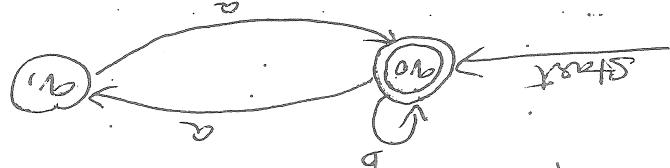
$$\{ \alpha_0, \alpha_1 \} = \alpha$$

$$M = (\alpha, \beta)$$

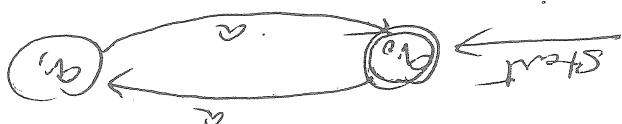
Generally, the above DFA can be defined as
 $M = (Q, \Sigma, S, q_0, F)$ where



Since there is no definition of $\lim_{n \rightarrow \infty} s_n$, there is a
concept of subsequences that define the limit of a sequence.
In simple words, if all subsequences of a sequence converge to the same value, then the sequence converges to that value.



Immediatly after accepting even numbers of 6's we can accept any number of 6's.



Q7 :- The language accepted by the DFA can have even numbers of 0's as shown below :-

• *Highway* was first in my suggestion.

3) Definition of DFA to accept $L = \{w \in \{a, b\}^*: \text{every } i$

q_0	q_1	q_2
q_1	q_0	q_0^*
1	0	3

Transition table

is shown using transition table

$$S(q_3) = \{$$

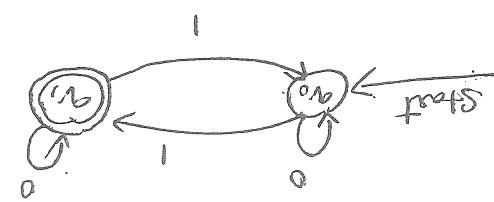
start state

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

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

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

Formal definition of DFA



Transition diagram

$$S(q_0, a) = q_0 \quad S(q_1, a) = q_1 \quad S(q_2, a) = q_2$$

$$S(q_0, 1) = q_1 \quad S(q_1, 1) = q_0 \quad S(q_2, 1) = q_2$$

$$S(q_0, 0) = q_0 \quad S(q_1, 0) = q_0 \quad S(q_2, 0) = q_2$$

$$S(q_0, a) = q((i+1) \bmod k) \text{ where } k=2 \text{ and } i=0, 1$$

The transitions can be defined using the following relation:

$L = \{w : |w| \bmod 2 = 1\}$ with q_0 as the start state and q_2^* as the final state.

The language accepted by the DFA can be written as -

as there

So: odd parity indicates that odd numbers 8 is should

4) Given a DFA to accept $L = \{w \in \{0, 1\}^* : w \text{ has odd parity}\}$

$$L = \{a^m : m \geq 0\}$$

∴

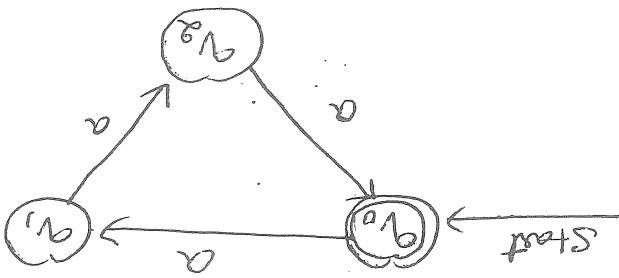
$$\{a^m : m \geq 0\} \text{ are accepted by } S \text{ if } m \equiv 0 \pmod{3}$$

∴

$$f(a) = \{a^m : m \equiv 0 \pmod{3}\} = \{a^0\} = \{1\}$$

∴ when $m \equiv 0 \pmod{3}$

The language accepted by DFA can also



$$q_0 \in \{a^m : m \equiv 0 \pmod{3}\} = \{a^0\} \quad | \quad a$$

$$q_1 \in \{a^m : m \equiv 0 \pmod{3}\} = \{a^1\} \quad | \quad 1$$

$$q_2 \in \{a^m : m \equiv 0 \pmod{3}\} = \{a^2\} \quad | \quad 0$$

$$\frac{\{a^m : m \equiv 0 \pmod{3}\}}{\{a^m : m \equiv 1 \pmod{3}\}} = \{a^m : m \equiv 2 \pmod{3}\}$$

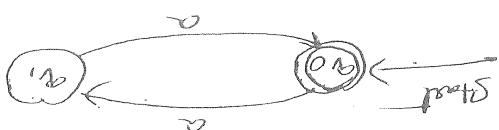
Transitions

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

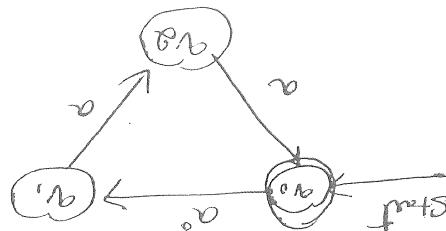
∴ $L = \{a^m : m \equiv 0 \pmod{3}\} \cup \{a^m : m \equiv 2 \pmod{3}\}$ is accepted by DFA

∴ $L = \{a^m : m \equiv 0 \pmod{3}\} \cup \{a^m : m \equiv 2 \pmod{3}\}$

∴ DFA with $\Sigma = \{a\}$ to accept $L = \{a^m : m \equiv 0 \pmod{3}\} \cup \{a^m : m \equiv 2 \pmod{3}\}$



$$\underbrace{M_1 \bmod 2}_{W_1}$$



$$\underbrace{M_2 \bmod 3}_{W_2}$$

Q1 :- we need to construct two machines M_1 and M_2

Q2 :- we have both $|w| \bmod 2$ and $|w| \bmod 3$, and $|w| \bmod 2$, $|w| \bmod 3$ and M_1 and M_2 .

$w \in L$ and $L = \{a\}$

Q3 :- DFA to accept a string w satisfying the following condition :-

Step 1 :- Using the relation $s_i \rightarrow s_j$ if the final state of s_i is self-looping after the transition $s_i \rightarrow s_j$ and $s_j \rightarrow s_i$ and M_1 and M_2

$$S(\{p, q\}, a) = (S_1(p, a), S_2(q, a))$$

self-looping.

Step 2 :- Obtain the transitions of S by combining M_1 and M_2 using self-looping.

where q_1 and q_2 are the states of machine M_1 and M_2

$$Q = Q_1 \times Q_2$$

product of Q_1 and Q_2

accept a given string w if it is accepted by both M_1 and M_2

Step 3 :- Now, the states of Q of combined machine M will be obtained by taking the cross product of the states of M_1 and M_2 .

a self-looped state of Q is the combination of self-looped states of M_1 and M_2 .

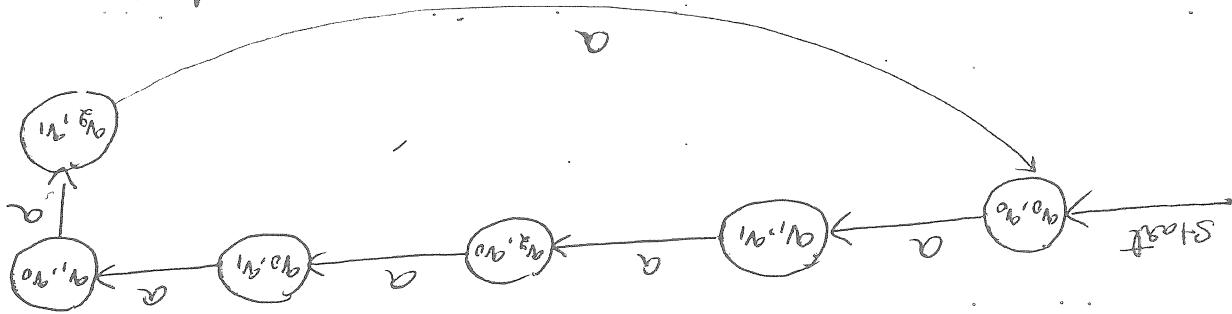
self-looped state :-

general method to solve string length mod k problems combined by string length mod k problems combined with string length mod k problems with string length mod k problems :-

Observe that those are machines: whose length is
 divided by 3 i.e., 1 mod 3
 \rightarrow The machine M_1 accepts a string w where length is
 divided by 3 i.e., 1 mod 3
 \rightarrow The machine M_2 accepts w where length is
 divided by 2 i.e., 1 mod 2
 \rightarrow The machine M_3 accepts w where length is
 combined machine $M = M_1 \times M_2$
 Now, the states A & B combined machine M that accept
 a given string w can be obtained by taking
 product of A_1 and A_2 as shown below:
 $A = A_1 \times A_2$
 $= \{ (q_0, q_0), (q_1, q_1), (q_2, q_2) \}$
 where state (q_0, q_0) is the start state
 of the pair (x, y) can be
 obtained as shown below:
 $S((x, y), a) = (S_1(x, a), S_2(y, a))$

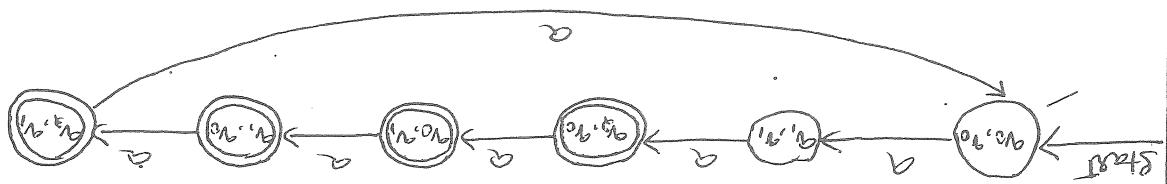
Transitions of M_1		Transitions of M_2	
a_0	a_1	a_0	a_1
a_1	a_0	a_1	a_0
a_2	a	a_2	a_1

$\{(\alpha_0, \alpha_0), (\alpha_1, \alpha_0), (\alpha_0, \alpha_1), (\alpha_1, \alpha_1)\} = T$
 state. So, in the above DFA, the final state are
 $\{(\alpha_0, \alpha_0), (\alpha_1, \alpha_0), (\alpha_0, \alpha_1)\}$ such that $x \leq y$ are final
 To accept strings y in such that $|y| \leq |x|$



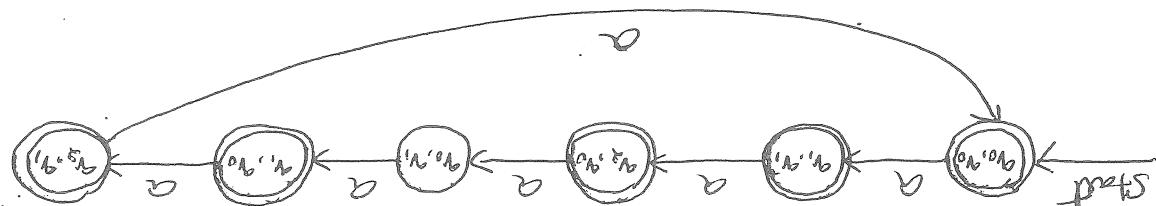
$S((\alpha_0, \alpha_0), a)$	$S((\alpha_1, \alpha_0), a)$	$S((\alpha_0, \alpha_1), a)$	$S((\alpha_1, \alpha_1), a)$
$S_1(\alpha_0, a)$	$S_2(\alpha_0, a)$	$S_3(\alpha_0, a)$	$S_4(\alpha_0, a)$
$S_1(\alpha_1, a)$	$S_2(\alpha_1, a)$	$S_3(\alpha_1, a)$	$S_4(\alpha_1, a)$
$S_1(\alpha_0, \alpha_0)$	$S_2(\alpha_0, \alpha_0)$	$S_3(\alpha_0, \alpha_0)$	$S_4(\alpha_0, \alpha_0)$
$S_1(\alpha_1, \alpha_0)$	$S_2(\alpha_1, \alpha_0)$	$S_3(\alpha_1, \alpha_0)$	$S_4(\alpha_1, \alpha_0)$
$S_1(\alpha_0, \alpha_1)$	$S_2(\alpha_0, \alpha_1)$	$S_3(\alpha_0, \alpha_1)$	$S_4(\alpha_0, \alpha_1)$
$S_1(\alpha_1, \alpha_1)$	$S_2(\alpha_1, \alpha_1)$	$S_3(\alpha_1, \alpha_1)$	$S_4(\alpha_1, \alpha_1)$
$S_1(\alpha_0, \alpha_0, a)$	$S_2(\alpha_0, \alpha_0, a)$	$S_3(\alpha_0, \alpha_0, a)$	$S_4(\alpha_0, \alpha_0, a)$
$S_1(\alpha_1, \alpha_0, a)$	$S_2(\alpha_1, \alpha_0, a)$	$S_3(\alpha_1, \alpha_0, a)$	$S_4(\alpha_1, \alpha_0, a)$
$S_1(\alpha_0, \alpha_1, a)$	$S_2(\alpha_0, \alpha_1, a)$	$S_3(\alpha_0, \alpha_1, a)$	$S_4(\alpha_0, \alpha_1, a)$
$S_1(\alpha_1, \alpha_1, a)$	$S_2(\alpha_1, \alpha_1, a)$	$S_3(\alpha_1, \alpha_1, a)$	$S_4(\alpha_1, \alpha_1, a)$
$S_1(\alpha_0, \alpha_0, \alpha_0)$	$S_2(\alpha_0, \alpha_0, \alpha_0)$	$S_3(\alpha_0, \alpha_0, \alpha_0)$	$S_4(\alpha_0, \alpha_0, \alpha_0)$
$S_1(\alpha_1, \alpha_0, \alpha_0)$	$S_2(\alpha_1, \alpha_0, \alpha_0)$	$S_3(\alpha_1, \alpha_0, \alpha_0)$	$S_4(\alpha_1, \alpha_0, \alpha_0)$
$S_1(\alpha_0, \alpha_1, \alpha_0)$	$S_2(\alpha_0, \alpha_1, \alpha_0)$	$S_3(\alpha_0, \alpha_1, \alpha_0)$	$S_4(\alpha_0, \alpha_1, \alpha_0)$
$S_1(\alpha_1, \alpha_1, \alpha_0)$	$S_2(\alpha_1, \alpha_1, \alpha_0)$	$S_3(\alpha_1, \alpha_1, \alpha_0)$	$S_4(\alpha_1, \alpha_1, \alpha_0)$
$S_1(\alpha_0, \alpha_0, \alpha_1)$	$S_2(\alpha_0, \alpha_0, \alpha_1)$	$S_3(\alpha_0, \alpha_0, \alpha_1)$	$S_4(\alpha_0, \alpha_0, \alpha_1)$
$S_1(\alpha_1, \alpha_0, \alpha_1)$	$S_2(\alpha_1, \alpha_0, \alpha_1)$	$S_3(\alpha_1, \alpha_0, \alpha_1)$	$S_4(\alpha_1, \alpha_0, \alpha_1)$
$S_1(\alpha_0, \alpha_1, \alpha_1)$	$S_2(\alpha_0, \alpha_1, \alpha_1)$	$S_3(\alpha_0, \alpha_1, \alpha_1)$	$S_4(\alpha_0, \alpha_1, \alpha_1)$
$S_1(\alpha_1, \alpha_1, \alpha_1)$	$S_2(\alpha_1, \alpha_1, \alpha_1)$	$S_3(\alpha_1, \alpha_1, \alpha_1)$	$S_4(\alpha_1, \alpha_1, \alpha_1)$

- $\frac{1}{2} \in \mathbb{Z}$
- (3) mod 2 (3) mod 3
- b) $|w| \bmod 3 \neq |v| \bmod 2$ where $w \in \Sigma$ and $\Sigma = \{a, b\}$
- c) $|w| \bmod 3 \leq |v| \bmod 2$ where $w \in \Sigma$ and $\Sigma = \{a, b\}$
- d) Obtain a DFA to accept the foll. language
- Assignment :-



$f = \{(q_3, q_0), (q_0, q_1), (q_1, q_2), (q_2, q_3)\}$

Note:- To accept strings y we need that $|y| \bmod 3 \neq |x| \bmod 2$, the above DFA can be modified as shown below, with the following final states -



Q6, the above DFA to accept the given language can be modified as shown below:

Step 1: Find the remainders obtained by dividing the number of specified symbols by k .

General method to solve $N(a) \bmod k$ problems are:

Number of specified symbols $\bmod k$ problems:

Step 2: Divide the remainders obtained by the following condition:

$$g(a^k, a) = a^{(k+1) \bmod k}$$

Obtained using the following relation:

Step 3: Divide by the transpositions. The transpositions can be

number of specified symbols by k .

Step 4: Divide by the transpositions obtained by the start state and final state.

Step 5: Write the DFA. The DFA can be written using the transitions given in steps with start state and final state as obtained in step 3.

Step 6: Identify the start state and final state:

Where k is the divisor.

$\therefore g(a^k, a) = a^{(k+1) \bmod k}$

Obtained using the following relation:

Step 7: Identify the start state and final state.

Suppose the language to be accepted is ...

No. of strings the start state is ...

As the start state is the final state.

Step 8: If $a^k \equiv 1 \pmod{N(a)}$ then the string is accepted.

Step 9: If $a^k \not\equiv 1 \pmod{N(a)}$ then the string is rejected.

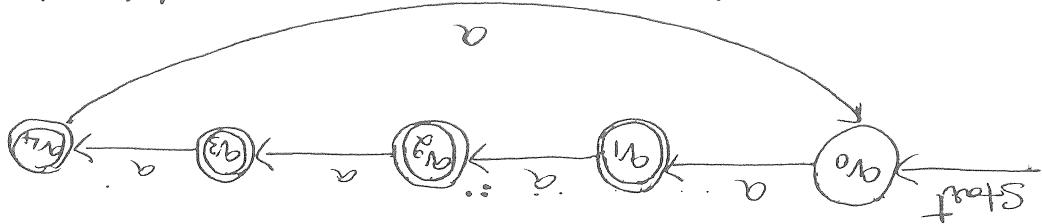
$a_0 = \text{start state}$

S is shown in transition diagram and function table

$$Z = Q_2$$

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

The DFA $M = (Q, Z, S, q_0, f)$ can be defined as:



$$0: S(q_0, a) = q_1 \quad a: S(q_1, a) = q_2 \quad 4: S(q_4, a) = q_4 \quad 3: S(q_3, a) = q_2$$

$$2: S(q_2, a) = q_3 \quad a: S(q_3, a) = q_4 \quad 1: S(q_1, a) = q_0 \quad 0: S(q_0, a) = q_1$$

$$2: S(q_2, a) = q_3 \quad a: S(q_3, a) = q_4 \quad 1: S(q_1, a) = q_0 \quad 0: S(q_0, a) = q_1$$

$$2: S(q_2, a) = q_3 \quad a: S(q_3, a) = q_4 \quad 1: S(q_1, a) = q_0 \quad 0: S(q_0, a) = q_1$$

$$2: S(q_2, a) = q_3 \quad a: S(q_3, a) = q_4 \quad 1: S(q_1, a) = q_0 \quad 0: S(q_0, a) = q_1$$

$$2: S(q_2, a) = q_3 \quad a: S(q_3, a) = q_4 \quad 1: S(q_1, a) = q_0 \quad 0: S(q_0, a) = q_1$$

$$2: S(q_2, a) = q_3 \quad a: S(q_3, a) = q_4 \quad 1: S(q_1, a) = q_0 \quad 0: S(q_0, a) = q_1$$

The transitions can be obtained as shown below:

i.e., q_1, q_2, q_3 and q_4 are final states

all are final states.

But, it is $N_a(w) \bmod 5 \neq 0$ and hence accept q_0 .

Note: If $N_a(w) \bmod 5 = 0$, then q_0 is the final state

$$Z = 0, 1, 2, 3, 4$$

$$X = 5$$

$$q_0 = \text{start state}$$

Remainder	0	1	2	3	4	0	1	2	3	4	0
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ...	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑

Goal

① Draw a DFA to accept language: $L = \{N_a(w) \bmod 5 \neq 0\}$

Result of transition for input symbol a

$A = \{(A_0, B_0), (A_0, B_1), (A_1, B_0), (A_1, B_1), (A, B)\}$

Symbol b
Symbol a
Transition for input a
Transition for input b

Given string w can be obtained by taking the cross product of A_1 and B_1 as

The state of combined machine is that accept a



$$N^b(w) \bmod q = 0$$

$$N^a(w) \bmod q = 0$$

$$(N^b(w) \bmod q = 0)$$

We need to construct a machine M_1 and M_2 one with respect to $N^a(w) \bmod q = 0$ and another with respect to $N^b(w) \bmod q = 0$.

$$L = \{N^a(w) \bmod q = 0 \text{ and } N^b(w) \bmod q = 0\}$$

Goal:- The given language can be written as :

even numbers of a's and even numbers of b's.

2) Obtain a DFA to accept strings of a's and b's having

(length)

Number of specified symbol mod k problems (with logcat)

$L = \{ w \mid Na(w) \bmod 3 = 0 \text{ and } Nb(w) \bmod 2 = 0 \}$

(8)

$L = \{ w : \text{Both } Na(w) \text{ and } Nb(w) \text{ are multiples of 2} \}$

(8)

$L = \{ w : \text{Both } Na(w) \text{ and } Nb(w) \text{ are divisible by 3} \}$

(8)

8 b3

$L = \{ w : w \text{ has even number of } a's \text{ and even number of } b's \}$

Note:- The language accepted by above DFA can be

$$L = \{ a_0 \}$$

$$a_0 = \text{start state}$$

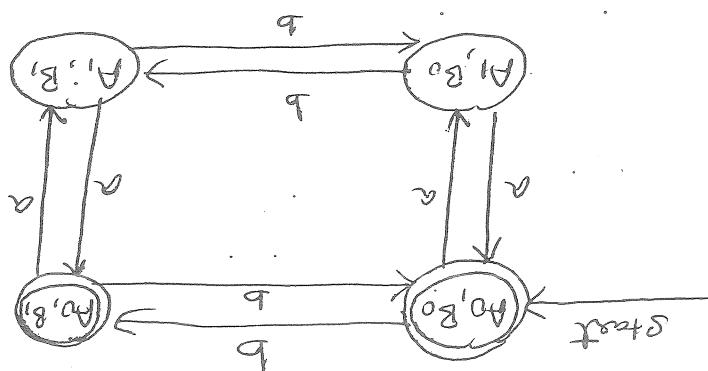
as shown in following diagram

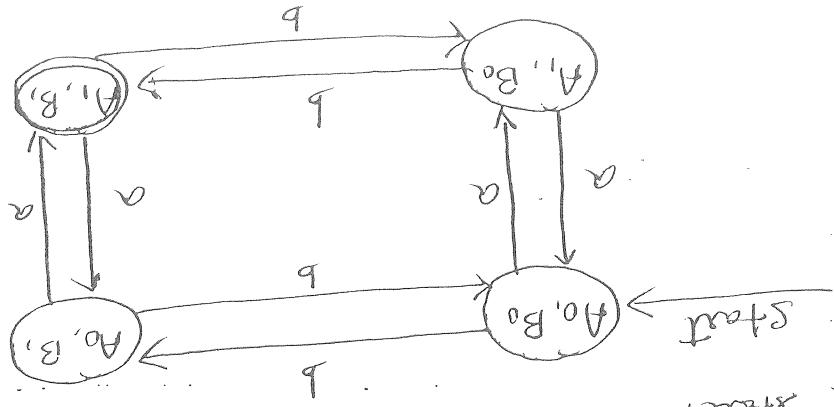
$$L = \{ a, b \}$$

$$L = \{ a_0, a_1, a_2, a_3 \}$$

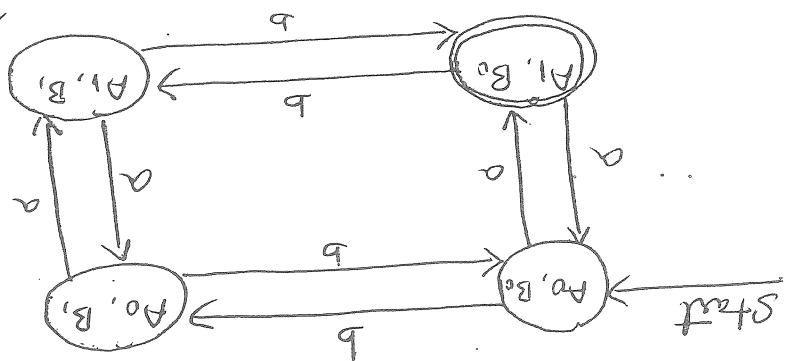
so, the DFA is $M = (Q, \Sigma, \delta, q_0, F)$

symbol a , so having the next cell transition are
symbol b , so having the previous cell transition are
The highlighted transitions are written for input

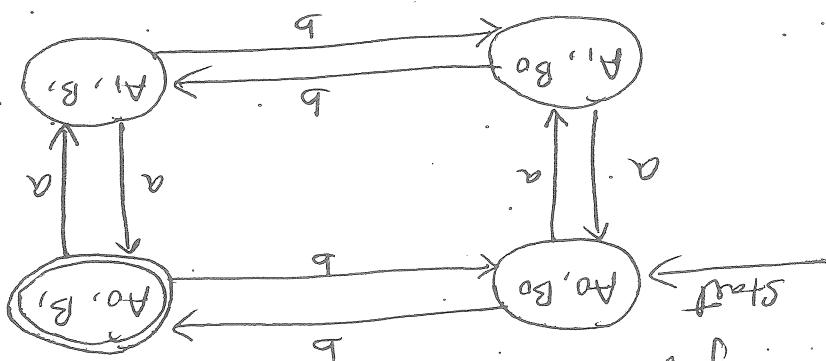




Note 3: The DFA to accept odd number of a 's and odd number of b 's can be obtained by making (A_1, B_1) as the final state.



Note 4: The DFA to accept odd number of a 's and even number of b 's can be obtained by making A_1, B_0 as the final state.



Note 5: The DFA to accept even numbers of a 's and odd numbers of b 's can be obtained by making A_0, B_1 as the final state.

Disadvantages of DFA :

- 1) Constructing some of the DFAs is very difficult.
- 2) The DFA cannot guess about the input.
- 3) The DFA is not very powerful.
- 4) At any point of time, the DFA is in only one state.
So, a DFA does not have the power to be in two states at once.

Note:- This is a difficult problem, we end up in spending lot of time to find the solution.

Step 3:- Some min strings are able to guess no. of states.

for ex. $L = \{a, aa, bb, aab, bba, aba, aabb, aaaa, bbbb, aabb, aaba, abaa, bbaa, baab, aabb, aaaa\}$

Write a DFSM to accept this language

$L = \{w \in \{a, b\}^* \mid 3rd character from right is a\}$

Note:- This is a difficult problem, we end up in spending lot of time to find the solution.

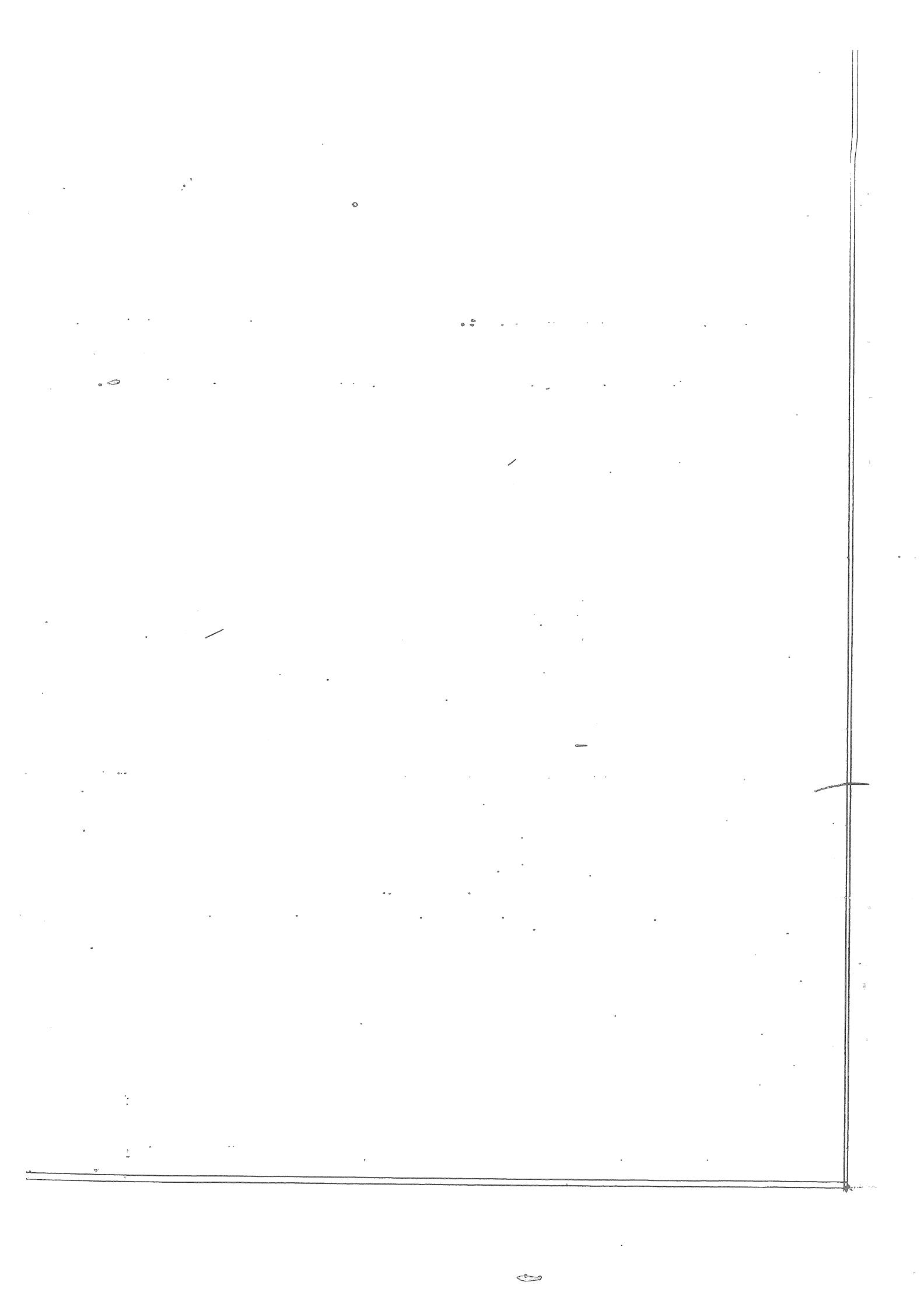
Step 3:- Some min strings are able to guess no. of states.

for ex. $L = \{a, aa, bb, aab, bba, aba, aabb, aaaa, bbbb, aabb, aaba, abaa, bbaa, baab, aabb, aaaa\}$

Write a DFSM to accept this language

$L = \{w \in \{a, b\}^* \mid every w ends in ab or ba\}$

Difference between both DFSM's:



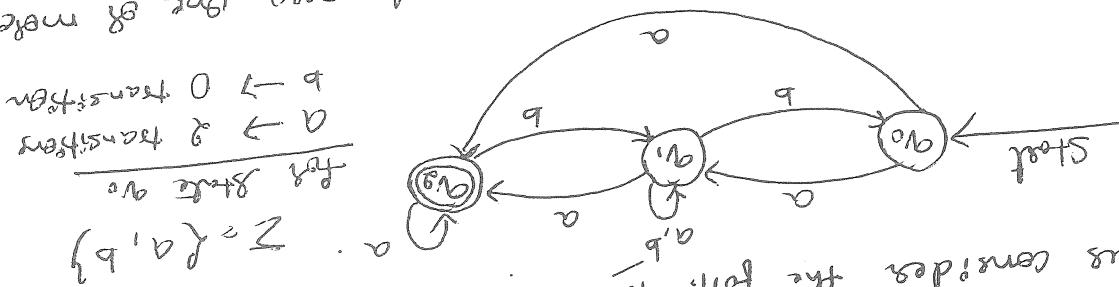
Computers are completely deterministic machines.

The state of the computer can be predicted from the input and initial state. We cannot find a computer which is "why non-deterministic finite automaton?". All the non-deterministic finite automaton has the ability to guess something about the input.

A "non-deterministic" finite automaton has very easy advantage. The various advantages of NFA are:

- Why non-deterministic finite automaton? All the non-deterministic finite automaton has the ability to guess something about the input.
- Using NFA. The various advantages of NFA are:
 - Constituting most of the NFAs is very easy.
 - If it is more powerful than DFA.
 - If it has the power to do in parallel rather of one.
 - If the NFA is an efficient mechanism to do some computation.
 - Simple and language centred.

Let us consider the following transition diagram:



In the FA shown above, there can be three cases, one of which transmits an input symbol. Such machines are called non-deterministic finite automata. Finite state machine of non-deterministic type is a non-deterministic function with alphabet as input and output.

What is a non-deterministic finite automaton?

Where $M = (Q, \Sigma, S, q_0, F)$

$Q \leftarrow$ non empty finite set of states

$\Sigma \leftarrow$ non empty finite set of symbols

$q_0 \in Q$ is the initial state

$F \subseteq Q$ is the set of final states

A mapping from $Q \times \Sigma$ to 2^Q .

$S \subseteq 2^Q$ is the transition function which is a map from $Q \times \Sigma$ to 2^Q .

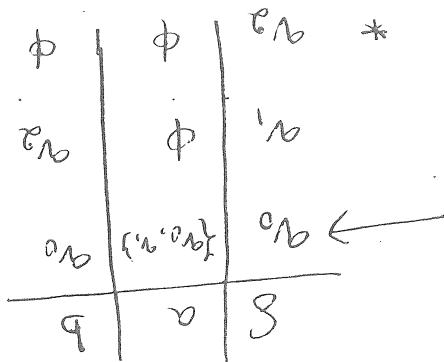
What is a non-deterministic finite function?

A finite automaton.

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

- followed by aa \Rightarrow two as make b's
 $L = \{w \in (a,b)^* \mid w \text{ is made up of } b \text{ in alphabetical order}\}$
 1) write a NFA to recognize the language

Acc: grammar :-



Q is shown using the transition table :-

$$T = \{q_0\}$$

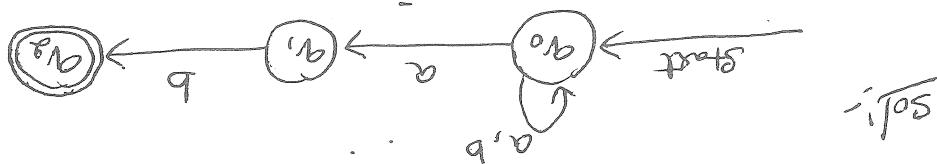
q_0 is start state

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

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

$$M = (Q, E, S, q_0, F)$$

NFA is formally defined as :-



Sol:-

2) Design an NFA to accept strings of a's and b's ending with ab.

Problems :-

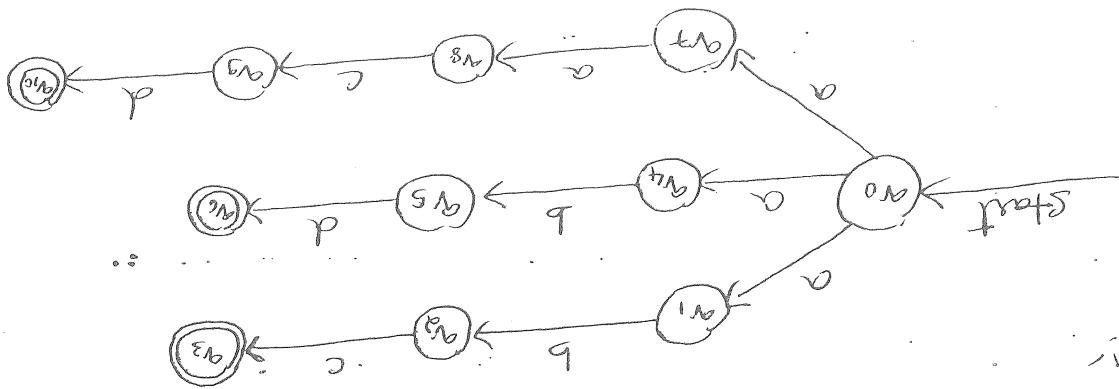
$a \in Q$ is the 3rd final state

$q_0 \in Q$ is the start state

machine enters into one of nine states

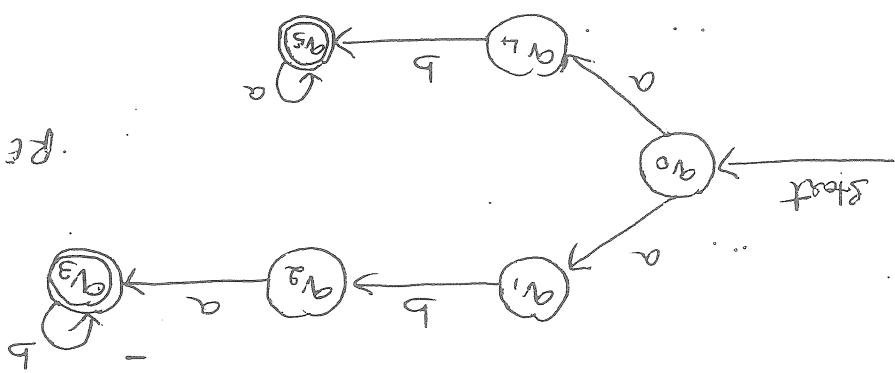
Based on the current state and input symbol, the

$$Q_E = abc + abd + acd$$

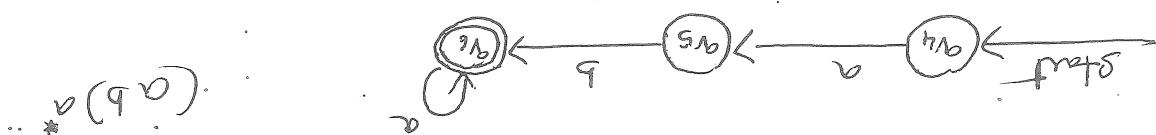


3) Design an NFA to recognize the following keywords
 $abc, abd \text{ and } acd$

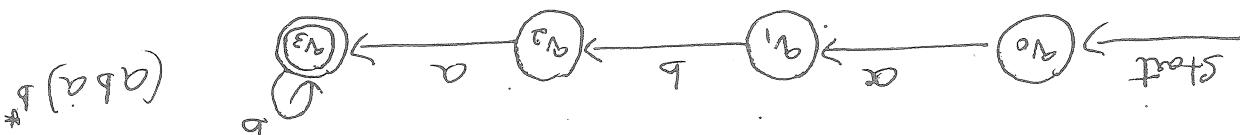
$$Q_E = (aba)^b + (ab)^a$$



The machine to accept either aba^b or ab^a where $n \geq 0$:



The machine to accept ab^a where $n \geq 0$:

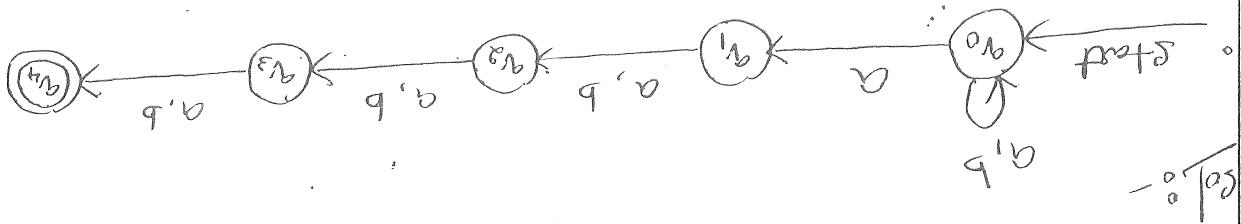


Goal:- The machine to accept $abab^n$ where $n \geq 0$:

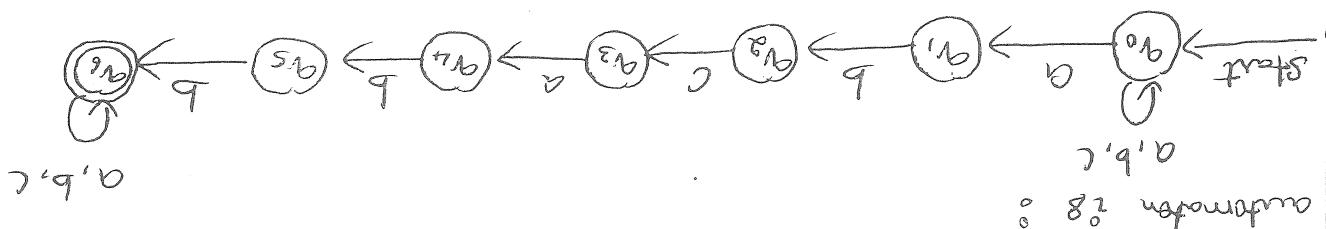
$$L = \{w \mid w \in abab^n \text{ for } abab \text{ where } n \geq 0\}$$

a) Obtain an NFA to accept the language

Note:- For every problem, write the transition table and define the machine formally. This is complicated.

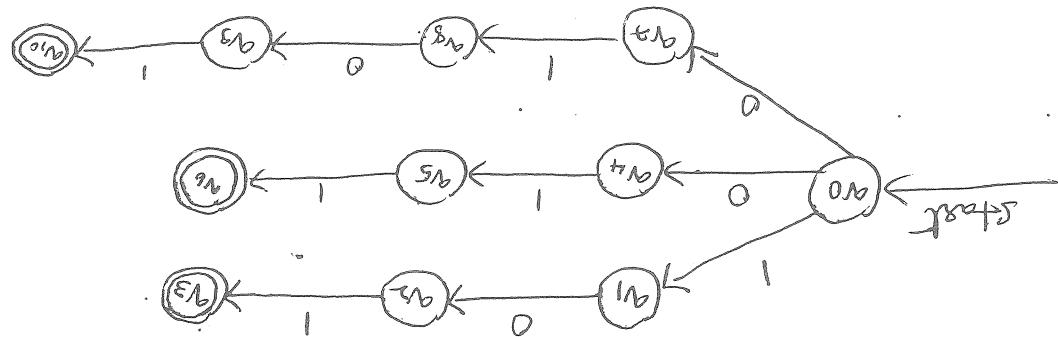


Q6) Design an NFA to accept strings of a^* s and b^* s such that fourth symbol from the right side is a .



Q7) The min. string that can be accepted by NFA is abcabb and the max. sequence to accept the string and the four symbols are a^* , b^* , c^* and $w = a^*b^*c^*a^*b^*b^*c^*$.

Q8) Design an NFA to accept $L = \{w \in \{a, b, c\}^* \mid x \text{ and } y \in \{a, b, c\}^* \text{ and } w = xabcabbay\}$



Q9) Design an NFA to recognize the following set of strings 101, 011 and 010

Q10) Design an NFA to recognize the following set of strings 101, 011, 010 and 0110

not already in S^D

$\Rightarrow \text{Add the state } \{q_1, q_2, \dots, q_n\} \text{ to } S^D, \text{ if } \{q_1, q_2, \dots, q_n\} \in \{q_1, q_2, \dots, q_n\}$

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

$S^D(q_k, a)$

$S^D(\{q_1, q_2, \dots, q_n\}, a) = S_n(q_k, a) \cup S^D(q_k, a) \cup \dots$

as shown below:

Step 3: Identify the transitions S^D of DFA.

for each state $\{q_1, q_2, \dots, q_n\}$ in S^D and for each input symbol a in Σ , the transition can be obtained

Step 2: Identify the alphabet of DFA.

The input alphabet of NFA are the output alphabet of DFA.

Step 1: Identify the start state of DFA.

Since q_0 is the start state of NFA, $\{q_0\}$ is the start state of DFA.

The procedure to convert an NFA to its equivalent DFA is shown below:

Let $M^N = (Q_N, \Sigma, S_N, q_0, F_N)$ be an NFA and $M^D = (Q_D, \Sigma, S_D, q_0, F_D)$ such that $L(M^D) = L(M^N)$.

These should be an equivalent DFA M^D .

accepts the language $L(M^N)$.

Procedure to convert an NFA into DFA:

So, we convert an NFA into a DFA.

Practically, non-deterministic machines will not exist.

to decide some complicated language correctly.

Construction of an NFA as an efficient mechanism

Conversion from NFA to DFA

Examples :

Thus, a DFA can be obtained using NFA.

Step 4: Identify the final state of DFA and if { q_1, q_2, \dots, q_n } is a state in S and if $\{q_1, q_2, \dots, q_n\} \subseteq F$ then $\{q_1, q_2, \dots, q_n\}$ will be the final state of DFA.

Note: The state has to be selected for each state that is added to S .

\hookrightarrow Add the transition from $\{q_1, q_2, \dots, q_n\}$ to $\{q_1, q_2, \dots, q_n\}$ on the input symbol a .

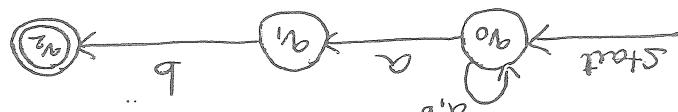
$$\begin{aligned}
 & \text{Step 2: } S^*(q_0, a_0, b) = S^*(q_0, a_0, a_0) \cup S^*(q_0, a_0, b) \\
 & S^*(q_0, a_0, a_0) = S^*(q_0, a_0, a_0) \cup S^*(q_0, a_0, a_0) \\
 & = S^*(q_0, a_0, a_0) = S^*(q_0, a_0, a_0)
 \end{aligned}$$

$$\begin{aligned}
 & \text{Step 3: } S^*(q_0, a_0, a_0) = S^*(q_0, a_0, a_0) \cup S^*(q_0, a_0, a_0) \\
 & = S^*(q_0, a_0, a_0) = S^*(q_0, a_0, a_0)
 \end{aligned}$$

- Step 3 :- Identifiy the start state of DFA - Start state of DFA is q_0
- Step 4 :- Identify the input alphabet of NFA as the input alphabet of DFA - The input alphabet of NFA are $\{a, b\}$
- Step 5 :- Identify the start state of DFA - Since q_0 is the start state of DFA
- Step 6 :- Identify the start state of NFA - [q_0] is the start state of DFA
- Step 7 :- Identify the start state of DFA - DFA. So, $S = \{a, b\}$

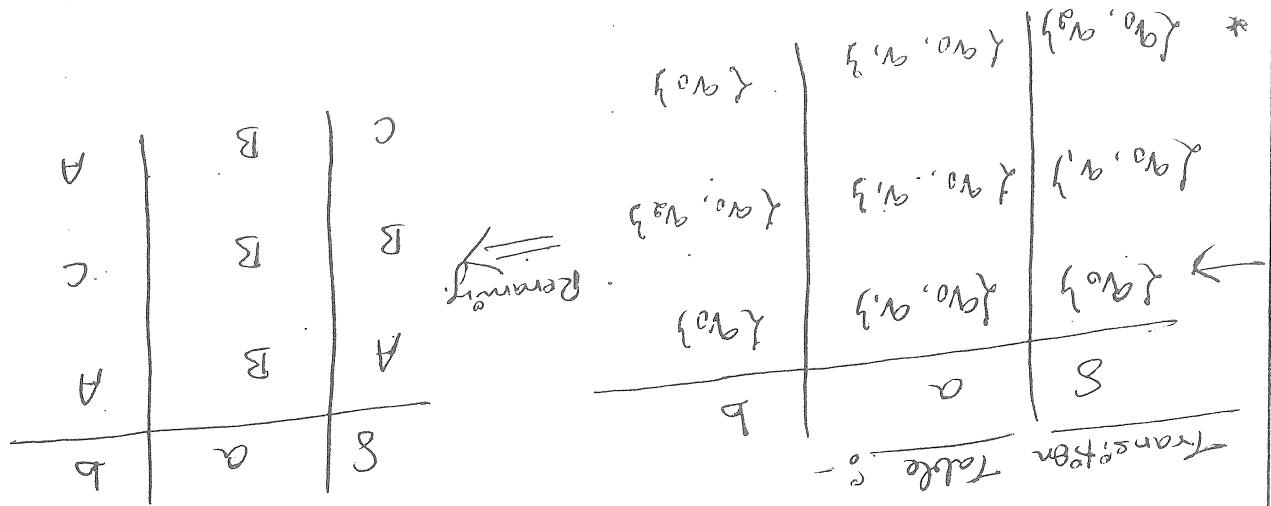
ϕ	ϕ	a_0	*
a_1	ϕ	a_1	.
a_2		a_2	.
a_0	$\{q_0, a_0\}$	a_0	\leftarrow
b	a	S	

Q1 :- The transition table for the above NFA can be written as



Q2 :- Convert the following NFA to its equivalent DFA.

Note 1 - Determining the states of DFA as A, B, C



$$F = \{q_0, q_3\}$$

Step 4 - Determine the final states of DFA: Since q_3 is the final state of NFA, whereas q_3 is present as an element, the corresponding set is the final state of DFA.

Since, no new state is generated the procedure is terminated.

$$\begin{aligned} &= \{q_0\} \cup \{\emptyset\} \\ &= S_n(\{q_0\}, b) \cup S_n(\{q_0\}, b) \\ &= S_n(\{q_0, q_3\}, b) \end{aligned}$$

$$\begin{aligned} &= \{q_0, q_3\} \cup \emptyset \\ &= \{q_0, q_3\} \end{aligned}$$

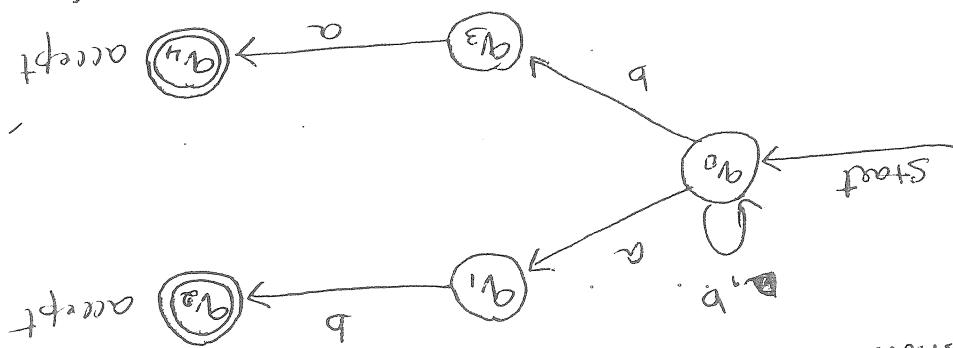
$$\begin{aligned} &= S_n(\{q_0\}, a) \cup S_n(\{q_0\}, a) \\ &= S_n(\{q_0, q_3\}, a) \end{aligned}$$

$$F = \overline{\{q_0, q_3\}}$$

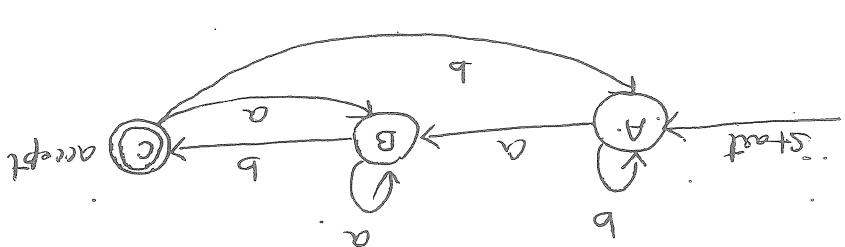
Final States: $\{q_0\}, \{q_0, q_3\}, \{q_0, q_1, q_2, q_3\}$

ϕ	ϕ	a_4	*	S is shown using the transition table.
ϕ	a_4	a_3	a_4	
ϕ	a_4	a_2	a_4	
ϕ	a_4	a_1	a_4	
a_2	ϕ	a_0	a_3	q_0 is the start state
a_0, a_3	a_0, a_1	a_0	a_4	$E = \{a_3, a_4\}$
a_0, a_1	a_0	S	a_4	$Z = \{a, b\}$
a_0, a_1, a_3	a_0	S	a_4	$B = \{a_0, a_1, a_3, a_2, a_3, a_4\}$

NFA is formally defined as : $M = (Q, Z, S, q_0, F)$



Q) Observe an NFA to accept strings of a^3 and b^2 ending with ab or ba . Connect this NFA to the equivalent DFA. Sol: The min. string that can be accepted by an NFA is bab . The equivalence of either ab or ba . The equivalence between strings consisting of either ab or ba .



S is shown using transition diagram

$F = \{C\}$ is the final state

$q_0 = A$ is the start state

$Z = \{a, b\}$

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

where,

The final DFA is given by $M = (Q, Z, S, q_0, F)$

$$S_0(\{q_0, q_3\}) = \emptyset$$

$$\phi \cap \{q_0, q_3\} = \emptyset$$

$$S_0(\{q_0, q_3, b\}) = S_0(\{q_0, b\}) \cup S_0(\{q_3, b\})$$

$$\text{step 2: } \{q_0, q_1, q_4\} \cap \{q_0, q_3\} = \emptyset$$

$$\phi \cap \{q_0, q_1, q_4\} = \emptyset$$

$$S_0(\{q_0, q_3, a\}) = \frac{S_0(\{q_0, q_3\}) \cup S_0(\{q_3, a\})}{\text{for state } \{q_0, q_3\}}$$

$$\text{step 3: } \{q_0, q_3, q_5\} \cap \{q_0, q_3\} = \emptyset$$

$$\phi \cap \{q_0, q_3\} \cup \{q_5\} = \emptyset$$

$$S_0(\{q_0, q_3, b\}) = S_0(\{q_0, b\}) \cup S_0(\{q_3, b\})$$

$$\text{step 4: } \{q_0, q_3\} \cap \{q_0, q_5\} = \emptyset$$

$$\phi \cap \{q_0, q_5\} \cap \phi = \emptyset$$

$$S_0(\{q_0, q_5, a\}) = \frac{S_0(\{q_0, q_5\}) \cup S_0(\{q_5, a\})}{\text{for state } \{q_0, q_5\}}$$

$$S_0(\{q_5, b\}) = \{q_0, q_3\} \cap \{q_5, b\} = \emptyset$$

$$S_0(\{q_0, q_5, a\}) = \frac{\{q_0, q_5\}}{\text{for state } q_5}$$

$$\text{step 5: } \{q_0, q_5\} \cap \{q_0\} = \emptyset$$

Step 3: Identify the transitions of DFA: start from the

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

of NFA are the input alphabets of DFA.

Step 2: Identify the alphabet of DFA: The input alphabet

$$\text{start of DFA. so: } Q_0 = \{q_0\}$$

Since q_0 is the start state of NFA, q_0 is the start

state of DFA.

Step 1: Identify the start state of DFA:

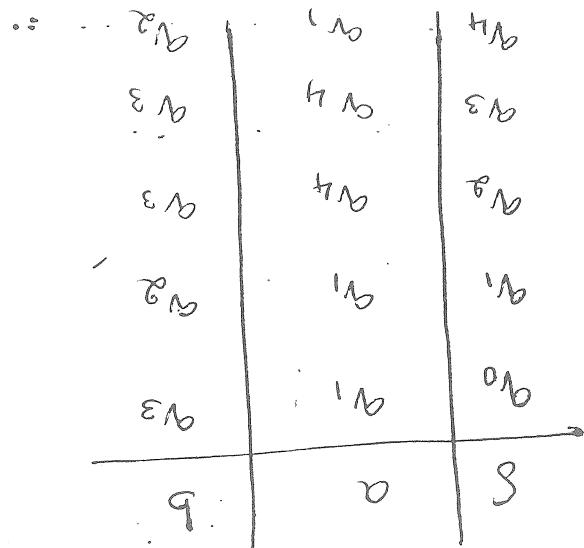
Note: Whenever a_3 and a_4 are present as an element, the corresponding set is the final set of DFA.

$\{a_0, a_3\}$	$\{a_0, a_1\}$	$\{a_0, a_1, a_4\}$	$\{a_0, a_1, a_3\}$
$\{a_0, a_3\}$	$\{a_0, a_1, a_4\}$	$\{a_0, a_1, a_3\}$	$\{a_0, a_1, a_3\}$
$\{a_0, a_3\}$	$\{a_0, a_1, a_4\}$	$\{a_0, a_1\}$	$\{a_0, a_3\}$
$\{a_0, a_3\}$	$\{a_0, a_1\}$	$\{a_0, a_3\}$	$\{a_0, a_3\}$
$\{a_0, a_3\}$	$\{a_0, a_1\}$	$\{a_0, a_3\}$	$\{a_0, a_3\} \leftarrow$

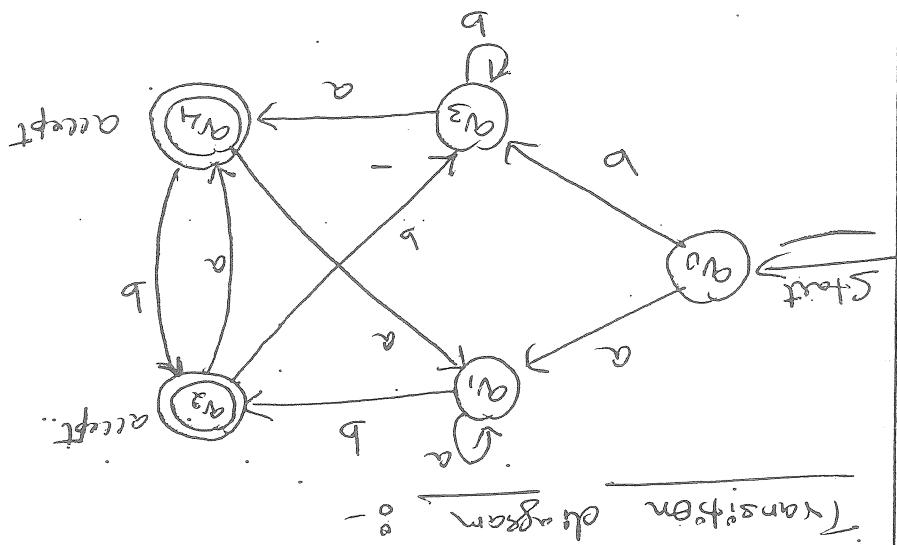
Transitions table

$$\begin{aligned}
 & S_n(\{a_0, a_1, a_3, a_4\}, b) = \{a_0, a_3, a_4\} \text{ existing state} \\
 & \{\phi\} \cap \{a_0\} \cup \{a_3\} = \{a_0, a_3\} \\
 & S_n(\{a_0, a_1, a_3, a_4\}, b) = \{a_0, a_3, a_4\} \cap S_n(\{a_0\}, b) \cup S_n(\{a_3\}, b) \\
 & \{a_0, a_3\} = \{a_0, a_3\} \text{ existing state} \\
 & \{\phi\} \cap \{a_0\} \cup \{a_3\} = \{a_0, a_3\} \\
 & S_n(\{a_0, a_1, a_3, a_4\}, b) = \{a_0, a_3, a_4\} \cap S_n(\{a_1\}, b) \cup S_n(\{a_4\}, b) \\
 & \{a_0, a_3\} = \{a_0, a_3\} \text{ existing state} \\
 & \{\phi\} \cap \{a_0\} \cup \{a_3\} = \{a_0, a_3\} \\
 & S_n(\{a_0, a_1, a_3, a_4\}, b) = \{a_0, a_3, a_4\} \cap S_n(\{a_0\}, b) \cup S_n(\{a_3\}, b) \\
 & \{a_0, a_3\} = \{a_0, a_3\} \text{ existing state} \\
 & \{\phi\} \cap \{a_0\} \cup \{a_3\} = \{a_0, a_3\} \\
 & S_n(\{a_0, a_1, a_3, a_4\}, b) = \{a_0, a_3, a_4\} \cap S_n(\{a_1\}, b) \cup S_n(\{a_4\}, b)
 \end{aligned}$$

Note:- It is observed that for every NFA those exactly same DFA will accept the same language as accepted by NFA.



- : Temporary use of the same



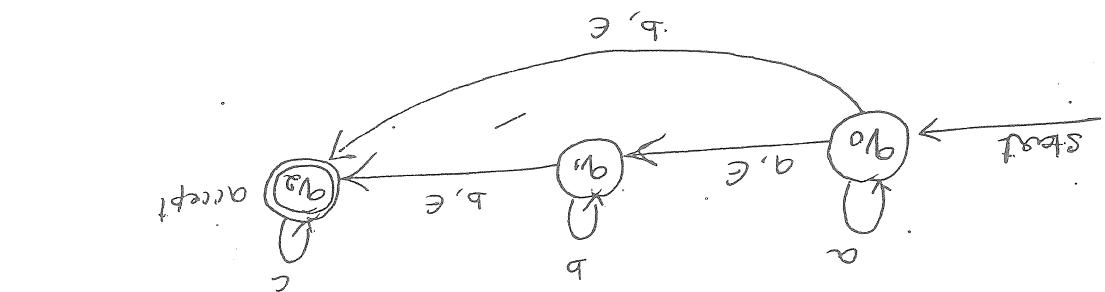
$$q_0, q_1, \dots, q_n$$

? lugubris Ag

$$\{a_0\}, \{a_0, a_1\}, \{a_0, a_2, a_3\}, \{a_0, a_3\}, \{a_0, a_1, a_2\}$$

The state of the above DFA are :

If there is an ϵ -transition, then NFA makes
 transition without reading any input symbol.
 If ϵ is present in a
 finite automaton then it is an E-NFA.
 So, if it is an NFA.
 Then, two transitions from state q_0 with input a
 In the above finite automaton, we have made
 two transitions from state q_0 with input a
 so, it is an NFA.



E-transitions - A transition with an empty input
 is called an ϵ -transition. i.e. if there is
 a transition from one state to another state
 without any input (no input implies empty string)
 denoted by ϵ) is called ϵ -transitions.

It is an extended model of NFA. An
 NFA with zero or more ϵ -transitions is called
 E-NFA (Finite Automata with ϵ -transition).

E-NFA, Minimization

Consider the following E-NFA : If a state changes after reading an input symbol there is a change of state after reading an input symbol a , then we can say that a is an input symbol a .

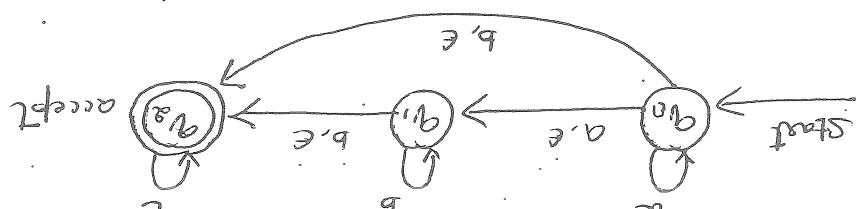
Input : $Z = \{a, b\}$

$Q = \{q_0, q_1, q_2\}$ denotes as : The power set of Q is denoted by Q^* which is the subset of Q^* set Q .

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

States : The E-NFA has three states q_0, q_1, q_2 .

Complement of E-NFA from the above figure -



Consider the following E-NFA :-

↳ Final State (q_3): q_3 with two circles
 ↳ Start State (q_0): q_0 with the label start
 ↳ Returns a set of states to the machine enters into
 ↳ Input symbol in Σ as a as the second parameter
 ↳ Step A as the first parameter
 ↳ The function accept to q_A

The transition function is defined as
 What is a transition function for E-NFA?

[Shows that]
 $g: Q \times (\Sigma \cup \{e\}) \rightarrow Q$

q	$= (q_0, g)$	g	ϵ	q_3
$(q_2, g) = (q_3, e)$	$g(q_2, e) = q_3$	g	e	q_3
$g(q_1, g) = (q_2, e)$	$g(q_1, e) = q_2$	g	e	q_2
$g(q_0, g) = (q_1, e)$	$g(q_0, e) = q_1$	g	e	q_1
$g(q_1, g) = (q_0, e)$	$g(q_1, e) = q_0$	g	e	q_0
$g(q_2, g) = (q_1, e)$	$g(q_2, e) = q_1$	g	e	q_1
$g(q_3, g) = (q_0, e)$	$g(q_3, e) = q_0$	g	e	q_0
$g(q_0, g) = (q_2, e)$	$g(q_0, e) = q_2$	g	e	q_2
$g(q_1, g) = (q_2, e)$	$g(q_1, e) = q_2$	g	e	q_2
$g(q_2, g) = (q_3, e)$	$g(q_2, e) = q_3$	g	e	q_3

Current state Input Next state Permutation Current state

What is an e-NFA?
 The e-NFA is a 5-tuple of quintuple endearing
 $(Q, \Sigma, \delta, q_0, F)$
 where -
 Q is non-empty, finite set of states
 Σ is non-empty, finite set of symbols
 δ is non-empty, finite set of states
 q_0 is a transition function which is mapping from
 Σ to non-empty, finite set of alphabets
 Σ as transition function which is mapping from
 $Q \times (\Sigma \cup \epsilon) \rightarrow Q$
 Based on the condition those can be a
 transition to other states with or without any
 input symbol.
 Transitions to other states without any input symbol
 $q_0 \in Q$ is the start state
 $F \subseteq Q$ is set of accepting states

The e-NFA has five components:
 $M = (Q, \Sigma, \delta, q_0, F)$

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

(States, Input - alphabets, transitions, start, final)

The e-NFA has five components:

The $E\text{-closure}$ of q is denoted by $E\text{Closure}(q)$.

The set of all states which are reachable from q via ϵ -transitions only is called $E\text{-closure}$ of q .

If $E\text{Closure}(q)$ contains p and if there is a transition from state p to state q labelled a , then state q is also in $E\text{Closure}(q)$.

Example: — The $E\text{-closure}(q)$ for each $q \in Q$ is shown below:

```

graph LR
    Start((Start)) -- "a" --> Start
    Start -- "b,e" --> q0((q0))
    Start -- "a,e" --> q0
    q0 -- "a" --> q0
    q0 -- "b" --> q0
    q0 -- "a,e" --> Start
    q0 -- "b,e" --> Start
    q0 -- "a" --> q1((q1))
    q0 -- "b" --> q1
    q1 -- "a,e" --> Start
    q1 -- "b,e" --> Start
    q1 -- "a,e" --> q0
    q1 -- "b,e" --> q0
    q1 -- "a" --> q2((q2))
    q1 -- "b" --> q2
    q2 -- "a,e" --> Start
    q2 -- "b,e" --> Start
    q2 -- "a,e" --> q1
    q2 -- "b,e" --> q1
    
```

$E\text{Closure}(q_0) = \{q_0, q_1, q_2\}$

$E\text{Closure}(q_1) = \{q_1, q_2\}$

$E\text{Closure}(q_2) = \{q_2\}$

The states q_0 and q_2 are reachable from q_1 without giving any input. So, the state q_1 is reachable from q_2 without giving any input. So, the states q_0 , q_1 and q_2 are reachable from q_1 without giving any input. So, the state q_2 is reachable from q_0 without giving any input. So, the states q_0 and q_2 are reachable from q_0 .

$E\text{Closure}(q_0) = \{q_0, q_1, q_2\}$

$E\text{Closure}(q_1) = \{q_1, q_2\}$

$E\text{Closure}(q_2) = \{q_2\}$

The states q_0 and q_2 are reachable from q_0 without giving any input. So, the state q_1 is reachable from q_0 without giving any input. So, the states q_0 , q_1 and q_2 are reachable from q_0 .

What is $E\text{-closure}$?

ϕ	a_1	a_2	ϕ	ϕ	a_3
a_1	ϕ	a_1	a_1	ϕ	a_1
a_2	ϕ	ϕ	a_2	a_2	a_2
a_3	a_3	a_3	a_3	a_3	E

Q as shown below using the transition table:

q_2 is final state

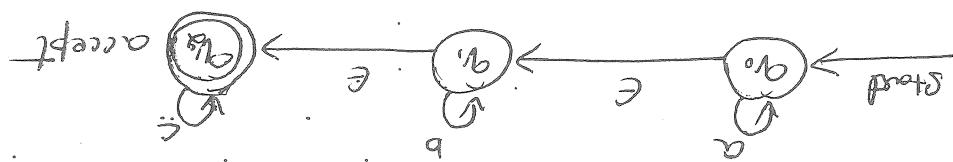
q_0 is start state

$$Z = \{a, b, c\}$$

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

where:

Thus, an E-NFA is given by $M = (Q, Z, S, q_0, F)$



3) Design an E-NFA which accepts strings consisting of zero or more digits followed by zero or more digits.

ϕ	a_3	ϕ	a_3	*
ϕ	ϕ	ϕ	a_3	
ϕ	ϕ	ϕ	a_1	
a_1	ϕ	a_1	a_0	
a	b	a	S	

S is shown using the transition table:

$$E = \{q_3\}$$

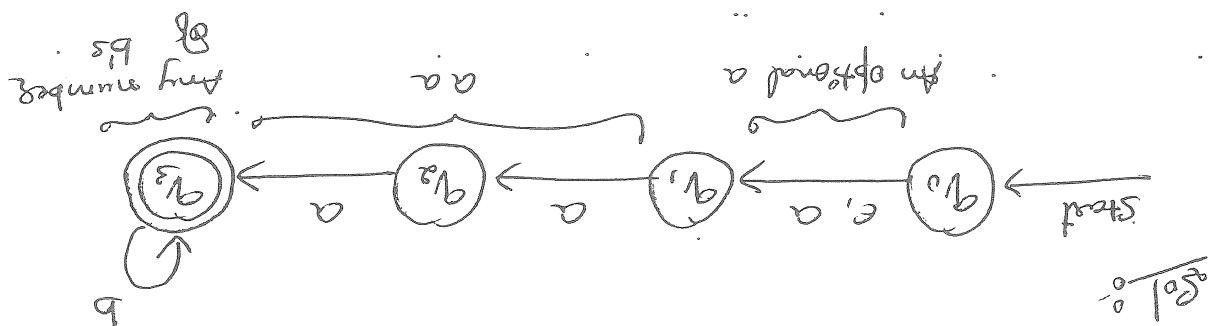
q_0 is start state

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

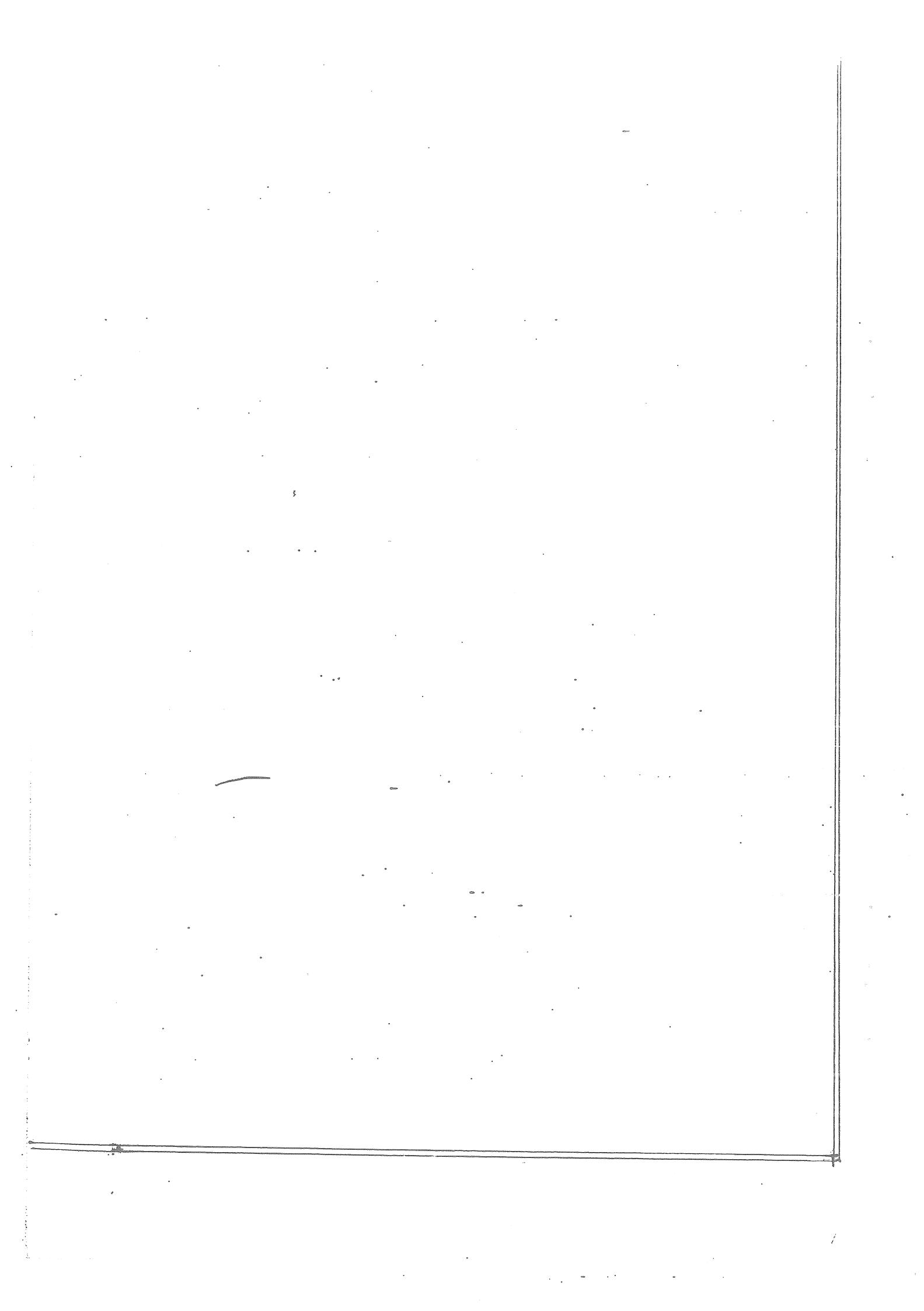
$$Q = \{q_0, q_1, q_2, q_3\}$$

$$M = (Q, Z, S, q_0, E)$$

Thus, an NFA is given by -



M is made up of an optional a followed by a and b .



$$\phi = S(\{q_3, a\})$$

on input a:

Consider the state $\{q_3\}$:

$$S(\{q_1, q_3, c\}) = \text{CLOSE}(\{q_3\}) = \{q_3\}$$

on input c:

$$S(\{q_1, q_3, b\}) = \text{CLOSE}(\{q_3\}) = \{q_1, q_3\}$$

on input b:

$$\phi = S(\{q_1, q_3, a\})$$

on input a:

Consider the state $\{q_1, q_3\}$:

$$(S(q_0, q_1, q_3, c)) = \text{CLOSE}(\{q_3\}) = \{q_3\}$$

on input c:

$$S(\{q_0, q_1, q_3, b\}) = \text{CLOSE}(\{q_3\}) = \{q_1, q_3\}$$

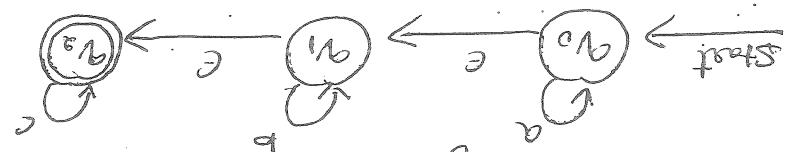
on input b:

$$S(\{q_0, q_1, q_3\}) = \text{CLOSE}(\{q_0\}) = \{q_0, q_1, q_3\}$$

on input a:

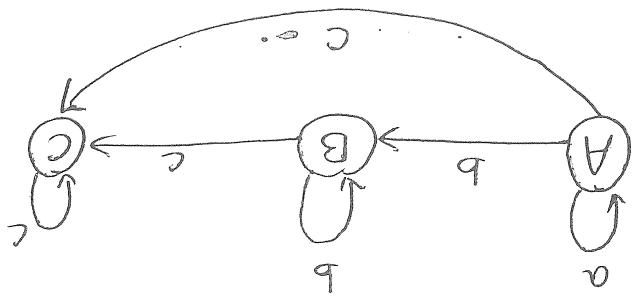
Consider the state $\{q_0, q_1, q_3\}$:

Solution: Evidently the start state of DFA is $\text{CLOSE}(q_0)$ i.e. $\{q_0, q_1, q_3\}$. Since q_0 is the start state of NFA i.e. $\text{CLOSE}(q_0) = \{q_0, q_1, q_3\}$ the start state of DFA is $\{q_0, q_1, q_3\}$.



Convert the following NFA to its equivalent DFA.

4.



Transistor diagram:

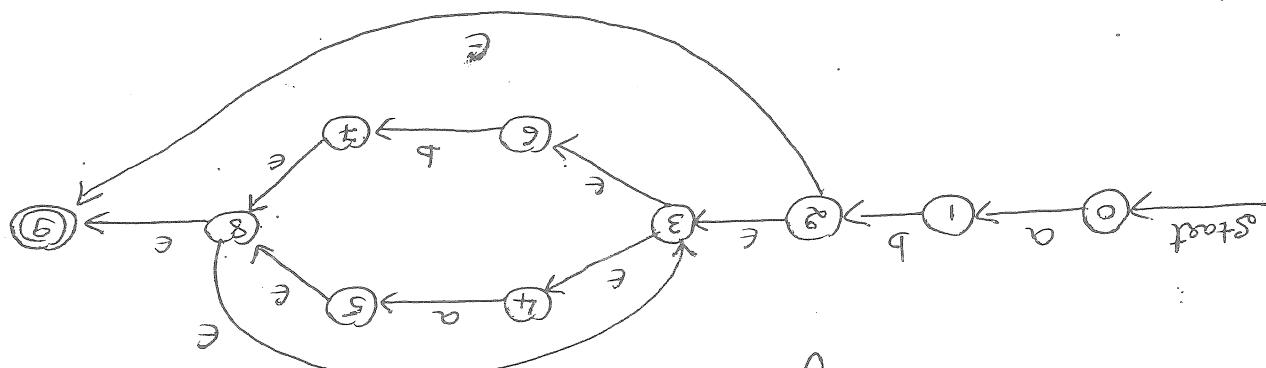
$\{a_1\}$	ϕ	ϕ	$\{a_1, a_2\}$
$\{a_2\}$	$\{a_1, a_2\}$	ϕ	$\{a_1, a_2\}$
$\{a_3\}$	$\{a_1, a_2\}$	$\{a_1, a_2, a_3\}$	$\{a_1, a_2, a_3\}$
c	b	a	g

The transistor table is shown below:

$$\text{on input } c : \quad S(a_3, b) = \text{close}(a_1) = \{a_1\} \quad \text{on output } c : \quad S(a_3, b) = \text{open}(a_1) = \phi$$

$$\text{on input } b : \quad S(a_3, b) = \phi \quad \text{on output } c : \quad S(a_3, b) = \{a_1\}$$

Since 0 is the start state of DFA.
 So, - Identify the start state of DFA.
 Since 0 is the start state of DFA, $\text{CLOSE}(0)$ is
 the start state of DFA.
 $\text{CLOSE}(0) = \{0\}$
 $S(A, a) = \{1\}$
 $S(A, b) = \{\emptyset\}$
 $S(B, a) = \text{CLOSE}(S_e(B, a))$
 $S(B, b) = \text{CLOSE}(S_e(B, b))$
 $S(C, a) = \text{CLOSE}(S_e(C, a))$
 $S(C, b) = \{5, 6, 9\}$
 $S(D, a) = \{3, 4, 5, 6, 8, 9\}$
 $S(D, b) = \{5, 6, 9\}$
 $S(E, a) = \{3, 4, 5, 6, 8, 9\}$
 $S(E, b) = \{5, 6, 9\}$



(Q) Convert the following E-NFA to its equivalent DFA.

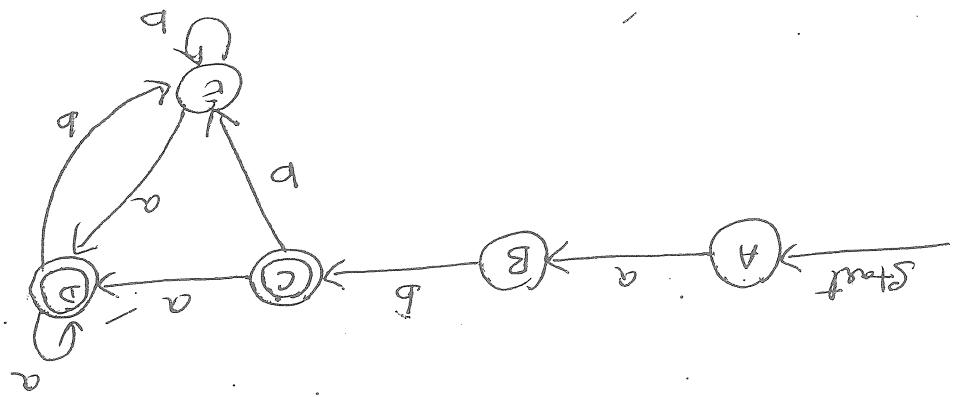
$$\begin{aligned}
 &= \{3, 4, 6, 7, 8, 9\} \text{ (Explain step 8+9)} \\
 &= \{7, 8, 9, 3, 4, 6\} \\
 &= \text{closure}\{7\} \\
 S(E, b) &\leq \text{closure}(S_E(E, b))
 \end{aligned}$$

$$\begin{aligned}
 &= \{3, 4, 5, 6, 8, 9\} \text{ (Explain step 8+10)} \\
 &= \text{closure}\{5\} \\
 S(E, a) &= \text{closure}(S_E(E, a))
 \end{aligned}$$

$$\begin{aligned}
 &= \{3, 4, 6, 7, 8, 9\} \text{ (Explain step 8+11)} \\
 &= \{7, 8, 9, 3, 4, 6\} \\
 &= \text{closure}\{7\} \\
 S(D, b) &= \text{closure}(S_D(D, b))
 \end{aligned}$$

$$\begin{aligned}
 &= \{3, 4, 5, 6, 8, 9\} \text{ (Explain step 8+12)} \\
 &= \{5, 8, 9, 3, 4, 6\} \\
 &= \text{closure}\{5\} \\
 S(D, a) &= \text{closure}(S_D(D, a))
 \end{aligned}$$

$$\begin{aligned}
 &= \{3, 4, 6, 7, 8, 9\} \\
 &= \{7, 8, 9, 3, 4, 6\} \\
 &= \text{closure}\{7\} \\
 S(C, b) &= \text{closure}(S_C(C, b))
 \end{aligned}$$



			E
			D
			C
			B
			A
	a	b	c
	b	c	d
	c	d	e
	d	e	
	e		

Transition Table:

$$\{ {}^n \Phi \} = (\Phi_n \cap \Phi) \text{ if } n > 0$$

$$E_{\text{CoF}}(g_{a_0, a_1} \cup g_{a_2, a_3})$$

$$\text{E}[\log \text{sf}(x_0, x_1)] = \text{E}[\log \text{sf}(x_0, x_3)]$$

$$f_{\alpha_0, \beta_0} = \text{classe}(q_0)$$

• *Stair* *Stair* *Stair* *Stair* *Stair*

To obtain $S(a_0, 4.+) =$
Since a_0 is the start state of e-NFA, close(a_0) is the

· ۷۷۰

8. Shown using the transitive

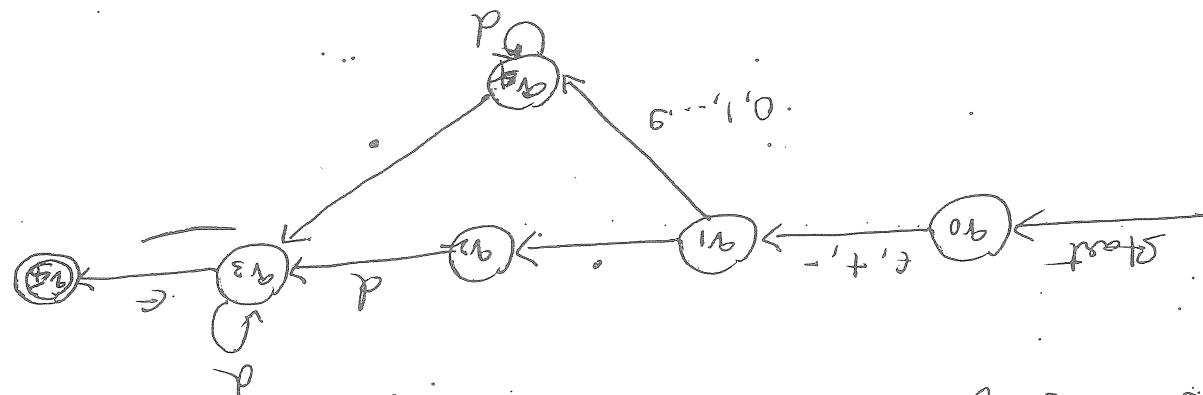
the final state off the field $\{ \phi_b \} = \{$

$$f_{\{0,1\}}(g) = \{0,1\}$$

$$\{p^{\prime \prime \prime \prime -}+\} = 3$$

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

The module $M = (\mathbb{Q}, \mathbb{Z}, S, \alpha, F)$



Q5:- A decimal number has an effect of $-8e$ - $3e$ + $8e$. The effect of same degree is followed by a decimal of $0.1e$, and it is followed by a decimal of $0.01e$. The effect of $8e$ is followed by a decimal of $0.001e$.

3) Obtain an NFA with e-transitions to accept demand numbers and obtain $g(a_0, 4^k)$

- (iii) Compute the automata to DFA
by the automata
- (ii) Give all the states of length three as less accepted
by the automata
- (i) Compute E-closure of each state

$\{p\}$	\emptyset	$\{q\}$	$\{r\}$	$\{s\}$	\emptyset
	\emptyset	$\{q\}$	$\{r\}$	$\{s\}$	\emptyset
$\{q\}$	$\{q\}$	$\{r\}$	$\{r\}$	$\{s\}$	\emptyset
$\{r\}$	$\{q\}$	$\{r\}$	$\{p\}$	\emptyset	\emptyset
$\{s\}$	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset

a) Consider the following DFA

i) Determine the equivalent DFA for the above DFA.
Automaton:

as accepted

Since q_5 is the final state, the string for

$$\text{Thus } C(q_0, q_5) = \{q_3, q_5\}$$

$$= \{q_3, q_5\}$$

$$= \text{Eclose}(q_3)$$

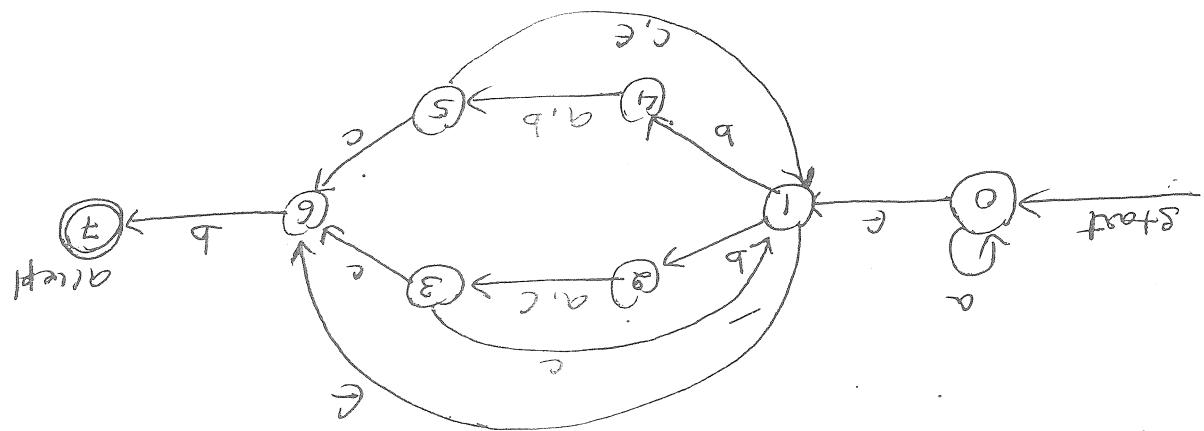
$$= \text{Eclose}(q_3 \cup \emptyset)$$

$$S((q_3, q_5), \tau) = \text{Eclose}(q_3 \cup q_5) = S((q_3, \tau) \cup S((q_5, \tau))$$

$$= \{q_3, q_5\}$$

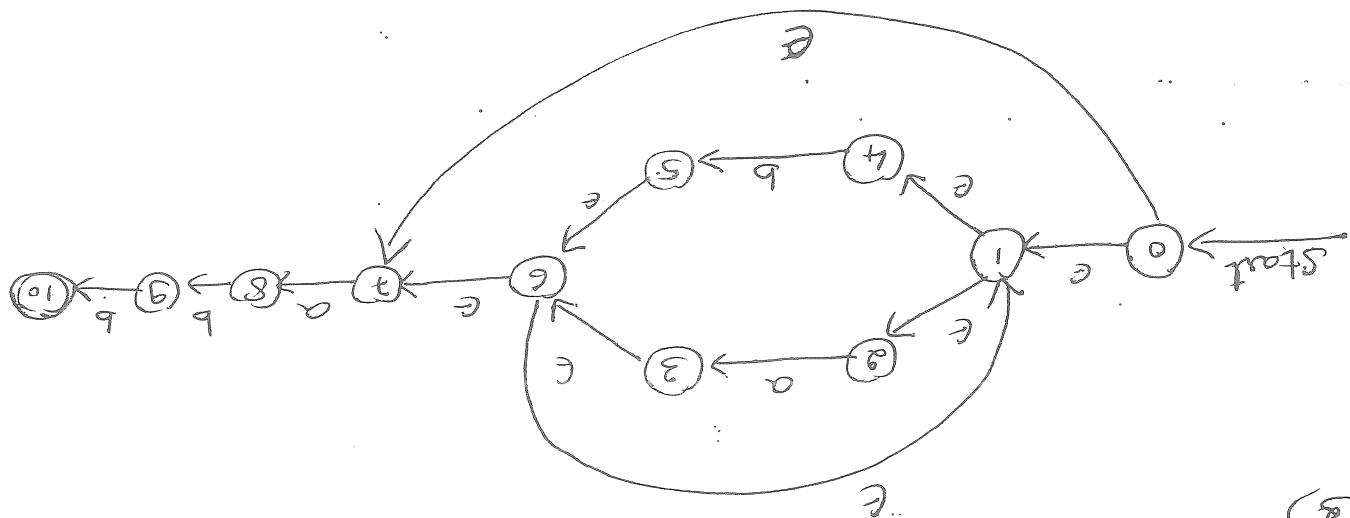
$$= \text{Eclose}(q_3)$$

$$S((q_3, q_5), \tau) = \text{Eclose}(S((q_3, \tau)))$$



4) Obtain the equivalent DFA for the following E-NFA.

Obtain an equivalent DFA



(8)

← Spilt the classes

$$\begin{array}{ll}
 ((5, b), [2, 4]) & ((5, a), [2, 4]) \\
 ((6, b), [1, 3, 5, 6]) & ((6, a), [1, 3, 5, 6]) \\
 & ((1, a), [2, 4]) \\
 & ((3, b), [2, 4]) \\
 & ((1, a), [2, 4]) \\
 & ((2, a), [1, 3, 5, 6])
 \end{array}$$

← No splitting is recommended

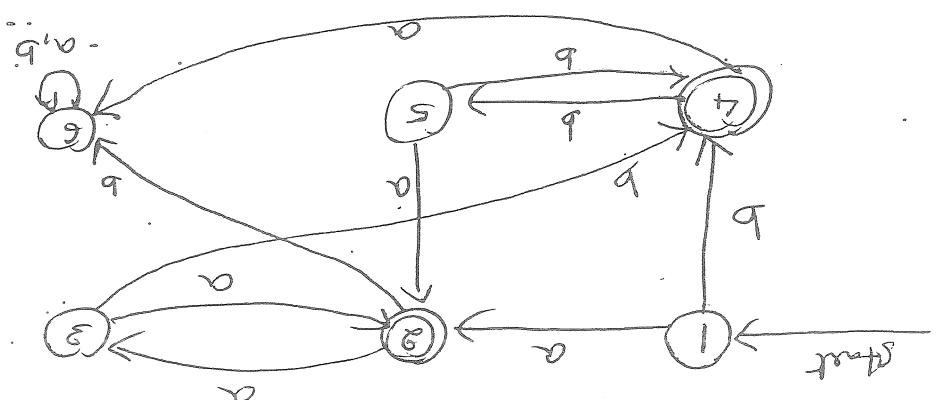
$$\begin{array}{ll}
 ((2, b), [1, 3, 5, 6]) & ((2, a), [1, 3, 5, 6]) \\
 ((4, a), [1, 3, 5, 6]) & ((4, a), [1, 3, 5, 6])
 \end{array}$$

Split:

Finally classes are : $\{ [2, 4], [1, 3, 5, 6] \}$

$\xrightarrow{\text{Acceptable}}$ $\xleftarrow{\text{Not acceptable}}$

$\{ 1, 2, 3, 4, 5, 6 \}$	$\{ 6 \}$	$\{ 2 \}$	$\{ 5 \}$	$\{ 4 \}$	$\{ 3 \}$	$\{ 2 \}$	$\{ 1 \}$
a	b	a	b	a	b	a	b
a	b	a	b	a	b	a	b
a	b	a	b	a	b	a	b
a	b	a	b	a	b	a	b



Using min. DFS find a minimal matching.

class : $\{ [6], [5, 6], [4, 5], [4, 3, 5], [6] \}$
 file no 4
 $(5, 6), (4)$
 $(5, 6), (4)$
 $(3, 6), (4)$
 $(3, 4), (3)$
 $(1, 6), (4)$
 $(1, 4), (2)$
 \therefore 3

class = $\{ [6], [5, 6], [4, 5], [4, 3, 5], [6] \}$
 and [6]

file no 6
 $(6, 6), (6)$
 $(6, 6), (6)$
 not parallel

parallel or shifting no
 $(5, 6), (4, 5)$
 $(5, 6), (4, 5)$

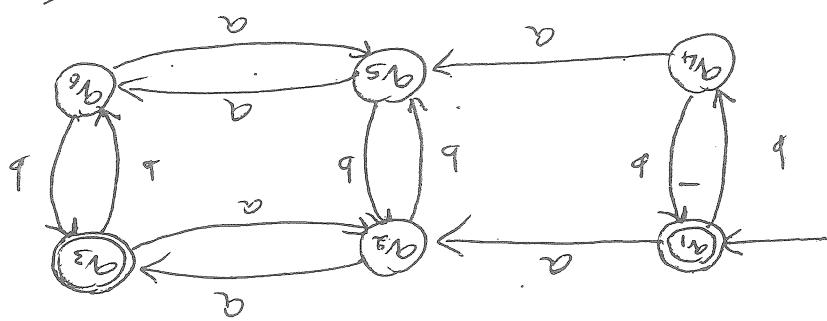
$(3, 6), (3, 4)$
 $(3, 4), (3, 4)$
 $(1, 6), (4, 3)$
 $(1, 4), (2)$

class : $\{ [6], [5] \}$ and [4]
 there two classes must be split:
 $(4, 6), [6]$
 $(4, 6), [6]$
 $(4, 6), [6]$
 \therefore 2

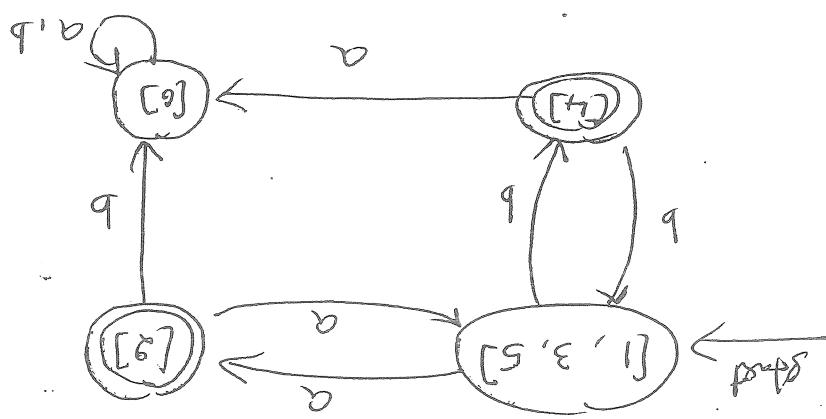
\therefore class = $\{ [4, 6], [1, 3, 5], [6] \}$

class : $[1, 3, 5], [6]$

Equivalent class: $\{ [a_1, a_2], [a_3, a_4, a_5, a_6] \}$



Q) Find the minimum DFA for the automata given.



$\Pi_0 = \{ [A, B, C, D, E, F, G, H], [C] \}$	$\Pi_1 = \{ [A, E, G], [B, H], [D, F], [C] \}$
$\Pi_2 = \{ [A, B, E, G, H], [D, F], [C] \}$	
$\Pi_3 = \{ [A, B, E, G, H], [D, F], [C] \}$	
$\Pi_4 = \{ [A, B, D, E, F, G, H], [C] \}$	

G11

G12

G13

G14

G15

G16

G17

G18

G19

G20

G21

G22

G23

G24

G25

G26

G27

G28

G29

G30

G31

G32

G33

G34

G35

G36

G37

G38

G39

G40

G41

G42

G43

G44

G45

G46

G47

G48

G49

G50

G51

G52

G53

G54

G55

G56

G57

G58

G59

G60

G61

G62

G63

G64

G65

G66

G67

G68

G69

G70

G71

G72

G73

G74

G75

G76

G77

G78

G79

G80

G81

G82

G83

G84

G85

G86

G87

G88

G89

G90

G91

G92

G93

G94

G95

G96

G97

G98

G99

G100

G101

G102

G103

G104

G105

G106

G107

G108

G109

G110

G111

G112

G113

G114

G115

G116

G117

G118

G119

G120

G121

G122

G123

G124

G125

G126

G127

G128

G129

G130

G131

G132

G133

G134

G135

G136

G137

G138

G139

G140

G141

G142

G143

G144

G145

G146

G147

G148

G149

G150

G151

G152

G153

G154

G155

G156

G157

G158

G159

G160

G161

G162

G163

G164

G165

G166

G167

G168

G169

G170

G171

G172

G173

G174

G175

G176

G177

G178

G179

G180

G181

G182

G183

G184

G185

G186

G187

G188

G189

G190

G191

G192

G193

G194

G195

G196

G197

G198

G199

G200

G201

G202

G203

G204

G205

G206

G207

G208

G209

G210

G211

G212

G213

G214

G215

G216

G217

G218

G219

G220

G221

G222

G223

G224

G225

G226

G227

G228

G229

G230

G231

G232

G233

G234

G235

G236

G237

G238

G239

G240

G241

G242

G243

G244

G245

G246

G247

G248

G249

G250

G251

G252

G253

G254

G255

G256

G257

G258

G259

G260

G261

G262

G263

G264

G265

G266

G267

G268

G269

G270

G271

G272

G273

G274

G275

G276

G277

G278

G279

G280

G281

G282

G283

G284

G285

G286

G287

G288

G289

G290

G291

G292

G293

G294

G295

G296

G297

G298

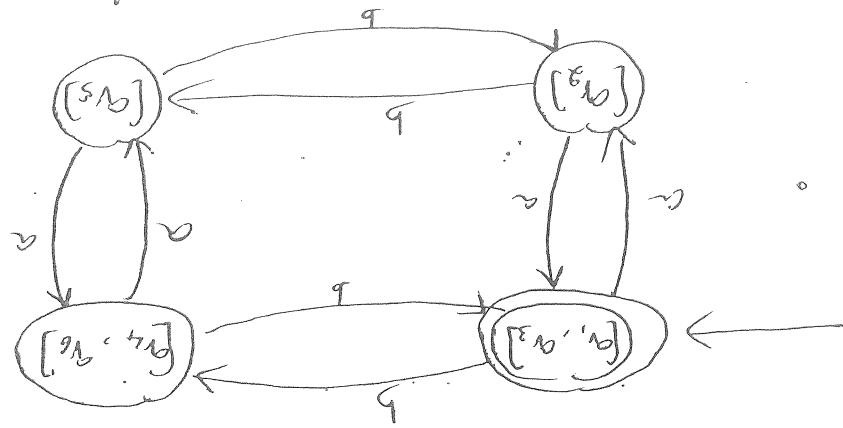
G299

G300

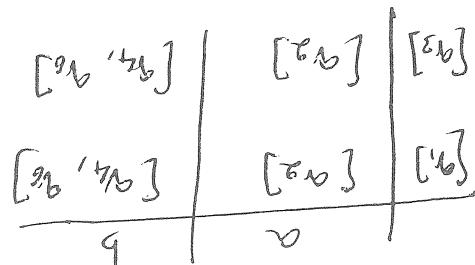
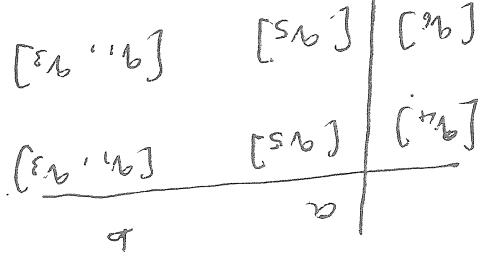
② Obtain the distinguishable table for the alternation and then minimize the state of DFA.

Employee Id: 481976
Employee Name: Venugopal VanajakshiAkashminalarayyan
Official Email ID: VVO0481976@TechMahindra.com
Submission Date: 16/11/2018
Description: Flexi Plan
Business Purpose: Rembursement
Status: Submitted
Default Financing Location: Cisco-BR-CessnaPark
Accounting Date: 16/11/2018
Accounting Template: STANDARD
Department to be Debited: IBU-CTP-01
Project Activity ID: OVERHEAD
Creation Date: 16/11/2018
By: 481976

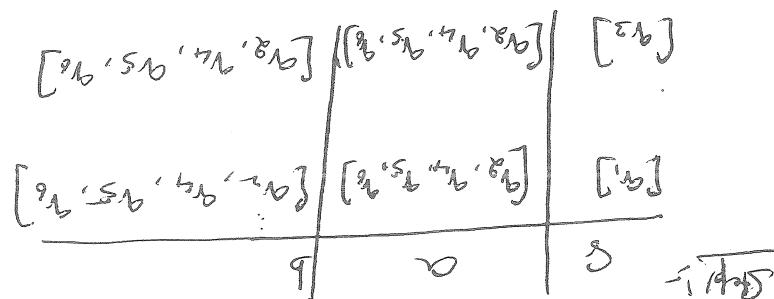
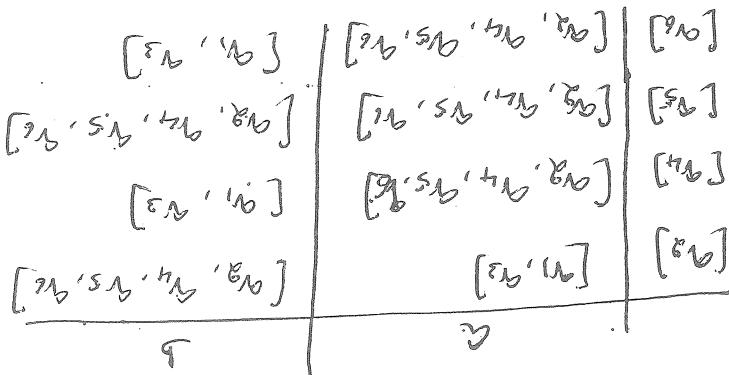
Mutating DNA



$$\text{Equivalent class} = \{[a_4, a_6], [a_4, a_3], [a_4, a_6], [a_5]\}$$



$$\text{Step 2, Equivalent class} = \{[a_4, a_6], [a_4, a_3], [a_4, a_6], [a_5]\}$$



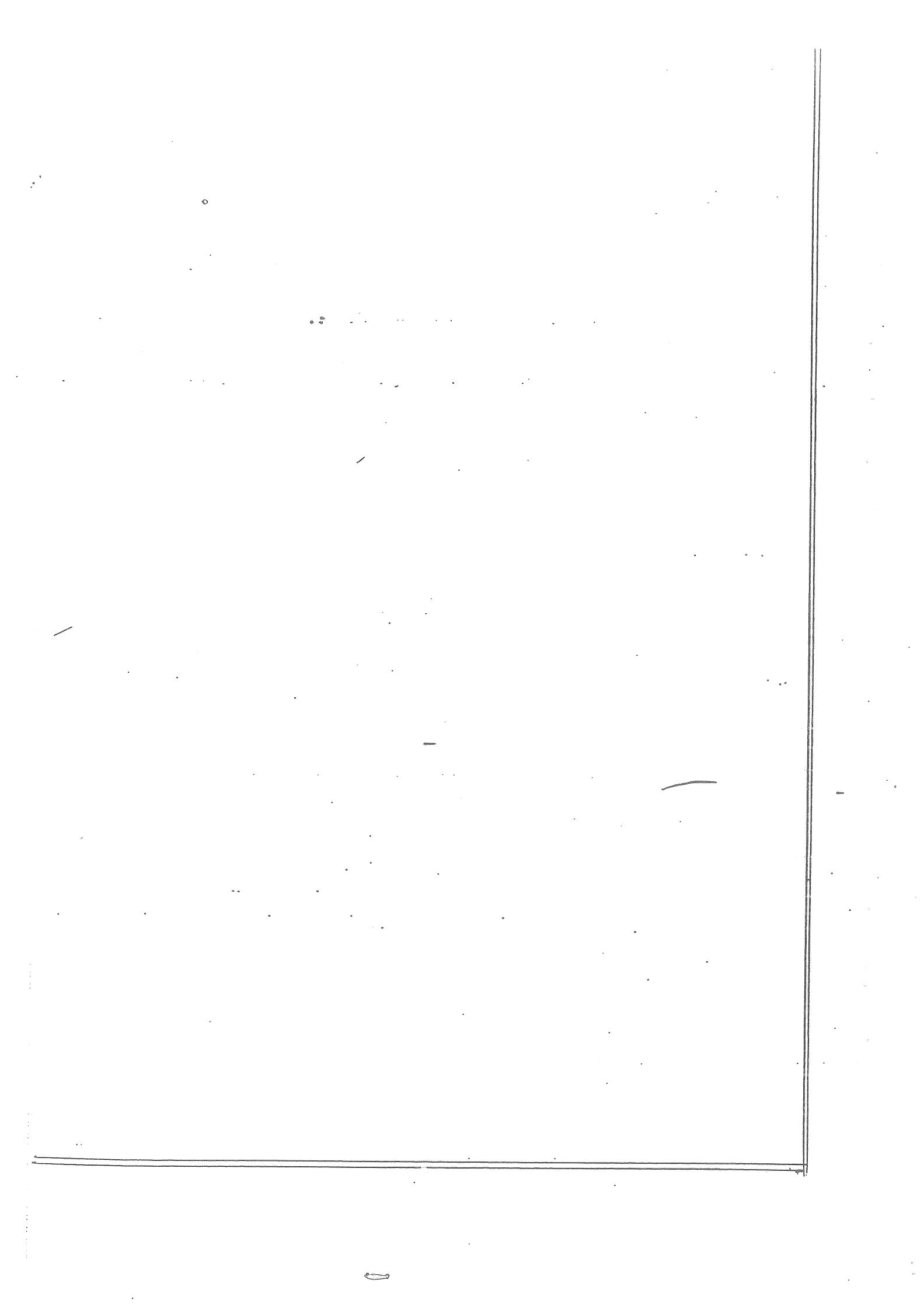
1) Find: the minimum pad DFA for the following

\therefore A es igual a

Q) Explain the difference between the following terms of classification of software for the purpose of DPA.

B	A	B	C	D	E	F	G
C	D	E	F	G	H	I	J
D	E	F	G	H	I	J	K
E	F	G	H	I	J	K	L
F	G	H	I	J	K	L	M
G	H	I	J	K	L	M	N
H	I	J	K	L	M	N	O
I	J	K	L	M	N	O	P
J	K	L	M	N	O	P	Q
K	L	M	N	O	P	Q	R
L	M	N	O	P	Q	R	S
M	N	O	P	Q	R	S	T
N	O	P	Q	R	S	T	U
O	P	Q	R	S	T	U	V
P	Q	R	S	T	U	V	*
Q	R	S	T	U	V		

3) Meaningful following Dia using full filling also



The decision procedures of regular languages give us algorithms for decoding letters into automata that have the same language. If the two automata accept the same string, we can minimize the automata that have the cost of the circuit tends to decrease.

Equivalence and minimization of finite automata:

The language generated by a DFA is unique. But, there can exist many DFAs that accept the same language. In such cases, the DFAs are said to be equivalent. During computations, it is desirable to keep the DFA with fewer states since the space is proportional to the no. of states of DFA. This can be achieved by minimizing the DFA. Two can be found first by finding the distinguishing state and the number of states and then it is compared to find states that are equal (indistinguishable) if and only if $S(p, w)$ and $S(q, w)$ are non-final states for all $w \in \Sigma$.

Indistinguishable: Two states p and q of a DFA are indistinguishable if and only if $S(p, w)$ and $S(q, w)$ are final states for all $w \in \Sigma$. i.e. $S(p, w) \in F$ and $S(q, w) \in F$ and $S(p, w) \neq S(q, w)$ and $S(q, w) \neq S(p, w)$ are non-final states for all $w \in \Sigma$.

Equivalent: Two states p and q of a DFA are equivalent if and only if $S(p, w) \neq S(q, w)$ and $S(q, w) \neq S(p, w)$ for all $w \in \Sigma$.

Irreducible DFA: Two states p and q of a DFA are irreducible if and only if $S(p, w) \neq S(q, w)$ and $S(q, w) \neq S(p, w)$ for all $w \in \Sigma$.

(Transitive) $\vdash ((m \cdot h) \vee (h \cdot n)) \rightarrow ((m \cdot h) \vee (h \cdot n))$

$$\left(\begin{array}{l} \text{1. } \exists x A \\ \leftrightarrow \exists x B \end{array} \right) \leftrightarrow (\exists x A \leftrightarrow \exists x B) \in \mathcal{E} \text{ as } h \in A \vdash$$

$(\forall x \in A \leftrightarrow \forall x \in A)$ ist eine Reflexivität.

as it is an equivalence relation because of the following:

→ a \neq aa
No, they are doctory's hole because a \neq L
and aa \in L

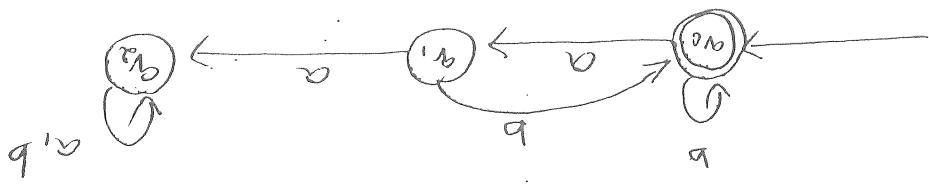
→ a aa, aac
they are Pindaric's hollow because both of them do not belongs to L

— $L \in \lambda \cap (a, b)$; $|L|$ is even; then $\frac{f_L}{2}$ (8).

Example 1: If $L = \{a_j\}$, then a_2 is an element of L . i.e. any number of a_i generated belongs to L .

$\exists \forall (m) \in S(a, m)$ and $S(p, m) \in$ definierte Mengen

Digraphs & Shadles :- If there is atleast one sharing w such that the $\{f, w\}$ and $S(a, w)$ is final state and the others is non final state, then the graph is called Digraphs & Shadles.



key point about L :

L contains both strings that are in L and strings that are not.

If these are strings that would take form for L to the dead state, then there will be no equivalence class a .

Or if a is the dead state, then there will be some equivalence class a that corresponds to dead state.

Class a is that corresponds to dead state.

comes equivalence class contains a . If we make some equivalence class contains a , it will correspond to the start state of the minimal machine that accepts L .

$L = \{w \in \Sigma^* : w \text{ is binary, } a, b \text{ is immediately followed by } b\}$

Determining L :

To determine equivalence class of w ,
 (a) $\{e, b, ab, \dots\}$ (+ all strings in L that end in a &
 b) $\{a, abba, \dots\}$ (+ strings that follow a by b
 not followed by b)

c) $\{(aa, abaa, \dots)\}$ (+ strings that are in L and contain aa
 as substring of aa)

Note:-

No equivalence class of w is empty
 Every string in L is in exactly one equivalence class of w

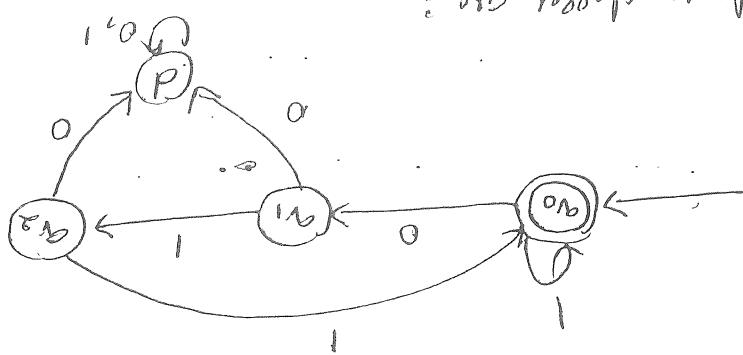
(ends in dead state) \leftarrow

(in L but ends in 0) \leftarrow

(in L but ends in 0) \leftarrow

{strings in L} \leftarrow

[equivalence classes are]:



Q01

$L = \{w \in \{0, 1\}^*: w \text{ is prime-length}\}$ followed by 11)
3) determine equivalence class. for

Note:- There may be more than one equivalence class that contains strings that are in L.

$\Rightarrow \{aa, abaa, ababb, \dots\} \subseteq L(a)$

$\{b, bab, abab, \dots\} \subseteq L(b)$

$\{a, aba, ababa, \dots\} \subseteq L(a)$

$\Rightarrow \{e\} \subseteq L(e)$

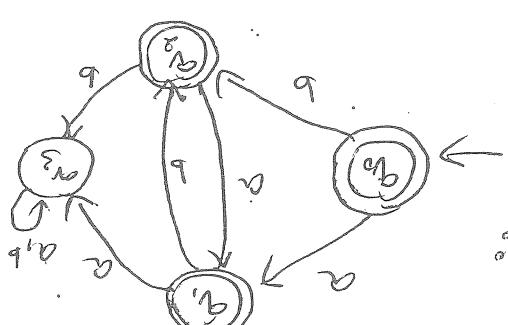
The equivalence classes are:

Q01

A determine equivalence classes of L

are same if

g) $L = \{w \in \{a, b\}^*: w \text{ has two adjacent characters}$



The equivalence classes are:

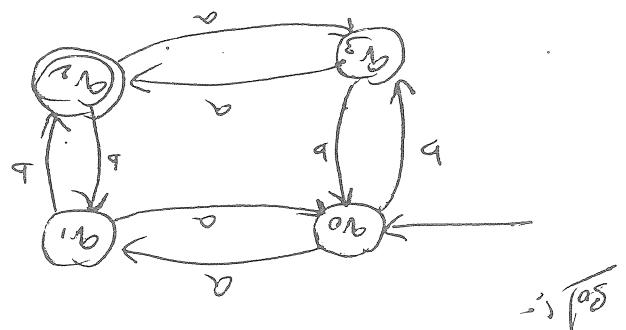
Q01

Each equivalence class depends on some number m .
 So if $L = A_m B_n$, then all a as an infinite number
 of equivalence classes, therefore a and b equals

$$\begin{aligned} & \left\{ a_1 a_2 a_3 \right\} \leftarrow \\ & \left\{ a_1 \right\} \leftarrow \\ & \left\{ a \right\} \leftarrow \\ & \left\{ e \right\} \leftarrow \left\{ e_1 \right\} \end{aligned}$$

5) Ordering is for $A_m B_n$ where $a \leq b$

$$\begin{aligned} & \left\{ \text{even numbers of } a \text{ as } b_1 b_2 \right\} \leftarrow \\ & \left\{ a \in L \text{ but one more } b \right\} \leftarrow \\ & \left\{ a \in L \text{ but one more } b \right\} \leftarrow \\ & \left\{ \text{changes in } L \right\} \leftarrow \end{aligned}$$



4) $L = \{ w \in (a, b)^*: w \text{ has odd number of } a \text{ and an odd number of } b \}$

Pigeonhole Principle states that if p items are put into n boxes with $p \geq n$ then at least one box must contain more than one item.

Proof. Suppose that the number of states in m were less than the number of equivalence classes of all elements to the power n . Then by Pigeonhole Principle, there must be atleast one equivalence class of size ≥ 2 . Then consider any two elements x, y in this class. Since $x \neq y$, there must be some i such that $x_i \neq y_i$. Then since x, y are in the same equivalence class, there must be some j such that $x_j = y_j$. This contradicts our assumption that x, y are in different equivalence classes.

The number of states in m is greater than or equal to the number of equivalence classes of all elements to the power n .

Let $M = (Q, \Sigma, S, \delta_0, F)$ be a DFA such that

Let L be regular language and

Theorem:- $\exists L$ imposes a lower bound on the minimum number of states of a DFA for L

Myhill-Nerode Theorem

Theorem: A language is regular iff the number of equivalence classes of equivalence classes of Σ is finite.

Proof: If the language is regular, then there exists some DFA M that accepts L . If L is regular, then there are m states in M . By the previous theorem, m is the number of states in M . Hence the number of equivalence classes of Σ is finite.

Hence the number of equivalence classes of Σ is finite.

If L is regular, then there exists some DFA M that accepts L . M has some finite number of states m . By the previous theorem, m is the number of states in M . Hence the number of equivalence classes of Σ is finite.

Building a minimal DFA from Σ is as follows:

Let $Z = \{a, b\}$ and $L = \{w \in \Sigma^* \mid a \text{ is adjacent to } b\}$; we have two adjacent classes $[a]$ and $[b]$ if and only if $a \in L$.

Same is

Equivalence classes of Σ are:

$\{a, aba, ababa, \dots\} \cup \{b, ab, bab, \dots\} \cup \{e\}$

$\{a, a\}$

$\{b, b\}$

$\{e\}$

Build minimal DFA M to accept L as follows:

$\left[\alpha \right] = \left(\alpha, \sim \right)$ ←
Equivalence classes \Leftrightarrow sets $\{e\} = \{a_0\}$ ←
Accepting sets are all equivalence classes ←
that contain strings in $\{a_0\} \cup \{a_1\}$ ←

Theorem :- There exists a unique minimal DFA for every regular language.

Proof :- Let L be a regular language over some alphabet Σ . Then there is a DFA M that accepts L and that has precisely n states where n is the number of equivalence classes of Σ : Any other DFA that accepts L must have more states than M as it must do equivalent to M except for some names.

Now we prove by construction of $M = (Q, \Sigma, S, \delta, q_0, F)$ that M is unique.

We define S as $\{C_i : i \in I\}$ where I is the set of equivalence classes of Σ . Here C_i contains all strings that are equivalent to each other under \equiv .

δ is defined as $\delta(C_i, a) = C_j$ where j is the state to which a moves from i .

q_0 is defined as $q_0 \in C_{q_0}$ where q_0 is the initial state.

F is defined as $\{C_i : i \in I \text{ and } C_i \cap L \neq \emptyset\}$.

Theorem :- If L is a regular language over some alphabet Σ , then there is a DFA M such that M accepts L and M has at most n states where n is the number of equivalence classes of Σ .

Proof :- Proof is by contradiction. Suppose there is another DFA M' with m states that accepts L and $m < n$. Let s be a string in L such that $s \notin M'$. Then there is a state q in M such that $s \in L(q)$ and $q \notin F$. Let t be a string such that $t \in L(q)$ and $t \notin M'$. Then $t \in L(q)$ implies $t \in L(q')$ for some $q' \in F$. But $t \notin M'$ implies $t \notin L(q')$. This is a contradiction.

Theorem :- If L is a regular language over some alphabet Σ , then there is a DFA M such that M accepts L and M has at most n states where n is the number of equivalence classes of Σ .

$(\epsilon, s) \xrightarrow{w} (\epsilon, s)$
 $\text{if } t = \epsilon, s \text{ is any string}$
 $(\epsilon, s) \xrightarrow{w} (\epsilon, s+t) \quad \text{Because } s = \epsilon$
 $(\epsilon, \epsilon) \xrightarrow{w} (\epsilon, \epsilon)$
 $\text{Combining these two tells us that read } k+1 \text{ char}$

$(\epsilon, \epsilon) \xrightarrow{w} (\epsilon, \epsilon)$ $\text{dissertation of } s$
 $\text{as we read one more character}$

$|y| = k$
 $(\epsilon, \epsilon) \xrightarrow{w} (\epsilon, \epsilon)$ induction hypothesis
 $\text{as we read first character}$
 $c = z$

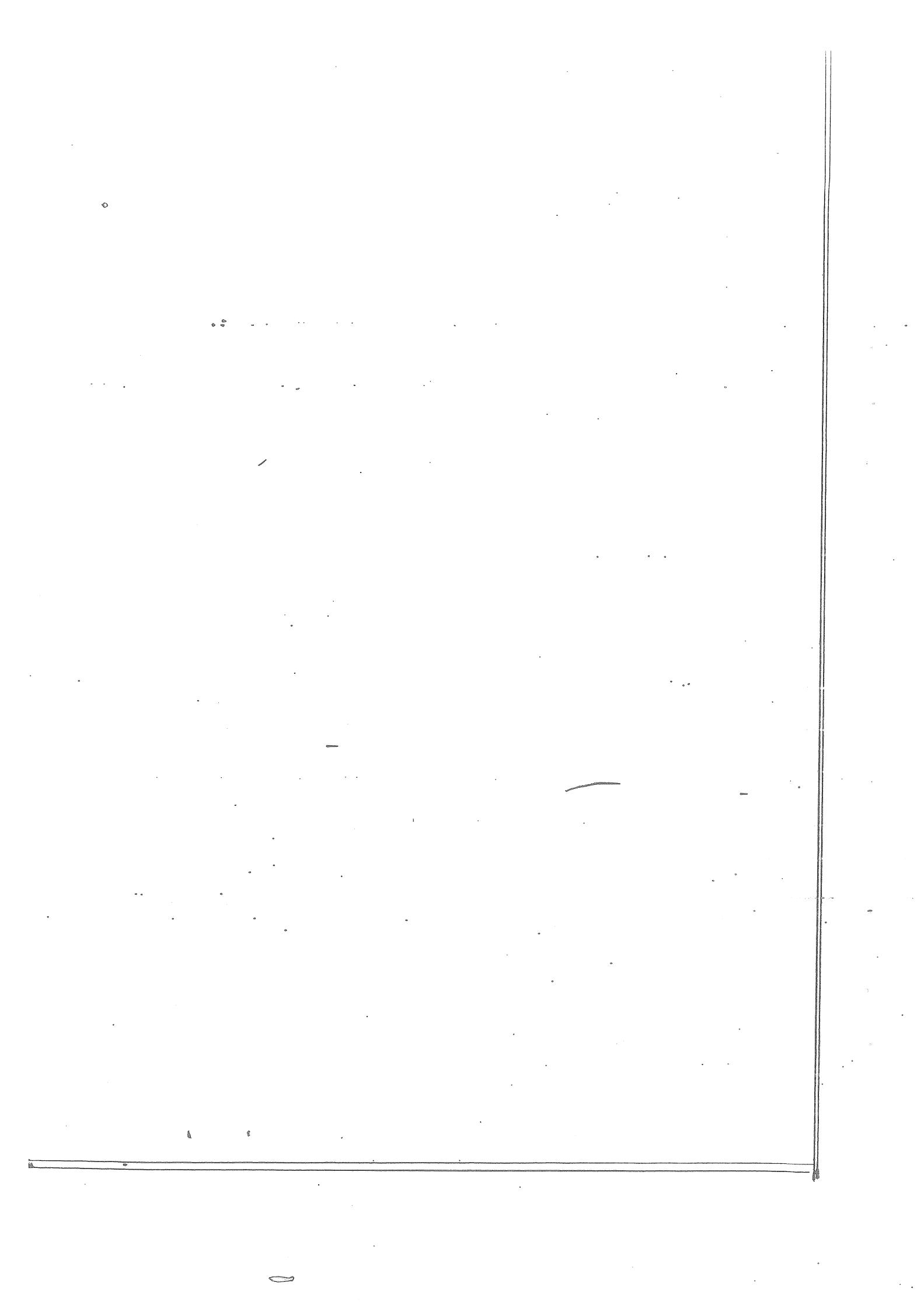
$|s| = k+1$, but $s = \epsilon$ where $\epsilon \leq$ and
 $|s| = k$, the claim is true. What happens

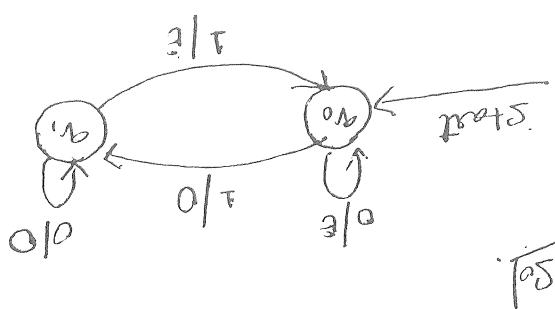
$((\epsilon, \epsilon), s)$
 $L = L(m)$ i.e. in acceptable language L . To
 prove this shows that as $\epsilon \in ((\epsilon, \epsilon), s)$

ϵ is a sequence of all square values from differentiation
 each of them. The fact that concatenation
 passes and it produces a unique value for

$* \quad \epsilon \text{ is a function defined for all } (start, input)$

m will accept $\{s\} \in A$, which by the
way in which A was constructed, if would be
if the strings in $\{s\}$ are in L , in accept
possibly strings in w .
By theorem [numbers of equivalence classes of
all impasse losses bound on numbers of strings in
any form that accepts L], there exists no
smaller machine in M that also accepts L .
* Thus if no different machine M that
also has in L and that accepts L .





Q) Construct a Mealy machine which accepts strings of 0's and 1's such that output EVEN when the input string has odd numbers of 1's and output ODD when the input string has even numbers of 1's.

Let $S = \{0, 1\}$ be the input alphabet.
 $Z = \{0, 1\}$ be the output alphabet.
 $Q = \{q_0, q_1, q_2, q_3, q_4\}$ be the states.
 $\delta: Q \times S \rightarrow Q$ be the transition function.
 $\lambda: Q \times S \rightarrow Z$ be the output function.

Mealy machine $M = (Q, S, \delta, \lambda, q_0)$ is defined as follows:

Mealy machine M is a state machine where an output is associated with each transition as called Mealy machine. A finite state machine whose output is

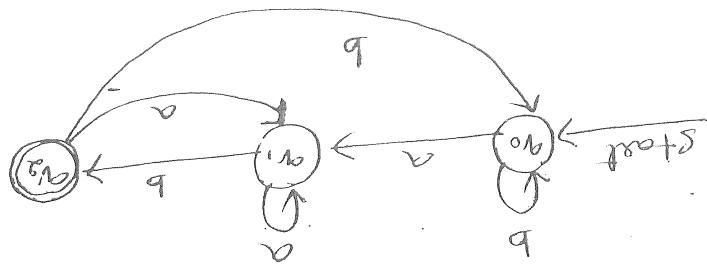
Mealy machine \leftarrow
 Moore machine \leftarrow

They are classified into two types:

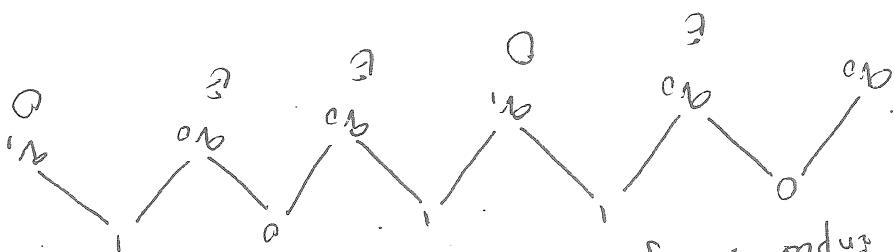
classified with either a transition or a state as called finite state transducers.

A finite state machine whose an output is

finite state Transducer \leftarrow



Q: Labeled a mealy machine which accepts strings of a 's and b 's ending with ab .
 S: q_3 is initial and q_0 is final. a 's and b 's could have same path or different paths.
 P: a_1 and b_1 's could have same number of steps and could have different numbers of steps.



The sequence of moves made by the mealy machine for the input string 01101 is:
 Step 1: Initial state q_0
 Step 2: Transition from q_0 to q_1 on input 'a' (labeled E)
 Step 3: Transition from q_1 to q_2 on input 'b' (labeled E)
 Step 4: Transition from q_2 to q_3 on input 'a' (labeled O)
 Step 5: Transition from q_2 to q_1 on input 'b' (labeled O)
 Step 6: Transition from q_1 to q_3 on input 'a' (labeled O)

	q_1	q_2	q_3
a_1	0	0	1
a_0	1	0	0
b_1	0	1	0

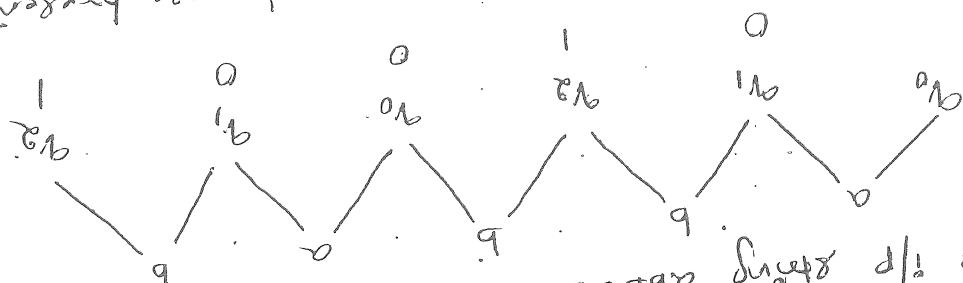
Q: What is the transition table?
 A: $T = \{ (q_i, a_j) \mid q_i \text{ is state } i, a_j \in \{a, b\} \}$
 $S = \{q_0, q_1, q_2, q_3\}$
 $Z = \{a, b\}$
 $Q = \{q_0, q_1, q_2, q_3\}$

$$M = (Q, Z, T, S, A, q_0)$$

Formally, the mealy machine can be defined as:

string abba

so, the pattern a, b, a, b is present because that on the output too is also present.



The sequence of moves made by the Mealy machine for the string abba is:

q_0	0	q_2	a
1	0	q_3	b
0	0	q_4	a
0	0	q_5	b
0	0	q_0	a

output function table

$$y = f_0, r_3$$

$$z = f_1, b_5$$

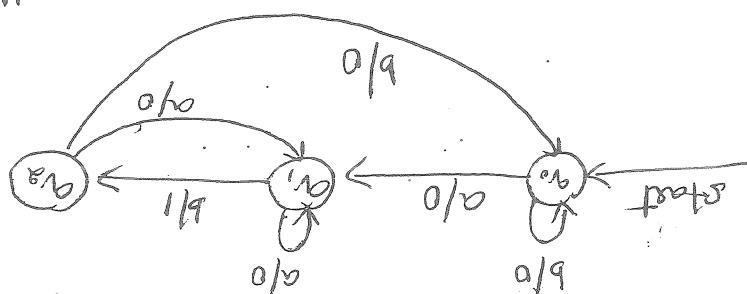
$$\theta = f_{q_0}, a, q_5$$

$$M = (Q, Z, T, S, A, \theta)$$

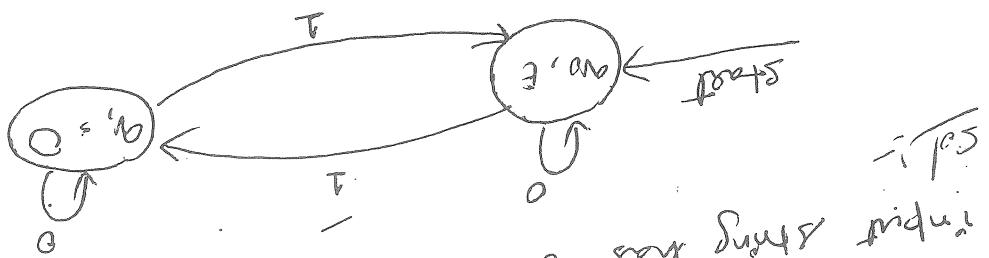
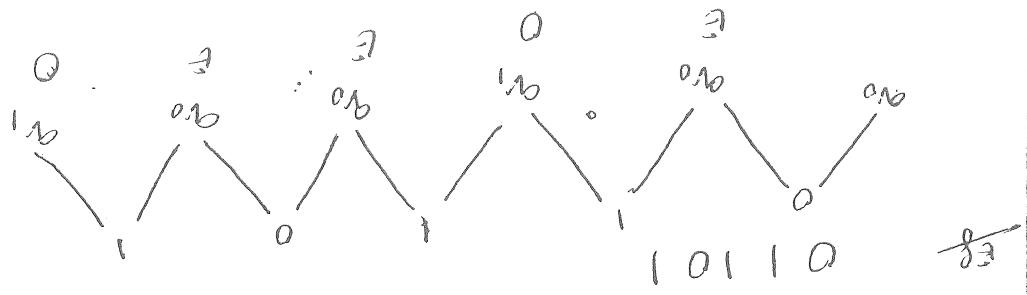
$$q_0 = \text{start state}$$

→ Output a lower pattern string a, b, a, b if it is not present, a, b, a, b is present.

← Mealy machine output 1 whenever the pattern abba



Change the fsm with suffix Mealy machine.



→ Consider a more machine which accepts strings
that ends with even number of 1's
and is output even other than the P/F splitting
input string has odd number of 1's
as and is output even other than the P/F splitting

Format: $Q \rightarrow \text{finite set of } 8 \text{ states}$
 $Q \rightarrow \text{set of } 8/16 alphabet}$
 $Q \times S \rightarrow \text{transition function } (Q \times S \rightarrow Q)$
 $Q \rightarrow \text{output function } (Q \rightarrow T)$
Input strings has odd number of 1's

called machine
called machine M = (Q, S, T, δ, q_0)

Machine $\frac{\text{output}}{\text{format}}$ associated with each state is

$$(\omega)_T = (\omega)_T$$

July 1987

Q. 3. `classes = new class`
 Q. 3. `classes = new class`

If there are any states P and Q such that there is no any character C such that when C is used, P goes to the element of classes and Q goes to another then P & Q must be split. The rest of these classes into new classes. If the union of classes P do not do well, we split them up.

for each state q_i in \mathcal{E} do:
 for each character c in Σ do:
 $\text{Determine which element of } \mathcal{C}$
 $\text{leads to } q_i \text{ by } c$

9.2 For each equivalence class \equiv_m choose:
9.3 For each equivalence class \equiv_n choose:
9.4 If e contains more than one state, choose:
9.5 If e needs to be split.

Type II could a pass at which no change to class
has been made:

$$1. \text{ Classes} = \{A_1, A_2, A_3\}$$

$$\text{min } Dfsm(m : Dfsm) =$$

Algorithm for minimum spanning tree

$\{ (s \in \{k, l, s\}) : \alpha \in \delta_s \} = \{ (s \in \{k, l, s\}) : \alpha \in \delta_s \}$
 $A' = \{ \alpha \in k : \alpha \neq \phi \}$
 $s_i = \delta_{ps}(s)$
 In, contains one state for each demand of $P(k)$.
 So $m' = (k, l, s, A')$ looks
 like m .
 Proof:- Proof is by construction of an equivalent
 NDFSM m' . The states of m' corresponds to
 states of m .
 Equivalently m' form that accept to L . -
 accept same language L . These exists an
 equivalent on. NDFSM $m = (k, l, s, A)$ that
 Therefore

For every DFM there is an equivalent NDFSM.
 Proof:- Let m be a DFM that accepts some
 language L . m is also an NDFSM that
 happens to contain no e-transitions and
 no loops that goes to a
 transition s such that $\delta_s(s) = s$.
 So the NDFSM except s simply
 has no transitions from s to any other
 state. So s is an acceptor state.
 Therefore

Conversion from NDfsm to Dfsm :-

Algorithm to Compute w , given in [NDfsm to Dfsm]

ndfsm to d fsm ($M : NDFSM$) =

Compute $\epsilon ps(a)$

1. for each state a in A do :

2. $s_i = \epsilon ps(s)$

3. Compute s_i

4. $s_i = \epsilon ps(s)$

5. $\phi = s_i$ (b)

a) active-state - $\{s_i\}$

c) While there exist some element a of active states for which s_i has not yet computed do:

for each character c in C do :

new-state = ϕ

for each state a in A do

new-state = new-state \cup $\epsilon ps(p)$.

Add the transition $(a, c, new-state)$ to S ,

If new-state \neq active-state then insert it into active-states.

4. $x_1 = \text{active-state}$

5. $A = \{x_1 : A \neq \emptyset\}$

26
Kings

- To categorise the problems that are easy to solve and those that are not, a unifying frame work is to be defined to which only computational problem can be cast.
- That frame work is to be defined to which only computational problem can be cast.
- Let L be the definition of language and to that frame work is to language recognition.
- Let L be the definition of language and to be the starting. Then "Is w in L ?" question is answered with either yes/no; which is defined as decision problem.
- * Problems that are already stated ~~as~~ decision problems can be divided into a major categories.
- * Problems that are already stated ~~as~~ decision problems.
- To solve these kind of problems encode the input as strings, define a language that contains exactly the set of inputs for which the answer is yes for "Is w in L ".
- The answer is yes for "Is w in L " if the input encodes it as a language problem and then encode it as a language problem.
- * Problems that place not related as decision problems.
- To solve these kind of problems encode the input as strings, define a language that contains exactly the set of inputs for which the answer is yes for "Is w in L ".
- The answer is yes for "Is w in L " if the input encodes it as a language problem and then encode it as a language problem.
- * First problem encode the problem as a decision problem.
- First problem encode the problem as a decision problem and then encode it as a language problem.
- * Given a search starting w and a web document containing an appropriate starting encoding;
- Given a search starting w and a web document
 - do they match?
 - (a) Should a search engine on input w , consider returning d ?
- $L = \{ \langle w, d \rangle : d \text{ is a candidate match for the query } w \}$

A Language Hierarchy:

Ans. We encode directed graph

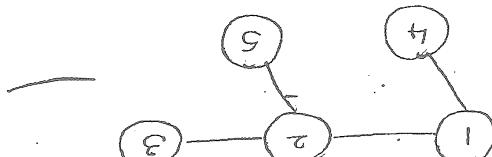
$$L = \{w \in \{0, 1\}^n : w = n_1 n_2 \dots n_r \text{ where } n_i \in \{0, 1\}\}$$

Laudaquare

for the above graph, there are 5 vertices & the edges are represented as

$$E(1, 2) = \{<110>\}$$

e.g., $V = \{(1), (10), (11), (100), (101)\}$
 Part of vertex joining & vertex
 number and E be the edge represented as

Let V be the vertices represented as binary


Let G be the graph represented as follows:
 Given an undirected graph, is it connected?
 3. Graph connectivity:

$$w = 2 + 4 = 10 \notin L.$$

$$\text{e.g., if } w = 2 + 4 = 6 ; 23 + 47 = 70 \in L.$$

$$L = \{w \mid w = <1> + <12> = <13>; w \in \{0, \dots, 9\}^*\}$$

2. To decide whether sum of 2 integers belongs to the integer.

4. Given a protein fragment f and a protein molecule p , could f be a fragment of p ? Represent each protein molecule as a sequence of amino acids a_1, a_2, \dots, a_n . Each a_i is represented as a character, now compare f and p as sequences of characters. If f is a substring of p , then f is a fragment of p . This can be transformed into decision question. Problems that are not already stated as decision questions, can be transformed into decision questions. The main idea is to use language recognition problems. The main idea is to encode into a single string, both the input and output of the original problem. P.

Casting problems as Decision question

Given two non-negative integers, compute their sum. E.g.: Addition as decision. Given two numbers is the sum of first two. E.g.: Transform the problem of adding a numbers into the problem of checking whether the third encoding: Transform the problem of adding a numbers into the problem of checking whether the third integer sum = $\{n \text{ of the form: } i_1 + i_2 = i_3 \text{ where } i_1, i_2, i_3 \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}\}$

Given list of integers, sort it. E.g.: Sorting as decision. Given this list of integers, sort it. Encoding: Transform the problem of sorting a list into problem of examining a pair of list to decide whether the second corresponds to the softed version of the list.

$$i_3 = i_1 + i_2$$

$$\{i_1, i_2, i_3 \mid i_1, i_2, i_3 \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}\}$$

Integersum = $\{n \text{ of the form: } i_1 + i_2 = i_3 \text{ where } i_1, i_2, i_3 \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}\}$

number is the sum of first two.

into the problem of checking whether the third

encoding: Transform the problem of adding a numbers

Given two non-negative integers, compute their sum. E.g.: Addition as decision.

of the original problem. P.

language recognition problems. The main idea is to encode into a single string, both the input and output of the original problem. P.

Given two non-negative integers, compute their sum. E.g.: Addition as decision.

Given two numbers is the sum of first two. E.g.: Transform the problem of adding a numbers into the problem of checking whether the third

encoding: Transform the problem of adding a numbers into the problem of checking whether the third integer sum = $\{n \text{ of the form: } i_1 + i_2 = i_3 \text{ where } i_1, i_2, i_3 \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}\}$

whether f is a substring of p .

problems that are not already stated as decision questions, can be transformed into decision questions. The main idea is to use language recognition problems. The main idea is to encode into a single string, both the input and output of the original problem. P.

$L = \{f, p\} \neq$ could be fragment from $p\}$.

Given a protein fragment f and a protein molecule p , could f be a fragment of p ? Represent each protein molecule as a sequence of amino acids a_1, a_2, \dots, a_n . Each a_i is represented as a character, now compare f and p as sequences of characters. If f is a substring of p , then f is a fragment of p . This can be transformed into decision question.

$a = \emptyset$

$q = (\text{select name age} = a)$

$d = (\text{name, age, phone})$, (J, 23, 564), (M, 24, 284)

or to $d \}$

a is the correct result of applying

q is a string representing a query

$L = \{ d \# a : d \in \text{an encoding of a database},$

is correct.

Encoding: Transferring the task of executing the query
into the problem of evaluating a query to see if it

againsst the database.

Given a database and a query, execute the query

e.g.: Database querying as decision.

if the string is in L . String is not in L .

$w_2 = 1, 3, 5, 6, 9$

$w_1 = 1, 2, 3, 4, 5, 6, 7$

e.g.: $w_1 = 1, 5, 3, 9, 6$

w_2 is of the form l_1, l_2, \dots, l_n and w_2 contains
same object as w_1 , and w_2 is sorted

$L = \{ w_1 \# w_2 : \exists n \leq 1 (w_1 \text{ is of the form } l_1, l_2, \dots, l_n,$

Machine Based Hierarchy of language:

The computational models allows to write the programs in such a way that it accepts some language. Hierarchy of computational model is defined as follows:

1. ^{is} model is very simple, programs written for it is easy to understand, they run in linear time and algorithms exist to answer almost any question about such programs.
2. nd model is more powerful, but still limited. - finite state machine.
3. rd model is powerful enough to describe anything that can be computed by any sort of real computer.

- Turing machine.
The classes of languages that can be accepted by

finite state machine is called regular language.

shown in figure. FSM has a start state, with an unlabeled arrow leading to it. FSM states in its start state, if it accepts each character. Machine accepting is if it reads changes state based on the transitions as it reads

each character. After reading last character from s, otherwise leave it accepting state as marked with double circle under each character. If it reads characters changes state based on the transitions as it reads after reading last character from s, otherwise leave it



FSM states in its start state, FSM

a,b.

3

2

1

b

a

b

a

b

a

b

a

b

a

b

a

b

a

b

a

b

a

b

Context-Free Language

language accepted by push down Automata [PDA].

is context-free language.

[FSM] finite state machine has the drawback of counting the no. of characters it reads as it has the memory in the states.

eg $L = \{a^n b^n | n \geq 0\}$ in this the language accepted by L is all a's come first, # number of a's = number of b's.

trying to build fsm for the above language is not possible as there is no bound on no of a's.

Machine that consists of fsm and a stack of tokens to called PDA are designed to implement the base language.

Simple PDA that accepts language.

an b^n is as shown in figure:

As it reads the characters it pushes it on to the stack.

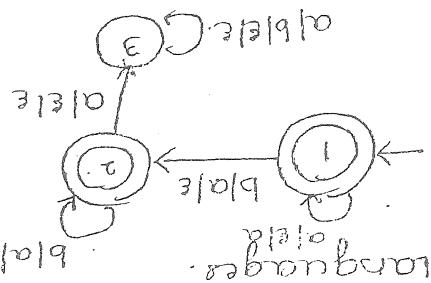
Transitions on PDA is based on next character to be read, top of the stack & the current state.

and hence up languages are context free.

most of the programming languages, unary language and decidable and semi-decidable language.

thus building fsm or pda for those languages is not possible.

The resulting machine which accepts the language is Turing machine.

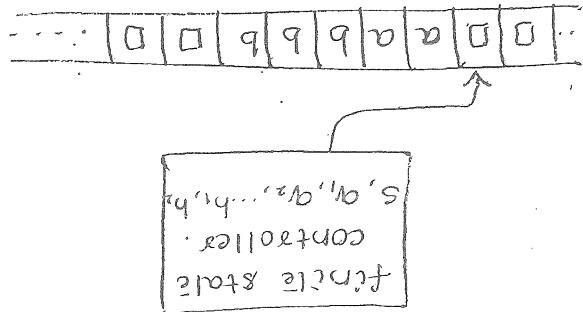


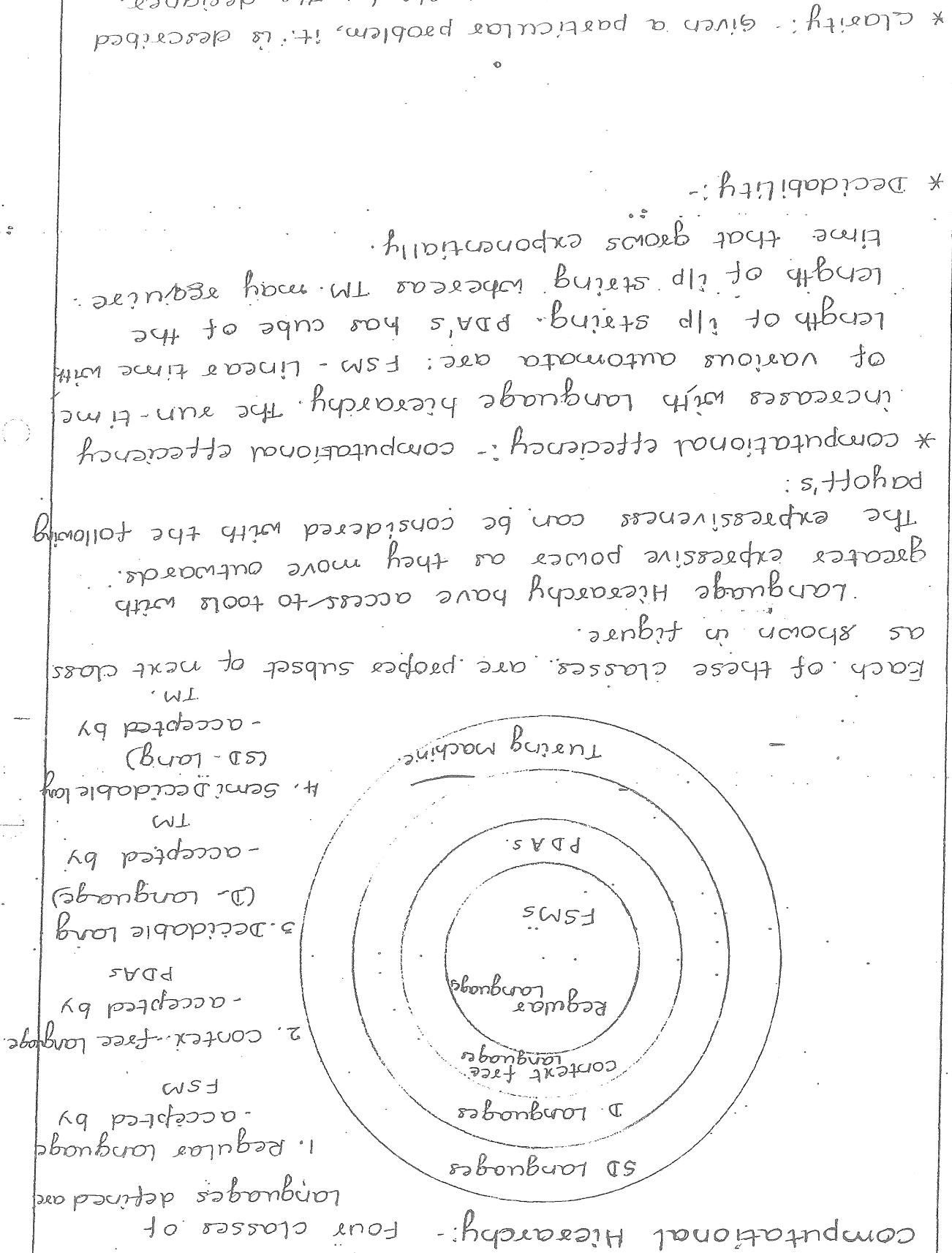
The structure of Turing machine [TM] is as follows in figure.

read/write head position at beginning location of the input string. At each step TM considers the state and the character that is on the tape pointing by read/write head. Read/write head can move by one direction or it can move back and forth many times/forever.

Turing machine can be used to define two new classes of languages

- * language L is decidable if there exists a Turing machine M that accepts all strings that are in L and fails to accept every string that is not in L. (given a string that is not in L, M may accept or it may loop forever).
- * language L is semi-decidable if there exists a Turing machine M that accepts all strings that are in L and fails to say yes but may fail to say no.





- P ⊂ NP ⊂ PSPACE
- * PSPACE, which contain those languages that can be decided by a machine whose space requirement grows as some polynomial function of the length of the input.
 - * NP, contains those language that can be decided by a non-deterministic machine with the property that the amount of time required to explore one sequence of guesses grows as some polynomial function of the length of the input.
 - * P, which contains those language that can be decided in time that grows as some polynomial function of the length of the input.
 - M.

Decidability Hierarchy of language classes:
Based on the resources [Time, Space] required by

- * Rule of Least Power: use the least powerful language of decidable & semi-decidable language.
- * No correspondence tools exist for broader class of suitable for expressing information, constraints on programs on word.
- * Regular expression / regular language - FSA.
- * Context free grammar / context-free language - PDA.
- * Rule of least power: use the least powerful language.
- * Rule of least power: use the least powerful language.

COMPUTATION:-

- Key ideas for computations are:
- * Decision procedures
- * Functions of language
- * Non-determinism
- * Decision procedures
- A decision procedure is an algorithm to solve decision problem. A decision procedure must be guaranteed to halt on all inputs. * Programs must be correct for all inputs. It must be the correct answer for given input.
- Example - I: check for even numbers.
- If $(x/2) * 2 = x$ then return true.
else FALSE.
- The procedure is decidable as it answers yes/no for given input x.
- Example - II: check for prime numbers
- To prove loop terminates.
- Procedure (α ; positive integer)
- For ($i=2$ to $\text{ceiling}(\sqrt{\alpha})$) do
 - while $(\alpha/i) * i = \alpha$ then return false
 - return true

procedure checks whether the given number is prime or not. It is decidable.

where $\text{ceiling}(\sqrt{\alpha}) = \lceil \sqrt{\alpha} \rceil$

Return true.

Example - 3 : Fermat numbers are prime Fermat numbers.
 Fermat number is a sequence defined as

$$F_n = 2^{2^n} + 1, n \geq 0$$

$$F_0 = 3, F_1 = 5, F_2 = 17, F_3 = 257, \dots$$

Example - 4 : Check for small prime Fermat numbers.
 Fermat number is guaranteed to have as the value of procedure is square root of candidate value which exceeds 1,000,000.
 Given a program P that takes a string w as an input and the calculation of p on w
 halton (P; program, w; string) =
 1. Simulate the execution of p on w
 2. If simulation halts return True
 halton (P; program, w) =
 if parameter, does P halt on some value of w?
 If P halts, does P halt on some value of w?
 this kind of procedure is called semidecision procedure.

non-deterministic.

In both of the cases of choose(), the function is

private.

determining that no elements of S that satisfy

fail to halt if there is no mechanism for

saturation.

that all elements of x of S , $p(x)$ is not

halt and return false if it can be determined

* else choose() will

with a value other than false, if there is one

* return some element x of S such that $p(x)$ holds

S may be finite or infinite. choose() will

The choose() is presented with set of S values where

* choose(x from S ; $p(x)$)

halt.

fail to halt if any of the actions fail to

halt and return false

* else choose() will

* return some success value, if there is one

success value of the value false. choose() will

of alternatives, each of which will return either a

The function choose is presented with a finite set

action n)

action 2;

* choose.(action 1;

Determinism and nondeterminism; consider function

Module-a

Regular Expressions (RE): what is a RE?, Kleene's theorem,
Applications of REs, Manipulating and Simplifying REs. Regular
Grammars: Definition, Regular Grammars and Regular Languages.
Regular Languages (RL) and Nonregular Languages: How many RLs, To
show that a language is regular, Closure properties of RLs, to show some
languages are not RLs.

Textbook 1: Ch 6, 7, 8: 6.1 to 6.4, 7.1, 7.2, 8.1 to 8.4

MODULE 2

~~Notes~~

The language accepted by DFA or NFA and C-NFA is called regular language. A regular language can be described using regular expressions consisting of the symbols such as ;, +, and *.

The three operators used to obtain a regular expression are : +, * and .

The regular expressions can also be denoted by Σ^* .

What is a regular expression?

A regular expression can be formally defined as follows:

ϕ is a regular expression denoting an empty language.

$L(\phi) = \emptyset$

$L(\epsilon) = \{\epsilon\}$

$L(E)$ is a regular expression indicating an empty string.

For a, belonging to the alphabet Σ , we have $a \in \Sigma$.

and $L(a) = \{a\}$ is the language.

Every element in Σ is an regular expression involving the denoted by $L(\phi) = \emptyset$.

The language generated by an empty set $\{\phi\}$ is a regular expression denoting an empty language.

$L(E)$ is a regular expression involving the language containing an empty string.

Let E be a regular expression involving the language containing every element in Σ .

Note:- The symbols () and . can be used in regular expressions. The idea of endowment of the regular expressions is determined by the parentheses and the priority associated with the operator.

Let us consider the following examples:

Union operator (+ operator)

Circumflex operator (^ operator)

Dot operator (operator)

The three operators used to obtain a regular expression are : +, * and .

The regular expressions can also be denoted by Σ^* .

What are the operators used to obtain a regular expression?

The three operators used to obtain a regular expression are : +, * and .

The regular expressions can also be denoted by Σ^* .

What is a regular expression?

A regular expression can be formally defined as follows:

ϕ is a regular expression denoting an empty language.

$L(\phi) = \emptyset$

$L(\epsilon) = \{\epsilon\}$

$L(E)$ is a regular expression indicating an empty string.

For a, belonging to the alphabet Σ , we have $a \in \Sigma$.

and $L(a) = \{a\}$ is the language.

Module - 2

Regular Expressions

RASHMI M
Asst. Professor
Dept. of CE
RNSIT

$(a+b)^*$

$(a+b)$

a^*

a^*

Regula^r expression
Expre^ssion

Meaning

The following rule shows some examples of regular expressions and the language corresponding to these RE.

Note:- The expression obtained by applying any of the rules from 1 to 4 are RE.

R_1 is a RE corresponding to the language $L(R_1)$

$$L(R_1) \cdot L(R_2)$$

$R_1 \cdot R_2$ is a RE corresponding to the language $L(R_1) \cup L(R_2)$

$R_1 + R_2$ is a RE corresponding to the language the language $L(R_1)$ and $L(R_2)$, then

$\Rightarrow R_1$ and R_2 are two regular expression denoting same RE.

\Rightarrow If R_1 and R_2 are two REs, then $R_1 \cup R_2$ is also

set of strings of a^* and b^* of any length including the Null string

String consisting of atleast one a or one b

(one or more a^*)

String consisting of atleast one a

(zero or more a^*)

String consisting of any number of a^* .

It's and is ending with any number.

String counting is a step followed by any number of steps including possibly many more steps.

for safety consulting of even numbers

of life changes by its end, and is ending with
the consequence of death.

Set of strings consisting of even numbers is followed by odd numbers by sets of strings.

get by staying constantly at least one a
followed by staying constantly at least one c

for θ satisfying a
 a 's and b 's having

end of strings of a_1 's and b_1 's together with the string ab
stacking up together

enduring
Safaris by air and by boat with the safari guides

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

$$*(a+b) ab$$

$$99^a * (a+b)$$

* a * b * c *

a⁺ b⁺ c⁺

20 * (1 + 0)

(1)

1 + 10

*1 * (a1 + a)

4) Obtain a regular expression representing strings of a 's and b 's having even length.

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

Ans

Goal: RE is $(e+a+b)^*$
 Length ≤ 10

Obtain RE representing strings of a 's and b 's of length 2

$$RE \text{ is } (e+a+b)^2$$

(Ans)

So, the RE is $(e+a+b)(e+a+b)$
 e, a, b, aa, ab, ba and bb
 can be written as :

Goal:- strings of a 's and b 's whose length is ≤ 2
 length ≤ 2

Obtain RE representing strings of a 's and b 's

$$RE \text{ is } (a+b)(a+b)$$

Ans

So, RE is : $(aa + ab + ba + bb)$
 aa, ab, ba and bb
 strings of a 's and b 's with length two are :

Goal:-
 a 's and b 's drawing length 2.

Obtain a regular expression representing strings of a 's and b 's drawing length 2.

g) obtain a RE for the language $L = \{a^m b^n | m+n \text{ is even}\}$

sol: $(a+b+c)^* b (a+b+c)^* a (a+b+c)^* b (a+b+c)^* + (a+b+c)^* a (a+b+c)^* a (a+b+c)^*$
 q) obtain a RE representing strings of a 's and b 's ending with b , and having no substring aa .

at least one a and atleast one b where $\Sigma = \{a, b\}$

sol: $(E+a)(b^*)^*(E+b)$

q) obtain a RE representing a language consisting of strings of a 's and b 's where all a 's and b 's are in the same conjugate with a 's and b 's.

sol: $(a+b)(a+b)(a+b)$
 q) obtain a regular expression representing strings of a 's and b 's having odd length.

sol: $a^m b^n | m \text{ and } n \text{ are odd numbers}$

case 1: $((m+n) * 2 + 1) * = ((m+n)* 2 + 1) * a (a)$
 also given that $(m+n)$ is even. This result is in even no. of symbols.
 so if it is given that -
 in a sequence of numbers
 case 2: $((m+n) * 2) * = ((m+n)* 2) * b (b)$
 also given that $(m+n)$ is even no. of symbols.
 so if it is given that -
 in a sequence of numbers
 even numbers
 RE = $a (a) * b (b) *$
 number of b 's result in even number of symbols.

case 2: $[m+n] \text{ is even} \Rightarrow$ odd numbers followed by odd numbers
 RE = $(a) * (b) * + a (a) * b (b) *$

$$(01+1) * 00 (10+1) + (0+0) * (10+1) + 1$$

above RE :

so, the complete RE is obtained by adding all the

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

Case 3 :- Two consecutive digits preceded by any combination of 1's and 0's and followed by 1's and 0's and is given by :

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

above regular expression and can be written as : we can have an optional 0 at the end of the

$$(1+0)*$$

Case 2 :- No two consecutive digits as can be obtained by any combination -

$$1*$$

Case 1 :- No three which may differ. Any number of following cases :-

$$1*$$

the string containing at most two 0's will have the following cases :-

Q1 :- Observe that the string may have any number of 0's. It is width no de as as width one or as width two 0's.

Consecutive 0's.

Q2) Obtain a RE representing a language consisting of strings of 0's and 1's with at most one pair of

$$\boxed{(\alpha+1)(\alpha+1) \cdot (\alpha+\beta) + (\alpha+\beta) * (\alpha+1)(\alpha+\beta)}$$

(8)

$$(\alpha+\beta) * (\alpha+1)(\alpha+\beta)$$

$$(\alpha+\beta) * (\alpha+1)(\alpha+\beta)$$

The above expression can also be written as follows -

~~Re-writing by simplifying~~

$$(\alpha+\beta) * (\alpha+1)(\alpha+\beta)$$

The above expression can be preceded by an apostrophe and followed by an apostrophe;

$$(\alpha+1)$$

Ques: No two consecutive zeros can be obtained using;

1. ~~Obtain a RE representing strings of 0's and 1's~~

$$\boxed{* (\alpha+1) * 000 * (\alpha+1)}$$

If 0's written as -

$$* (\alpha+1)$$

The string 000 can be preceded by 0's and followed by an apostrophe which is suppressed by the regular expression;

000

below:

Ques: The RE representing 3 consecutive zeros is shown

(i) Obtain a RE for the language $L(R) = \{m/n \in \{0, 1\}^* \mid m \neq n\}$ with atleast three consecutive 0's.

Goal :- The RE whole length is multiple of 3 can be obtained using $(a+b)(a+b)$

Goal :- The RE whole length is even can be obtained using $(a+b)(a+b)$

Goal :- Whole length is odd when a RE self-evaluating string of a 's and b 's is obtained

$$RE \text{ is } a (a+b)^* + b (a+b)^*$$

Goal :- Whole length is even when a RE self-evaluating string of a 's and b 's is obtained

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

any combination of a 's and b 's

But, the third symbol from the right can start with a

The second symbol from the right end should be a

Goal :- The first symbol from the right end can be a b

Whole second symbol from the right end is a

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

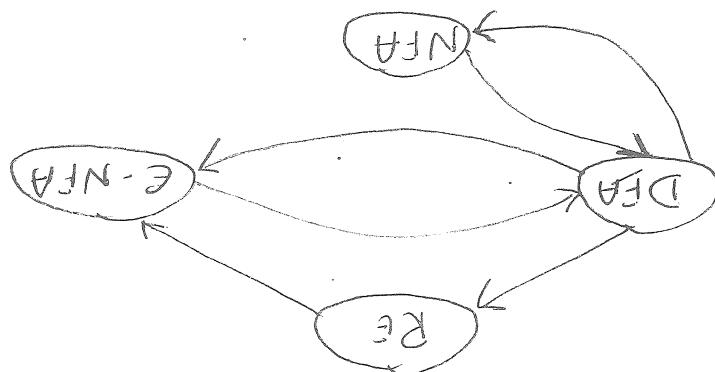
Starting with a and ending with b

$$\therefore RE = ((a+b)(a+b) * + (a+b)(a+b) *$$

To build a FSM from Regular Expression:

Theorem: Let R be a regular expression. Then there exists a FSM $M = (Q, \Sigma, S_0, F)$ which accepts $L(R)$.

Proof that there exists a finite automaton to accept the language $L(R)$ corresponding to the regular expression R .



The relation between FSM and regular expression (RE) is shown pictorially below:

any language defined by RE can be accepted by some finite automaton that can be accepted by a finite automaton can be defined by some finite automaton that can be accepted by some regular expression.

\hookleftarrow The languages that can be defined using RE are regular languages.

\hookleftarrow The RE are useful tools for defining patterns.

The RE are very significant for two reasons:

Kleene's Theorem:

Proof: By definition, ϕ , e and a are regular expressions. So, the cross-pending machines to recognize these languages for the specific expressions are shown below:

In the definition of a regular expression if R is the schematic representation of a regular expression ϕ, e and a then the start state and f is the final state of machine M to accept the language $L(R)$ is shown below. Where q_1 is the start state and q_2 is an equivalent expression ϕ, e and a . It is clear that if R and S are regular expressions which clearly uses these operations \cup , $*$ and * and $L(R)$ be a regular expression ϕ, e and a respectively and $L(S)$ be a regular expression ϕ, e and a respectively and $L(R) \cap L(S)$ be a regular expression ϕ, e and a respectively and $L(R) \cup L(S)$ be a regular expression ϕ, e and a respectively and $L(R)^*$ be a regular expression ϕ, e and a respectively.

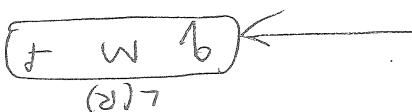


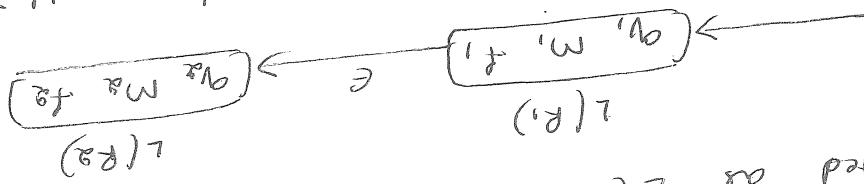
Fig: E-NFA's to accept ϕ, e and a



Proof: By definition, ϕ , e and a are regular expressions. So, the cross-pending machines to recognize the language for the specific expressions are shown below:

Below:

Case 1: $R = R_1 + R_2$: we can construct an NFA which accepts either $L(R_1)$ or $L(R_2)$ which can be superimposed to form $L(R_1 \cup R_2)$. This, in which the combined machine M and state q_f of machine M_2 , becomes the final state of machine M and the start state of machine M_1 , becomes the start state of M and the start state of machine M_2 , becomes the final state of machine M and the start state of machine M_1 . Thus, in which In state q_2 upon accepting $L(R_2)$, the machine outputs to form $L(R_1 \cup R_2)$. Thus, in which the combined machine M moves from state q_1 to state q_2 because the start state of machine M_1 moves to state q_2 which is the final state of machine M_2 .

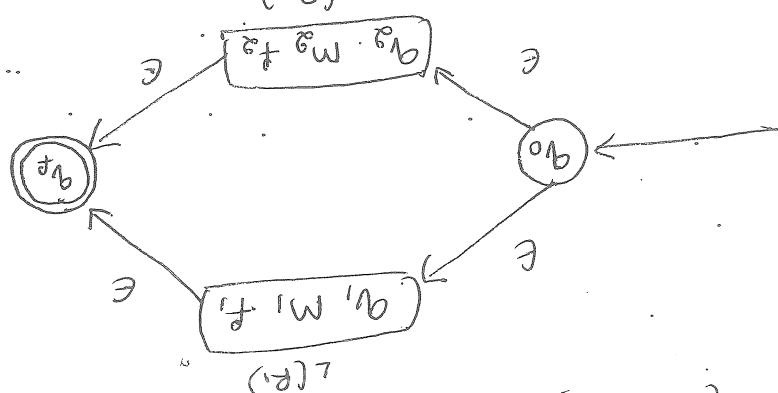


Case 2: $R = R_1 \cdot R_2$. we can construct an NFA which accepts $L(R_1) \cdot L(R_2)$ as shown below:

If q_1 is a dead from the above figure that the machine can be superimposed as $L(R_1 \cdot R_2)$ as shown below:

If q_1 is a dead from the above figure that the machine can be superimposed as $L(R_1 \cdot R_2)$ as shown below:

If q_1 is a dead from the above figure that the machine can be superimposed as $L(R_1 \cdot R_2)$ as shown below:

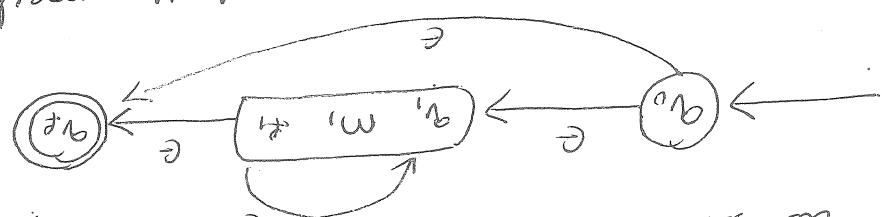


Case 2: $R = R_1 \cdot R_2$. we can construct an NFA which accepts $L(R_1) \cdot L(R_2)$ as shown below:

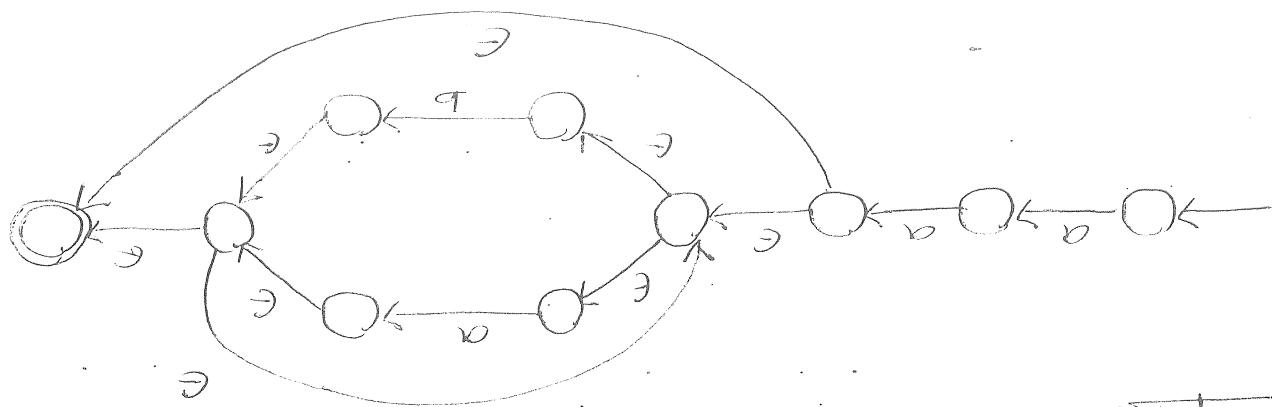
If q_1 is a dead from the above figure that the machine can be superimposed as $L(R_1 \cdot R_2)$ as shown below:

Case 3 : $R = (R_1)^*$

We can consult an NFA which accepts $L(R_1)^*$ as shown in fig.



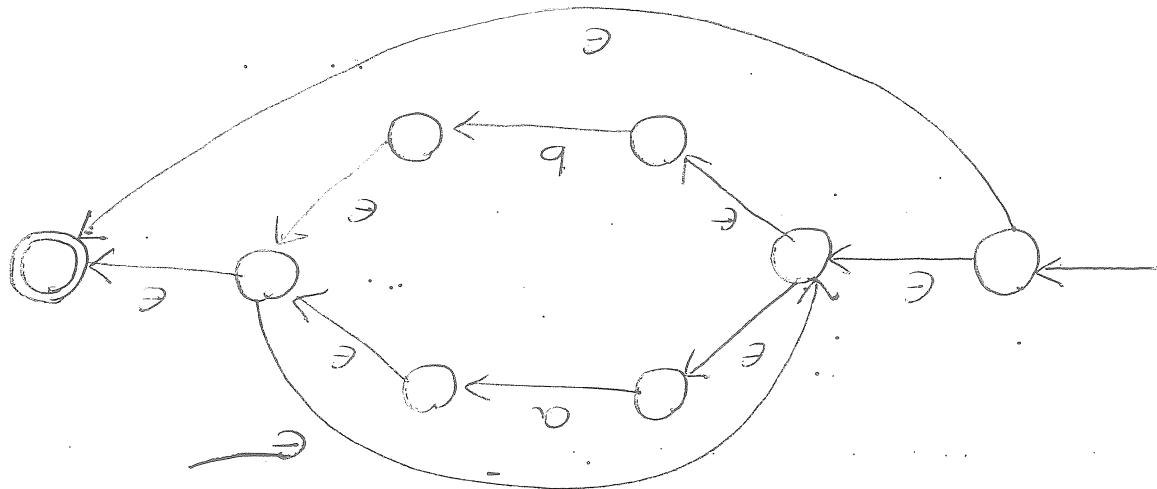
It is clear from the fig. that the machine can either accept or any number of $L(R_1)$'s thus accepting the language $L(R_1)^*$. Here, as to the state q_0 and the final state q_f the final state q_f .



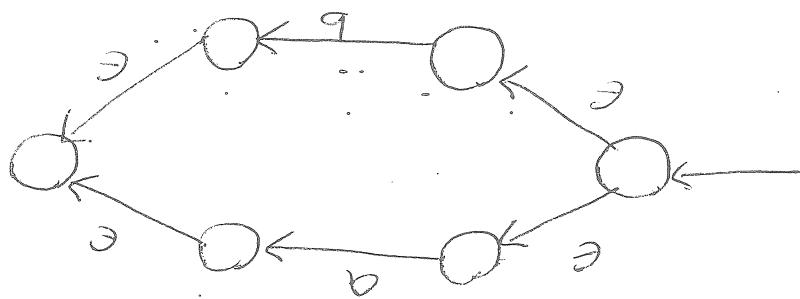
Step 4: The machine to accept $aa(a+b)^*$ is shown:



Step 3: The machine to accept aa is shown:

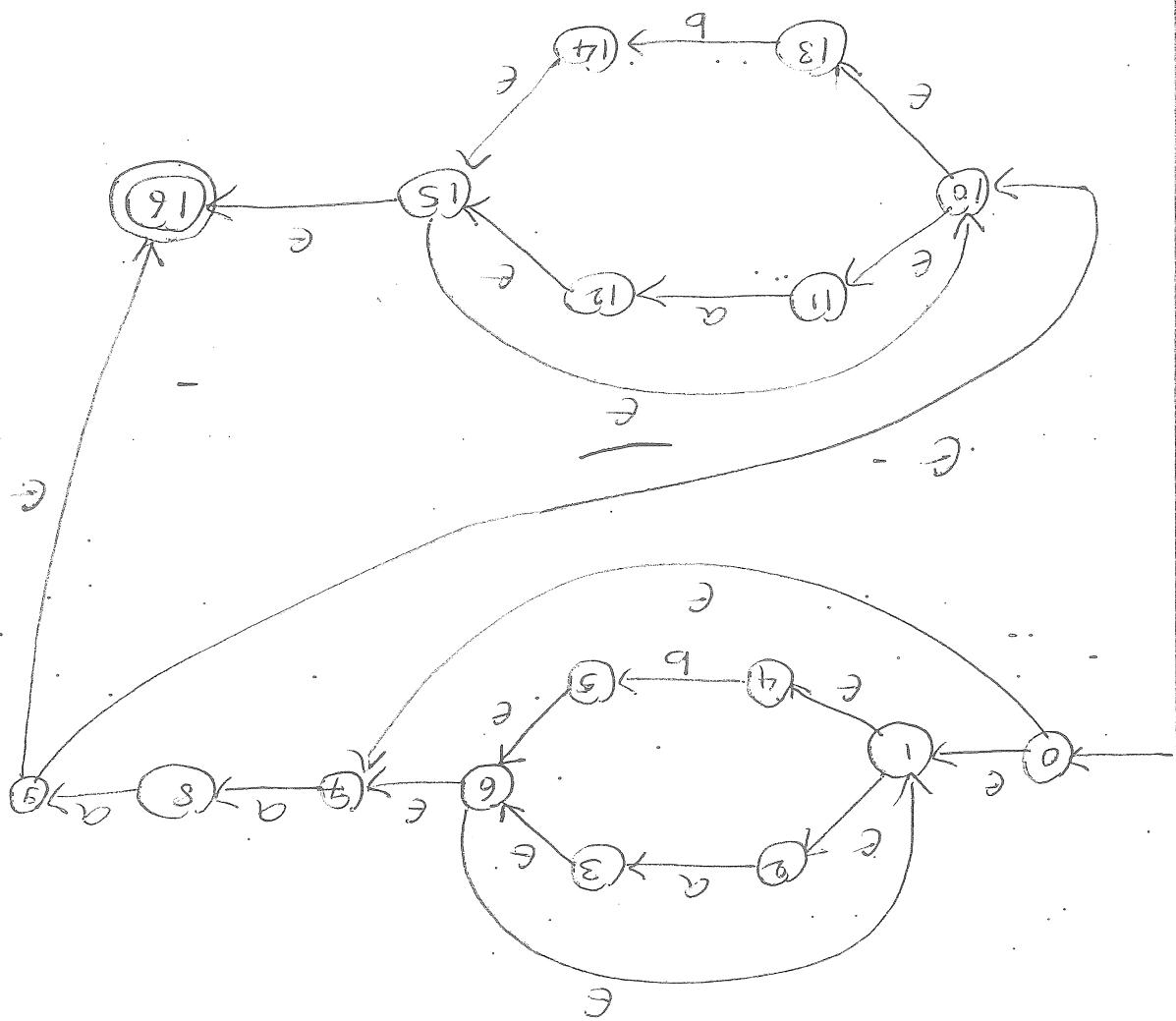


Step 2: The machine to accept $(a+b)^*$ is:

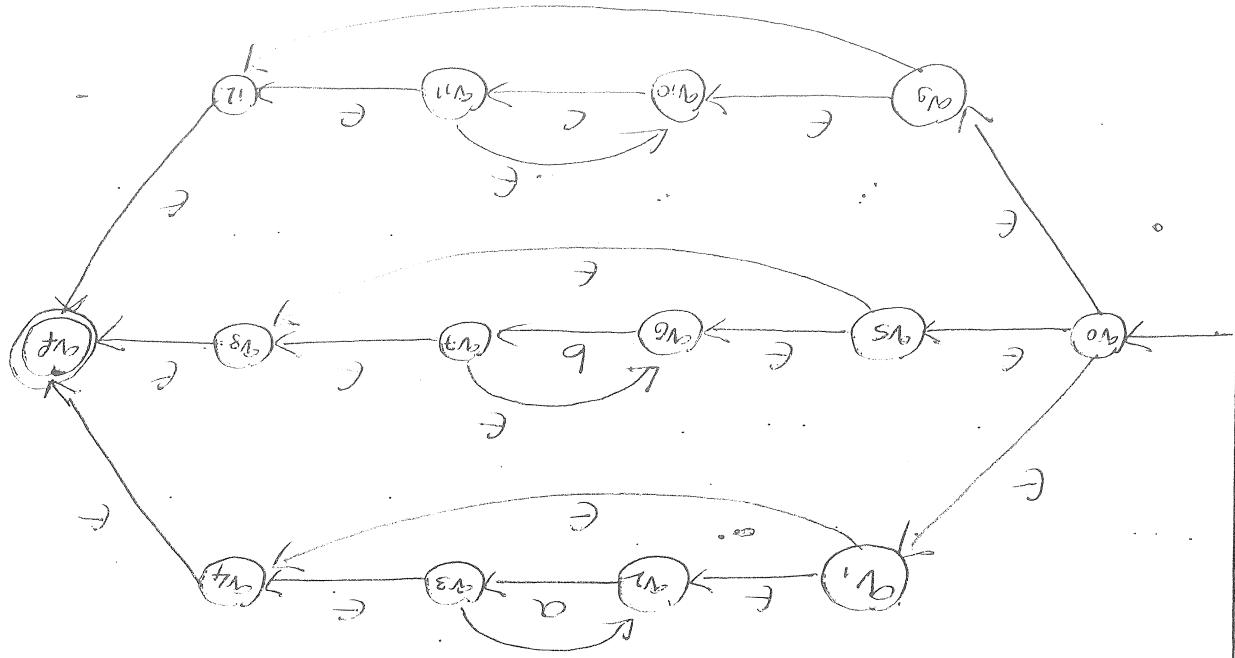


Step 1: The machine to accept $(a+b)$ is:

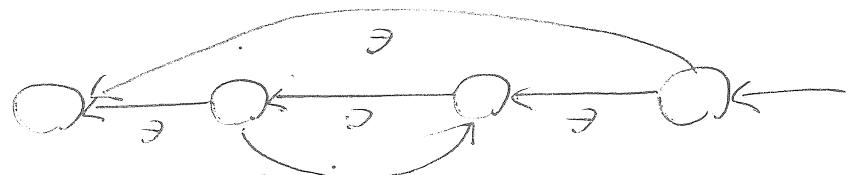
(a) Draw an NFA for the RE $(a+b)^*aa(a+b)^*$



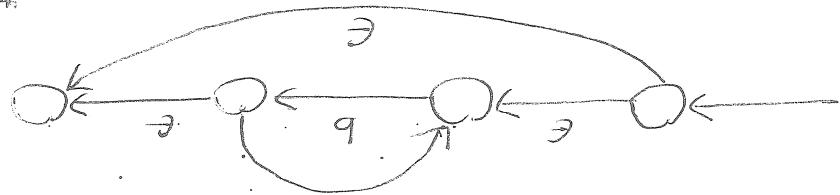
Step 5: The machine to accept $(a+b)^* \cdot aa \cdot (a+b)^*$



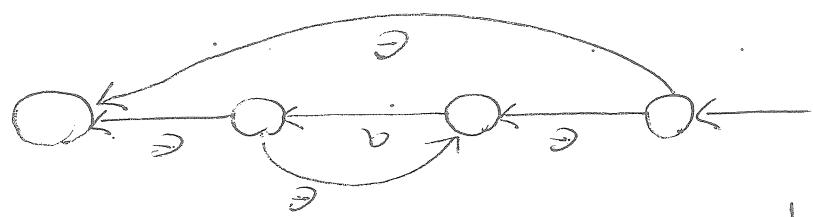
The machine corresponding to the RE $a^* b^* c^*$ can be shown below:



The machine corresponding to the RE c^* can be shown as

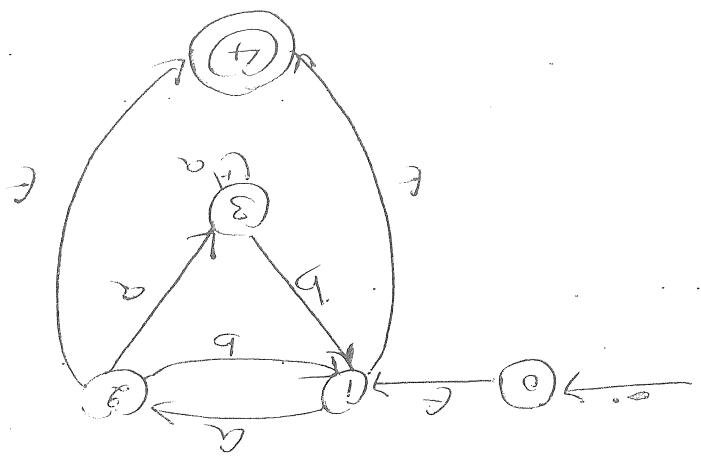


The machine corresponding to the RE b^* can be shown as

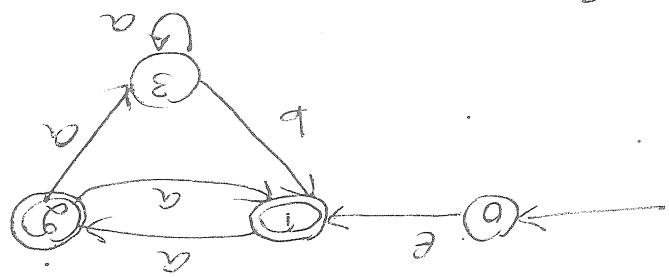


So: The machine corresponding to the regular expression $a^* b^* c^*$ can be written as

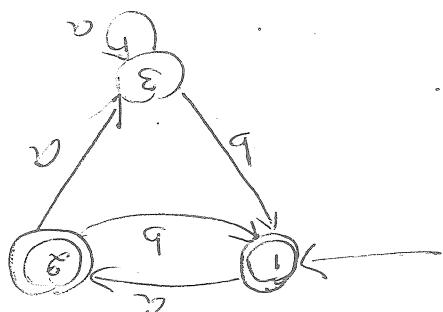
(a) Draw an NFA for the regular expression $a^* b^* c^*$



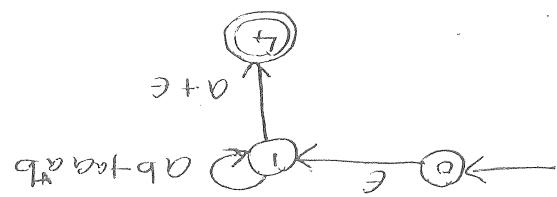
Since there are three final states, we choose
a new final state 4 and update E-transitions
from state 1 and 3 to this final state 4 as
shown below:



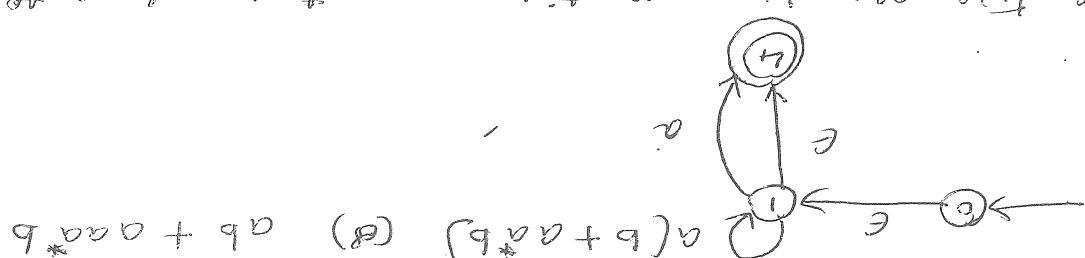
So:- Step 1 is the first step and it has
some incoming edges. So, if we have a new
state, apart from the start state 1 as transition as
shown below:



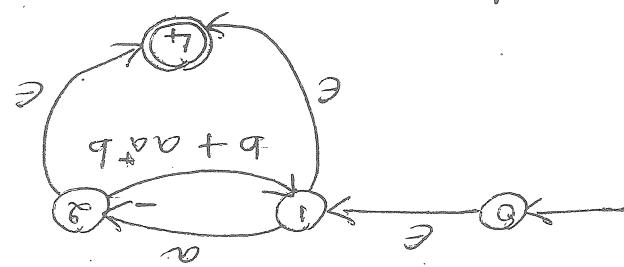
Following fsm
4) Build a Regular expression from the



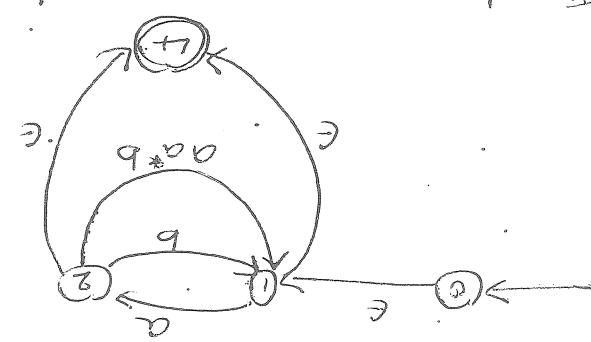
The two arcs from state 1 to state 4 represent
an optional a can be written as shown below:



The state 2 can be removed by having a self-loop
at state 1 with regular expression $a(b+a^*b)$ as
shown below:



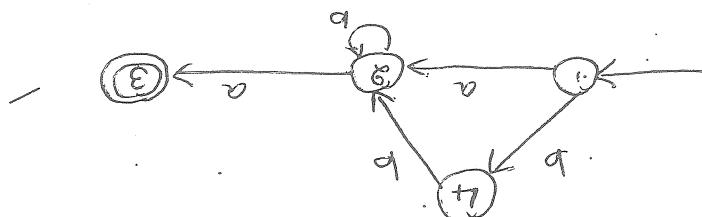
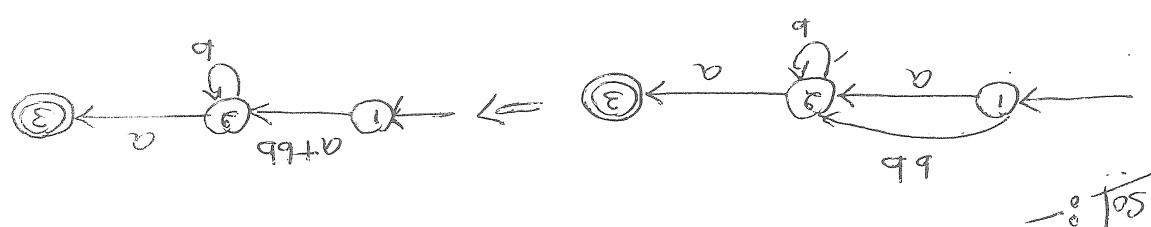
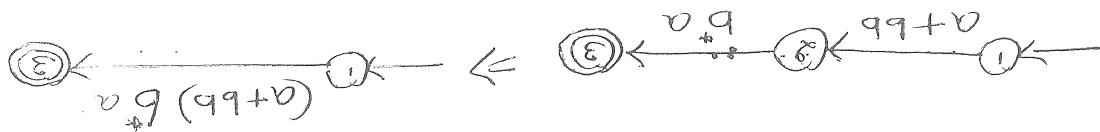
The two arcs from state 2 to state 1 can be
written using one arc



The label a^*b as shown:
a transition from state 2 to state 1 with
The state 3 can be removed by inserting

$$(a + bb)^*$$

So, the final RE representing the language accepted by PSM is :

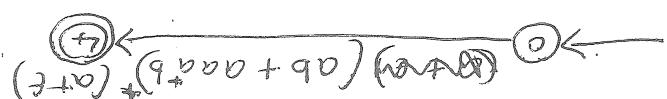


(Q2) Build a Regular expression from the following form

$$(ab \cup aaaa)^* (a \cup e)$$

By replacing the operators $+$, \cdot , \cup with union symbol \cup , the above RE can be written as :

$$\text{So, final RE is : } (ab + aaaa)^* (a \cup e)$$



The state 1 can be removed by having a path from $a(b + a^4b)$ instead of $a(b + a^4b)^*$ followed by an optional a as shown below:

To obtain RE from FSA:

The general procedure to obtain a RE from FA is:

Let $M = (Q, \Sigma, S, q_0, F)$ be an FA accepting the language L . Then there exists an equivalent RE (R) for the regular language L such that $R = L(R)$.

Instead of doing the transition relation either in steps shown below:

Symbol a in E, let us allow RE as the labels of states.

Another equivalent machine in which has only symbols a in E, we are supposed to build Giver a machine M , we do the following:

Two states: a start state and a final state.

→ The new machine M' has only one transition which goes from start state to final state. The label on this transition is a .

→ The new machine M' has only one transition which goes from start state to final state. The label on this transition is a .

remove all unnecessary states in a given form:

Now an algorithm to convert from FA to RE:

1) If any.

2) If the machine does not have any final state, then select the RE ϕ .

3) If the start state has any incoming edge to other states, then transition from more states to those states if machine M can be obtained by concatenation of machines M' and M'' .

4) If more than one final state are there
in machine M , then choose a new final
state q_f and connect all final states q_f
machine M to the new final state q_f on ϵ -
transitions and make all final states q_f as
new final states.

5) If there is only final state and there are
outgoing edges from this final state; then
also choose a new final state q_f and
connect this final state q_f machine in
the new final state q_f on ϵ -transitions and
make final state q_f as new final state.

6) Now, if the machine has only one state
and it is q_f then make q_f as new final state
and there are no other transitions, then the
machine acceptable with empty string and
and q_f is the final state and q_f is
the RE as E .

7) If long as we do not have one final state
and one final state perform the following steps:
→ Select an arbitrary state q_f to hold it as not the
final state and met the final state.
→ Remove it from the machine M .
→ Modify the transitions among remaining states so
that in accept the same strings as a RE.
→ In the remaining transitions many states
return to the RE with last from the final state.

$$\begin{aligned}
 E &= 8.6 \\
 \phi &= 18.4 \\
 O &= 8.14 \\
 \theta &= "4 \\
 \overline{R=0} &= 50.0
 \end{aligned}$$



g) Obtain RE for fsm

$$\left[\begin{array}{c} f_1 = f_2 \\ f_3 = f_4 \end{array} \right] = \left[\begin{array}{c} S + Q_1 \\ S + Q_2 \end{array} \right] = \left[\begin{array}{c} f_1 \\ f_2 \end{array} \right]$$

$$\left(\begin{array}{c} f_1 \\ f_2 \end{array} \right) = \left(\begin{array}{c} K_1 \\ K_2 \end{array} \right) \cup \left(\begin{array}{c} K_3 \\ K_4 \end{array} \right)$$

is always less than 2^{n-1} .
 states and some coming out of it. The value of K
 the state of automata i.e. some input entering into
 and $q_0 \rightarrow \text{initial state}$. The inputs are going through
 that $S(q_i, x) = q_j$. where $q_i \rightarrow q_j$ square state
 let q_i denote the set of states \times states
 form m , where $m = \{q_1, q_2, \dots, q_n\}, \subseteq S, Q, F\}$
 Let L be the set of language accepted by

Building RE from PSM using Kleen's method

$$O = \boxed{e^{i\theta}}$$

$$\partial \cap O =$$

$$e \cap O =$$

$$e \cdot (e) \cap e^{i\theta} =$$

$$e =$$

$$O \cdot (e) \cap e =$$

$$e \cdot e^{i\theta} \cap e^{i\theta} =$$

$$\phi =$$

$$e \cdot (\phi) \cap \phi =$$

$$\phi \cdot (e) \cap e =$$

$$O =$$

$$O \cdot e \cdot \cap O =$$

$$O \cdot (e) \cdot e \cap O =$$

$$e^{i\theta} \cdot (e) \cap e^{i\theta} =$$

$$e =$$

$$e \cap e =$$

$$e \cdot (e) \cap e =$$

Final ans

$$\boxed{k=1}$$

Note: Refer class notes for more examples.

$$1 \ 0 \ 0 \ 0 =$$

$$(\exists + I) \cdot (\exists + I) \ 0 \ 0 \ 0 =$$

$$\overline{\mathcal{E}^{\mathcal{E}}_{\mathcal{A}} \cdot (\mathcal{E}^{\mathcal{A}}_{\mathcal{B}})} \cap \mathcal{E}^{\mathcal{A}}_{\mathcal{B}} = \frac{\mathcal{E}^{\mathcal{A}}_{\mathcal{B}}}{K=2}$$

$$(\exists + I) = \phi \cap (\exists + I) = 0 \cdot \exists \cdot \phi \cap (\exists + I) =$$

$$\mathcal{E}^{\mathcal{A}}_{\mathcal{B}} \cdot (\mathcal{E}^{\mathcal{A}}_{\mathcal{B}}) \cdot \mathcal{E}^{\mathcal{A}}_{\mathcal{B}} \cap \mathcal{E}^{\mathcal{A}}_{\mathcal{B}} = \mathcal{E}^{\mathcal{A}}_{\mathcal{B}}$$

$$\phi \cdot = \exists \cdot \exists \cdot \phi \cap \phi =$$

$$\mathcal{E}^{\mathcal{A}}_{\mathcal{B}} \cdot (\mathcal{E}^{\mathcal{A}}_{\mathcal{B}}) \cdot \mathcal{E}^{\mathcal{A}}_{\mathcal{B}} \cap \mathcal{E}^{\mathcal{A}}_{\mathcal{B}} = \mathcal{E}^{\mathcal{A}}_{\mathcal{B}}$$

$$0 = 0 \cap 0 = 0 \cdot \exists \cdot \mathcal{E} \cap 0 =$$

$$\mathcal{E}^{\mathcal{A}}_{\mathcal{B}} \cdot (\mathcal{E}^{\mathcal{A}}_{\mathcal{B}}) \cdot \mathcal{E}^{\mathcal{A}}_{\mathcal{B}} \cap \mathcal{E}^{\mathcal{A}}_{\mathcal{B}} = \mathcal{E}^{\mathcal{A}}_{\mathcal{B}}$$

$$\mathcal{E} = \mathcal{E} \cap \mathcal{E} \cdot \mathcal{E} = \mathcal{E}$$

$$\mathcal{E}^{\mathcal{A}}_{\mathcal{B}} \cdot (\mathcal{E}^{\mathcal{A}}_{\mathcal{B}}) \cdot \mathcal{E}^{\mathcal{A}}_{\mathcal{B}} \cap \mathcal{E}^{\mathcal{A}}_{\mathcal{B}} = \mathcal{E}^{\mathcal{A}}_{\mathcal{B}}$$

$$= \overline{I=K}$$

$$\mathcal{E} + I = \mathcal{E}^{\mathcal{A}}_{\mathcal{B}}$$

$$0 = \mathcal{E}^{\mathcal{A}}_{\mathcal{B}}$$

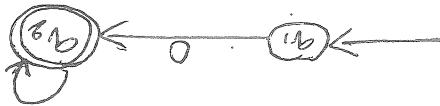
$$\phi = \mathcal{E}^{\mathcal{A}}_{\mathcal{B}}$$

$$\exists = \mathcal{E}^{\mathcal{A}}_{\mathcal{B}}$$

$$= \overline{O=K}$$

$$\mathcal{E}^{\mathcal{A}}_{\mathcal{B}} \cdot (\mathcal{E}^{\mathcal{A}}_{\mathcal{B}}) \cdot \mathcal{E}^{\mathcal{A}}_{\mathcal{B}} \cap \mathcal{E}^{\mathcal{A}}_{\mathcal{B}} = \mathcal{E}^{\mathcal{A}}_{\mathcal{B}}$$

$$= \overline{PQ}$$



2) Obtain RE for the following PSM.

For the given language, if regular expression and FMS
 and FMS can be constructed, then the language
 is called regular language. So, a regular language
 can either be represented using a regular
 expression or by a FMS.
 The Kleene theorem clearly tells us that there
 is no difference between the power of regular
 expressions and power of FMS as regular language
 can either be represented using REs, RL using
 FMS. But, we should note the following:
 → Sometimes something requiring regular expression will be easy
 → Sometimes something constituting DFA is easy
) Is the foll. language regular?
 $L = \{w : w \in \{a, b\}^*\text{ and there is no move from state } b\text{ to state } a\}$
 Sol:- The language L can be expressed by the
 regular expression:
 $a^* (b + \epsilon)^*$
 So, the given language is regular.

Equivalence of Regular Expressions and FMS
 (Kleene's Theorem)

a) Is the language $L = \{w : w \in \{a, b\}^*\}$ and no two consecutive letters are same \Rightarrow is regular?

Applications of Regular Expressions:

Regular expressions are widely used in all most all languages and operating systems.

(a) Regular expressions in UNIX: REs are extensively used in UNIX operating system. But, certain short "a+b+c+d+e....", the operator | is used in place of +, the operator | means "also as one of".

Most of the commands are involved inually uses regular expressions. For example, grep (Global Search for Regular Expression and Print) used to search for some common patterns. we can match an indefinite string by a demand number as we can search for a string in the text.

b) Pattern matching: It refers to a set of objects with some common properties. we can match an indefinite string by a demand number as we can search for a string.

$$RE = (e+a)^* (ba)^* (e+b)$$

(Ans)

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

Ans :-

Ques:-

Two consecutive letters are same \Rightarrow is regular?

all the tokens which are logically together
 This phase scans the source program and recognizes
 phase (This is the first phase of the compiler and analyzes
 extensively used in the design of lexical analysis
 e) Lexical Analysis: Regular expressions are

$$a) \alpha(n_3) = \text{Pathem} \propto \text{most accurate exactly in times}$$

$$b) \alpha(n, m_3) = \text{Pathem} \propto \text{most accurate at least in times and no more than}$$

Note:-

$$(A - \emptyset) \cup (A - \emptyset) \cup ((A - \emptyset) \cup (A - \emptyset)) \cup ((A - \emptyset) \cup (A - \emptyset)) \cup \{(3, \#)\}$$

no more than 8 characters
 → passed must contain atleast 4 characters and
 underscores character
 → passed content only letter numbers, and
 passed must begin with a letter
 → determining string is a legal passed value

$$b) \text{Matching up address: } (f - g)[1, 3] \cup (f - g)[1, 3] \cup (f - g)[1, 3]$$

$$- i(f - g)_+ \cup i(f - g) \cdot (f - g)$$

e) matching decimal numbers:

$$(x \cup y) \cup z = x \cup (y \cup z)$$

3) Union is associative.

$$x + y = y + x$$

4) Union is commutative.

Elementary building union (with respect to sets):

Let x, y and z are regular expressions. Then
Manipulating and Simplifying Regular Expressions. The
following identities are used while simplifying the
regular expressions:

$$\begin{aligned} & (1) x \cup y \Leftarrow x \text{ should occur minimum once.} \\ & (2) x \Leftarrow x \text{ must occur once} \\ & (3) x^* \Leftarrow x \text{ may occur zero or more numbers of times.} \\ & (4) x^0 = (x \cup e) \text{ i.e. } x \text{ optional.} \end{aligned}$$

Note:-

- b) In XML, the REs are one way to define parts of new document types.
- c) Meaningful words in pattern sequences are using regular expression matching.
- d) REs can be matched against the subject fields defined in emails to find same spans.
- e) The programming language Perl supports regular expression matching.
- f) Meaningful words in pattern sequences are called motifs. The motifs can be described using regular expression.
- g) In XML, the REs are one way to define parts of new document types.

$$(ax) + (bx) = (a+b)x \quad (3)$$

$$(ax + bx)r = axr + bxr \quad (4)$$

(III) Concentration distribution over uniform operation

$$\phi = \alpha = \phi$$

3) ϕ is a symbol for concentration

$$\alpha e = \alpha x = \alpha$$

4) e is the identity for concentration

$$(ab)x = a(bx)$$

5) Concentration is associative

(II) Identity involving concentration

Symbol α

Note:- The symbol α can be replaced by number

$$so, \alpha * \cup \alpha a = \alpha *$$

$$\text{then } A \cup B = A$$

5) Given any two sets A and B , if $B \subseteq A$

$$\alpha \cup \alpha = \alpha$$

6) union is idempotent

$$\alpha = \alpha \cup \alpha = \alpha$$

3) ϕ is the identity for union

Note: In the above expression, the symbol \oplus can be replaced

$$*(a+b)$$

$$*(a+b)$$

$$*(a+b)$$

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

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

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

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

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

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

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

$$\text{Simplify the RE: } ((a + \phi) + ab)(b + \phi) + ((a + b) + ab + ba)$$

$$(a + b) \in L(B) \subseteq L(A) \cap L(B) \text{ then } (A \cap B) \subseteq L(B)$$

$$a = b \text{ then } a \in L(A) \cap L(B) \text{ if } A \subseteq B$$

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

$$a = b \text{ then } a \in L(B) \text{ if } B \subseteq A$$

$$5) (A \cup B) = (A \cup B)$$

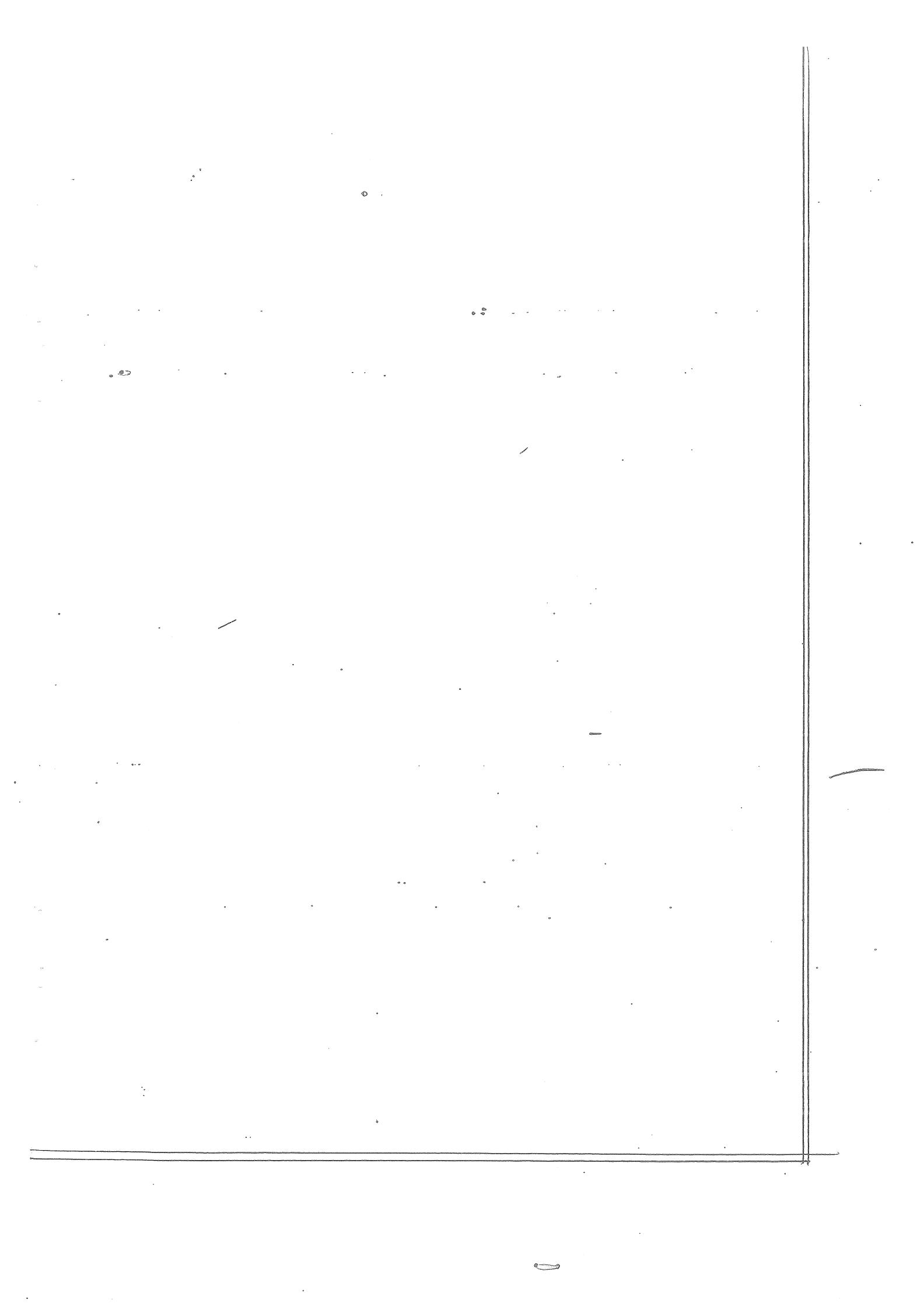
$$4) a * a = a$$

$$3) (A) = A$$

$$2) e * e = e$$

$$1) \phi * e = e$$

Idempotent involving Kleene star: (V)



Note 1 - The non-terminal words can be replaced by terminals
as non-terminals, whereas the terminals cannot be replaced.

/ The non-terminals can be replaced by terminals
as terminals
* Followed by combinations of non-terminals and
* Followed by an arrow
* Each production starts with non-terminal
Sentence \Rightarrow called productive.
The variables which are used to define the
variables

\Rightarrow The words "Rashmi", "Dmitri", "metly" are called
The variables are also called non-terminals
adverb are called variables.
In the above example: Sentence, noun, verb and

Sentence \Rightarrow (noun) \cup (verb) \cup (adverb)

Rashmi	verb	adverb	Noun
metly	adverb		

Consider the sentence :-

Irregular grammar

Regular grammars sometimes called as right

Regular grammar

RASHMI, M
ASY, PAF,
DEPT, G (SE)
PNSI7

Module - 2

Notation

$$\begin{array}{c} T \leftarrow S \\ T \leftarrow aS \\ T \leftarrow ST \\ S \leftarrow aSa \\ \hline \text{Not legal rules} \end{array}$$
$$\begin{array}{c} T \leftarrow aS \\ S \leftarrow e \\ S \leftarrow a \\ \hline \text{Legal Rules} \end{array}$$

non-terminal

a single terminal followed by a single

Right-hand side is a single terminal of

Left-hand side should be a single non-terminal

S (the start symbol) is a non-terminal

of the form $x \leftarrow y$

R (the set of rules) is a finite set of rules

Z (the set of terminals) is a subset of V

terminals and non-terminals

V is the rule alphabet without constants

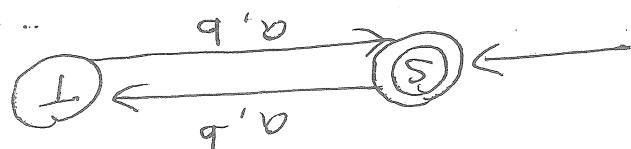
A regular grammar is a quadruple (N, Z, R, S)

$\Rightarrow abab \Leftarrow$
 $\Leftarrow ababS$
 $\Rightarrow abAT$
 $Sb \Leftarrow$
 $S \Leftarrow aT$

Derivation of string "abab"
 Derivation of string "abab" by using Rules

$T \leftarrow bS$
 $T \leftarrow aS$
 $S \leftarrow bT$
 $S \leftarrow aT$
 $S \leftarrow \epsilon$

Regular Grammar by deriving L



DFA accepting L

So: $R_E = ((a \cup b)^*)^*$

$L = \{ w \in (a, b)^* : |w| \text{ is even} \}$

i) Even length strings

Example of Regular Grammar:

$E \leftarrow B$
 $E \leftarrow A$

We generate two E-transitions.
A and B are final states, then in the grammar transition. For example, in a DFA, if the states Note: For each final state s of E , the E -

formally written as:
The language generated by grammar can be

$S \leftarrow AS$

$E \leftarrow S$

if any number of E is as:

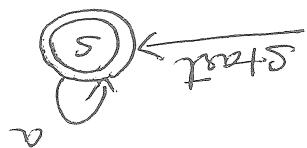
∴ The grammar to generate strings consisting

$$S \leftarrow S \leftarrow S \leftarrow AS \quad S \leftarrow S \leftarrow S \leftarrow S$$

$$\uparrow \qquad \uparrow \qquad \uparrow$$

$$S = (a \cdot S) S$$

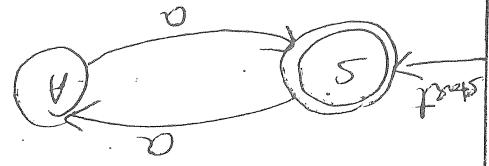
$E \leftarrow S$ $S \leftarrow$ a final state
Grammar Transitions



DFA is 18
 \Rightarrow

Q) Obtain grammar to generate strings consisting of any number of as.

$S \xrightarrow{a} S$
 $S \xrightarrow{e/a} S$
 The grammar is



$$\begin{array}{c}
 S \xleftarrow{a} S \\
 S \xleftarrow{e/a} S \\
 S \xleftarrow{a} A \\
 S \xleftarrow{e} S
 \end{array}
 \quad \frac{\text{Grammars}}{\text{Transducers}}$$

DFA

4) DFA from grammar to generate strings of
 even numbers of 'a's
 (DFA)

$$L = \{ (a+b)^n : n \geq 0 \}$$

The language generated by grammar can be
 generated by grammar can be
 above products can also be written as:
 Since all products start from S, the

$$\begin{array}{c}
 S \xleftarrow{a} S \quad S = (a, b) \\
 S \xleftarrow{b} S \quad S = (a, b) \\
 S \xleftarrow{a} S \quad S = (a, b) \\
 S \xleftarrow{b} S \quad S = (a, b)
 \end{array}
 \quad \frac{\text{Grammars}}{\text{Transducers}}$$



Grammars
 DFA is
 any number of 'a's and 'b's.

3) DFA from grammar to generate strings consisting of

$$L = \{w : w_0 \in \{a, b\}^*, w \in \{a, b\}^*\}$$

$$A \xrightarrow{\quad} \begin{array}{|c|c|} \hline a & b \\ \hline A & A \\ \hline \end{array}$$

$$S \xleftarrow{\quad} \begin{array}{|c|c|} \hline a & b \\ \hline S & A \\ \hline \end{array}$$

A is a final state

$$A \xleftarrow{\quad} e$$

$$A \xleftarrow{\quad} bA$$

$$A \xleftarrow{\quad} aA$$

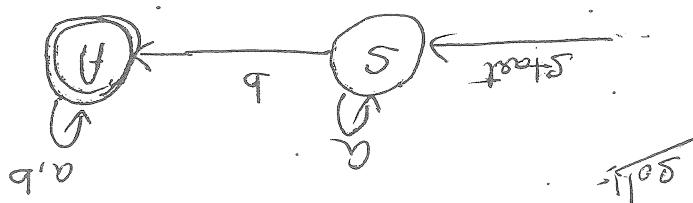
$$S \xleftarrow{\quad} bA$$

$$S \xleftarrow{\quad} aS$$

$$S = (S, a) \quad S = (S, b)$$

Grammar

Transitions



base b

- 5) Define grammar to generate strings consisting of any numbers of a's and b's with atleast one a

Example

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

$a_0 \xrightarrow{a} a_1 \xrightarrow{a} a_2 \xrightarrow{a} a_3 \xrightarrow{a} \dots$

$a_0 \xrightarrow{a} a_1 \xrightarrow{a} a_2 \xrightarrow{a} a_3 \xrightarrow{a} a_4 \xrightarrow{a} a_5 \xrightarrow{a} a_6 \xrightarrow{a} a_7 \xrightarrow{a} a_8 \xrightarrow{a} a_9$

Transitions

$a_0 \xrightarrow{a} a_1 \xrightarrow{a} a_2 \xrightarrow{a} a_3 \xrightarrow{a} a_4 \xrightarrow{a} a_5 \xrightarrow{a} a_6 \xrightarrow{a} a_7 \xrightarrow{a} a_8 \xrightarrow{a} a_9$

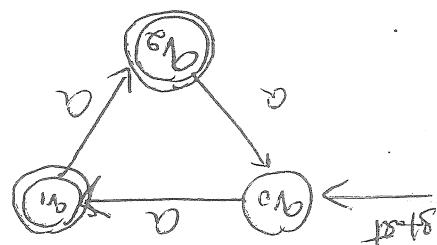
Transitions

$$S(a_0, a) = a_1 \quad a_0 \equiv a_1 \pmod{3}$$

$$S(a_1, a) = a_2 \quad a_1 \equiv a_2 \pmod{3}$$

$$S(a_2, a) = a_3 \quad a_2 \equiv a_3 \pmod{3}$$

$$S(a_3, a) = a_4 \quad a_3 \equiv a_4 \pmod{3}$$



DP

Accepting state = a_1, a_2

Starting state = a_0

\therefore The states are a_0, a_1, a_2 | $k > 3$

$\therefore e = 0, 1, a$

$e = 0, 1, a, 0, 0, 1, a, 0, 1, a, 0$

So, $D = 0, 1, a, 3, 4, 5, 6, 7, 8, 9$

where $e \in \{a\}$

6) obtain grammar to accept $L = \{w : |w| \bmod 3 > 0\}$

1) Obtain grammar to generate strings consisting of atleast one a.

2) Obtain grammar to generate strings consisting of any number of a's and b's with atleast one a.

3) Obtain grammar to generate strings of b's such that string length is multiple of 3.

4) Obtain grammar to generate strings consisting of any number of a's and b's with atleast one a.

5) Obtain grammar to generate strings consisting of multiples of three a's.

6) Obtain grammar to generate strings consisting of any numbers of a's and b's with atleast one a.

7) Obtain grammar to generate strings consisting of an odd number of a's and m ends in a.

8) Obtain grammar for $L = \{w : \text{there is a symbol } a, b, c \text{ in } w \text{ and } L \subseteq \{a, b, c\}^*\}$



two languages.

- automata that recognizes the intersection of these two different languages, we can construct an NFA for each language. So, if there are two automata for example, the intersection of two regular languages is also regular. So, if there are two regular languages it is a useful tool for building complex automata using the closure property, we can build languages as a closure property helps us to construct complex languages. The closure property helps us to construct more complex finite automata.
2. Closure property of regular languages: This property

a. Decidable property

b. Decidable property

are shown below:

The two important properties of regular languages

NFAs as E-NFAs.

languages as the languages accepted by DFAs as the same languages. Thus, we can define regular expressions in terms of others and all of them accept regular expressions. We know that each FA can be defined by DFAs, NFAs and E-NFAs and defined by

regular languages as the languages accepted by regular languages show four different definitions. The languages have known as sequential

The class of languages known as sequential

Properties of Regular Languages

f.

Regular languages

RNS17

Dept. of CSSE

Aest. professor

RASHMI. M

too) called pumping lemma
languages are not regular using one powerful
are not regular? we can prove that certain
question is "How to prove that certain languages
many languages which are not regular. Now, the
Apost from above four languages, these are so

4. Language consisting of matching parentheses

3. $L = \{a^p | p \geq 2\} \subset \text{prime numbers}$

2. $L = \{a^n b^n | n \geq 0\}$

1. $L = \{w : w \in \{0, 1\}^* \text{ and } \#0 = \#1\}$

of the non-regular languages:

Any finite language can be expressed using
regular expressions and we can construct finite
automata (DFA as NFA as E-NFA). Though the
regular languages are important, there are
non-regular languages which are very interesting
and important. For example, following are some
numbers of os and 1's

9. Decision problem of regular languages:
Using this property, we can decide whether
two automata define the same language. If
so, we can minimize the states of automata with
as few states as possible. The minimization of
switching circuit. This is because, as the
number of states of automaton decreases the size
of the circuit and hence the cost decreases.



Since we draw in input symbols, naturally we should have $m+1$ states in the sequence $q_0, q_1, q_2, \dots, q_m$. Where q_0 will be the start state and q_m will be the final state as shown below:

Proof: Let $M = (Q, \Sigma, S, q_0, F)$ be an FA and let $x = a_1, a_2, a_3, \dots, a_m$ choose $m \geq n$ and each $a_i \in \Sigma$ be the language accepted by DFA and is regular.

Then $x \in L$ if and only if $|x| \leq n$.

$\forall i \neq j, |v_i| \leq 1$

Satisfying the foll. conditions:

$$x = uvw$$

and is such that

string x can be broken into three substrings u, v, w where $|v| > 0$. Now, if the exists a constant c such that $|v|^c \leq n$. Let v easily satisfying $x \in L$, there accepted by M . Let v easily satisfying $x \in L$, there numbers of states. Let L be the regular language

Let $M = (Q, \Sigma, S, q_0, F)$ be an FA and has a

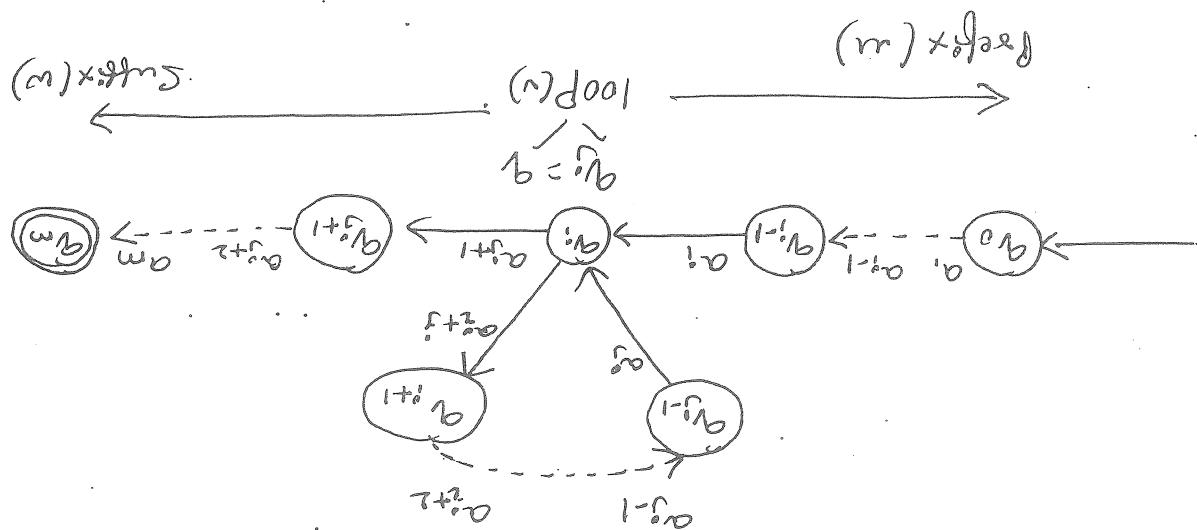
Theorem:

State and prove pumping lemma for regular languages

Pumping Theorem (Lemma) for regular languages

$$\text{With } q = 0$$

that can be accepted by the above FA as it machine goes from q_0 to q_f . The minimum string length $\alpha_i = q_f$) and suffx string so takes the loop string & takes the machine from q_0 to q_f , the string α_i takes the machine from q_0 to q_f , the observe from above figure that, the prefix



The first group is the string prefix from $\alpha_0 \alpha_1 \alpha_2 \dots \alpha_f$ i.e., $m = \alpha_0 \alpha_1 \alpha_2 \dots \alpha_f$

The second group is the loop string from $\alpha_{f+1} \alpha_{f+2} \dots \alpha_{f+3}$ i.e., $n = \alpha_{f+1} \alpha_{f+2} \dots \alpha_{f+3}$

The third group is the suffx from $\alpha_{f+4} \alpha_{f+5} \dots \alpha_m$ i.e., $m = \alpha_{f+4} \alpha_{f+5} \dots \alpha_m$

Since, $|X| \geq n$, by the pigeon hole principle it is not possible to have distinct transitions. One of the states can have a loop. Let the string α be divided into three substrings as shown below:

But, when $i=1$, the string ww is accepted by DFA.

When $i=2$, the string ww is accepted by DFA. So, if $i > 0$, the machine goes from a_i to a_i based on the value of x and then goes to accepting state in input string w . In general, if the string x is split into $a_0, a_1, a_2, \dots, a_{i-1}, a_i, a_{i+1}, a_{i+2}, \dots, a_m$ below:

$$S(a_0, a_1, a_2, \dots, a_{i-1}, a_i, a_{i+1}, a_{i+2}, \dots, a_m) = S(S(a_0, a_1, a_2, \dots, a_{i-1}, a_i), a_{i+1}, a_{i+2}, \dots, a_m)$$

$$= S(a, a_2, \dots, a_{i+1}, a_{i+2}, \dots, a_m)$$

$$= S(a, a_3, a_4, \dots, a_m)$$

$$Also, after the string$$

$$a, a_2, \dots, a_i$$

$$the machine will be in state a_i . Since a_i and a_j are same, we can input the string $a_j$$$

$$any numbers of times and the machine will stay in$$

$$as long as the input string is$$

$$of size m . Finally, if the input string$$

$$is of size m , then we have to find state a_m .$$

These can be expressed using the transitions as shown

Switches now, then for all $i \geq 0$, we have

$S(a_0, a_1, a_2, \dots, a_{i-1}, a_i, a_{i+1}, a_{i+2}, \dots, a_m) = S(S(a_0, a_1, a_2, \dots, a_{i-1}, a_i), a_{i+1}, a_{i+2}, \dots, a_m)$

below:

Applications of pumping lemma:

The pumping lemma is a very powerful tool and has the following applications:

1. Pumping lemma is used to prove that certain languages are non-regular. But, it cannot be used to prove that a given

2. Pumping lemma is used to prove that a language is regular.

3. Using pumping lemma, it is possible to prove that a language is accepted by a check whether a language is infinite.

FA is finite if and only if

The given language is non regular.

$$\rightarrow \not\exists \overline{w} = w$$

$$w = a^q b^r \text{ for } q \leq 0$$

$$w = (aa)(a)^q (bb) = aaaa bbb \notin L$$

$$w = (aa)(a)^q (bb) = aaaa bbb \in L$$

$$w = (aa)(a)^q (bb) = aaaa bbb$$

$$q = 0, 1, 2, \dots$$

Thus it is in the form ~~xyz~~

$$= (aa)(a)^q (bb)$$

$$w = \frac{z}{a} \frac{y}{a} \frac{x}{b} b$$

$$x \leq 3$$

$$n \leq |w|$$

$$np = n + n = 2n$$

$$a^p b^q$$

Let n be the no. of states

$$\frac{n}{m} = aaaa bbb$$

(i) Show that $L = \{a^p b^q \mid n \leq 0\}$ is non regular.

$$\begin{array}{c}
 \text{Q8} \\
 \text{L} = \{ a^m b^n c^k \mid m, n, k \in \mathbb{N}, m \geq n \geq k \} \\
 \text{L} = \overline{\{ a^m b^n c^k \mid m, n, k \in \mathbb{N}, m \geq n \geq k \}}
 \end{array}$$

(3) Show that the language $L = \{ a^m b^n c^k \mid m, n, k \in \mathbb{N}, m \geq n \geq k \}$ is not regular.

\therefore The language is not regular

$$L = \overline{\{ a^m b^n c^k \mid m, n, k \in \mathbb{N}, m \geq n \geq k \}}$$

$$a^m b^n c^k$$

$$\begin{array}{ccc}
 f & g & h \\
 \sim & \sim & \sim \\
 a & a & a \\
 \vdash & \vdash & \vdash \\
 a^m & a^n & a^k
 \end{array}$$

$$\begin{array}{c}
 u = a^m + b^n + c^k \\
 u \geq 1 + u \geq 1 + u = m + n + k = m
 \end{array}$$

$$L = \{ a^m b^n c^k \mid m, n, k \in \mathbb{N}, m \geq n \geq k \}$$

$$L = \{ a^m b^n c^k \mid m, n, k \in \mathbb{N}, m \geq n \geq k \}$$

not regular

(2) Show that the $L = \{ a^m b^n c^k \mid m, n, k \in \mathbb{N}, m < n < k \}$

" Hence language is not regular

$$L = \{a^{n-1} b^n \mid n \in \mathbb{N}\}$$

when $a =$

$$x \cdot y \cdot z = a^{n-1} \cdot a \cdot b^n$$

$$\frac{x}{a} \cdot \frac{y}{a} \cdot \frac{z}{b^n} =$$

$$n \geq m < n + m - 1$$

So $L = \{a^m b^n \mid m, n \in \mathbb{N}, m \neq n\}$ as the no. of a 's & b 's are equal

regular

so $L = \{\omega \mid \omega \in \{a, b\}^*, \omega(\omega) = \omega\}$ s.t. $\omega(\omega) = \omega$

$$L = \{w \mid w \in \omega\}$$

$$w_1 w_2 \in \omega$$

$$(w_1)(w_2) = w$$

when $a =$

$$(01)^k (10)^l (00^m 00^n 0) = 0^k 1^l 0^m 1^l 0^n 0$$

Hence L is not regular.

$$w = \overline{ad \cdot d \cdot da}$$

when $a \neq 0$

$$w = \overline{a \cdot d \cdot a \cdot da} = ab$$

$$w = \overline{d \cdot a \cdot c \cdot \frac{a}{d} \cdot \frac{b}{c}} = ab$$

$$w \in p \cdot q = p + q = L$$

$$L = \overline{ad \cdot da}$$

$$L = \overline{d + a} = d$$

$$L = \overline{a + f} = L$$

$$L = \overline{a \cdot a \cdot c \cdot a + f} = L$$

$$h(a) = a \quad h(b) = a \quad h(c) = c$$

Hence we can use substitution letters.

Homomorphism is closed under regular languages

$$L = \overline{a \cdot b \cdot c \cdot a + f}$$

regular

$$(5) S.T \quad L = \overline{(a \cdot b \cdot c \cdot a + f)^n} \text{ is not}$$

$$L = \{a^m b^n \mid m \in \mathbb{N}, n \in \mathbb{N} \text{ and } m \neq n\} \quad (1)$$

$$L = \{a^m b^n \mid m \in \mathbb{N}, n \in \mathbb{N} \text{ and } m \neq n\} \quad (2)$$

$$L = \{a^m b^n \mid m \in \mathbb{N}, n \in \mathbb{N} \text{ and } m \neq n\} \quad (3)$$

$$L = \{a^m b^n \mid m \in \mathbb{N}, n \in \mathbb{N} \text{ and } m \neq n\} \quad (4)$$

Hence not regular

$$\omega \omega = \frac{a_n b_m a_{n+1} b_n}{\text{when } q=0}$$

$$a_n b_m \cdot a_{n+1} b_n = (a_n b_m) \cdot (a_{n+1} b_n)$$

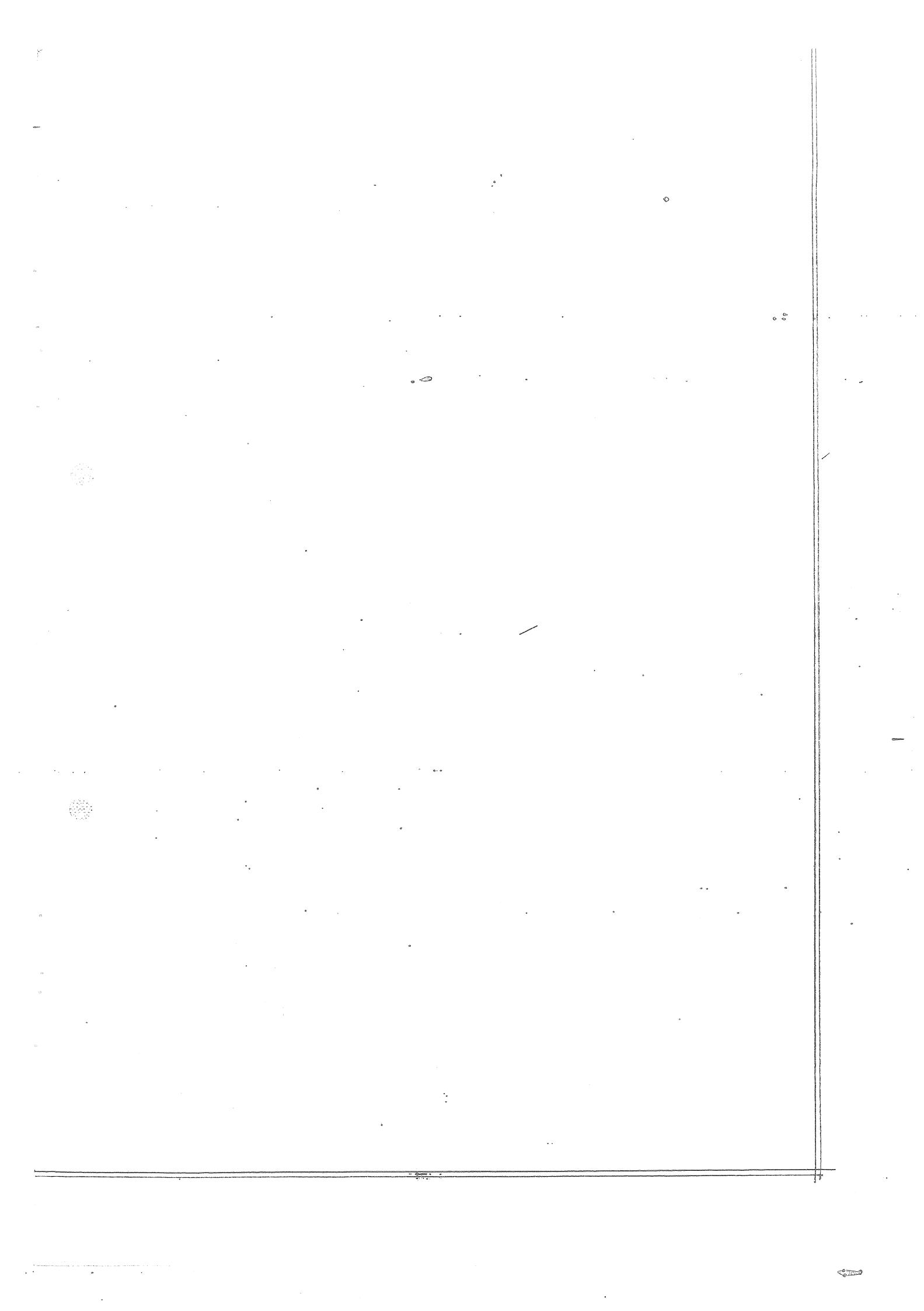
$$\frac{x}{a_n b_m} \cdot \frac{y}{a_{n+1} b_n} = \omega \omega$$

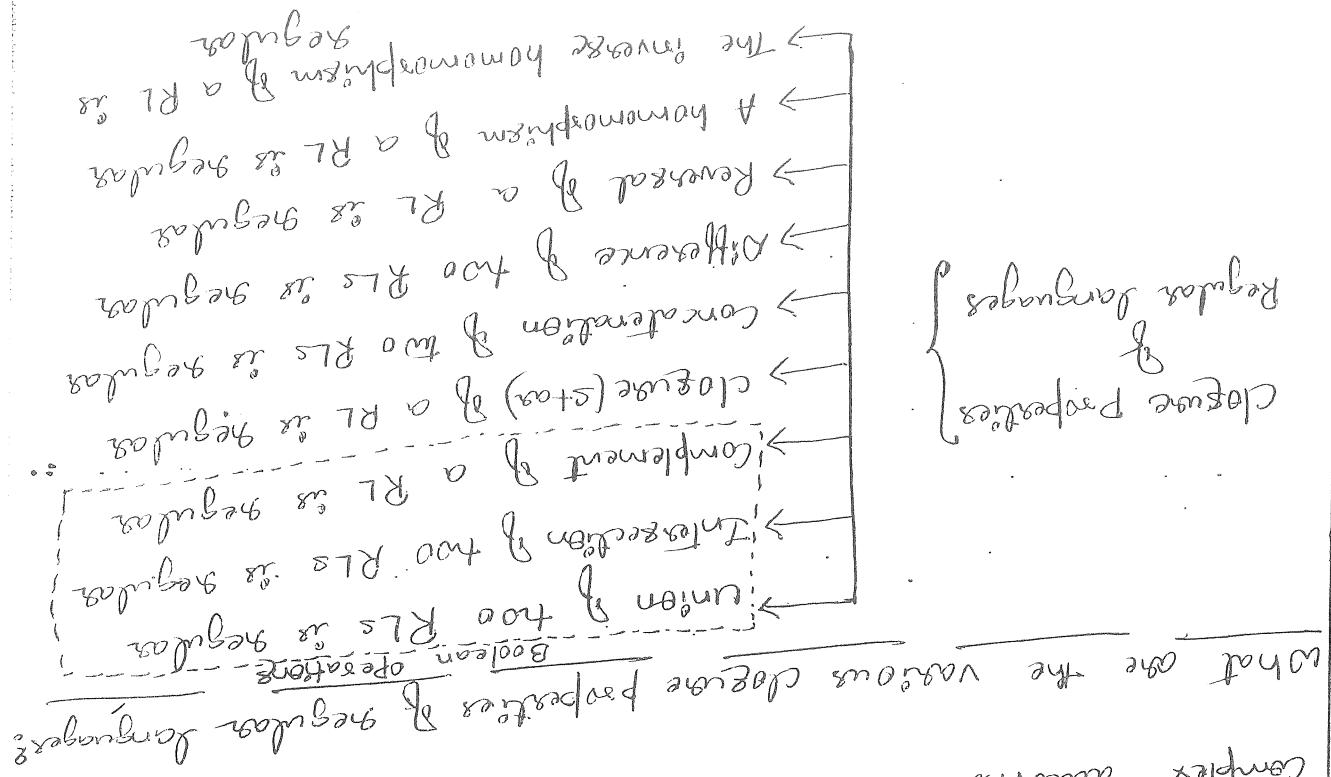
$$\omega \omega = a_n b_m a_{n+1} b_n$$

$$\text{So, if } \omega = a_n b_m$$

$$a_n b_m \approx 1 \text{ or } 0$$

$$L = \{\omega \omega \mid \omega \in \{a, b\}^*\} \text{ is not regular.} \quad (5)$$





Regular languages
of
Closure properties

Complex automata
thus using the closure properties we can build an
automaton thus obtaining the two automata
of these two languages. The automaton thus obtained
consists in an automaton that recognizes the closure of both
that sequence too different languages, we can
languages as regular. Thus, given two finite automata
for example, the intersection of two regular
languages is regular. Thus, given two finite automata
many have the closure properties we can build an
automaton thus obtaining the two automata
of these two languages. The automaton thus obtained
consists in an automaton that recognizes the closure of both
that sequence too different languages, we can
languages as regular.

closure properties.

Some languages can be constructed from already
existing languages using certain operations such as
union, intersection, concatenation etc. we can build
several for these two languages using some of the
closure properties of regular languages. These properties are called
closure properties of regular languages.

Q. What are closure properties?
A. Closure properties are whether a FA's accept

1. Closure properties of union, intersection etc

properties.

The regular languages exhibit Two types of

Properties of Regular languages:

Note :- The first three closure properties :
1. Union, Intersection and Complement are called
unions, intersections and complements and are called
Boolean operations. The regular languages are
closed under these three Boolean operations.
Note :- RL stands for Regular Language
RLs stands for Regular Languages

Theorem: If L_1 and L_2 are regular, then the regular language $L_1 \cap L_2$ is closed under intersection.

Proof: Let us consider $M_1 = (A_1; \Sigma, S_1, q_1, F_1)$ to hold

accept L_1 and $M_2 = (A_2; \Sigma, S_2, q_2, F_2)$ to hold

assume both the machines are DFA's.

\Rightarrow If p is a state in both machines, then $p \in A_1 \cap A_2$. If this is accepted L_2 .

\Rightarrow To accept the language $L_1 \cap L_2$, let us construct the machine M that simulates both M_1 and M_2

where the states of the machine M are the pairs (p, q)

where $p \in A_1$ and $q \in A_2$.

The transition for the machine M from the state $(S, (p, q))$ to the state $(S, (p', q'))$ is as follows

then the machine M moves from the state (p, q) to the state (p', q') if $S_1(p, a) = p'$ and $S_2(q, a) = q'$.

the effect of M_1 and M_2

In this manner, the machine M can simulate state (p, q) on input symbol a .

Now, the machine M moves from the state $(S, (p, q))$ to the state $(S, (p, q))$ if $S_1(p, a) = p$ and $S_2(q, a) = q$.

Now, the machine M moves from the state $(S, (p, q))$ to the state $(S, (p, q))$ where $p \in F_1$ and $q \in F_2$.

\Rightarrow Now, the machine $M = (A, \Sigma, S, q, F)$ accepts $L_1 \cap L_2$.

Now, the machine M moves from the state $(S, (p, q))$ to the state $(S, (p, q))$ where $p \in A_1 \times A_2$ and $q \in M_1 \cup M_2$.

\Rightarrow $L_1 \cap L_2$ where:

$q = (q_1, q_2)$ where q_1 and q_2 are the states defined by $(S_1(p, a), S_2(q, a))$

$A = A_1 \times A_2$

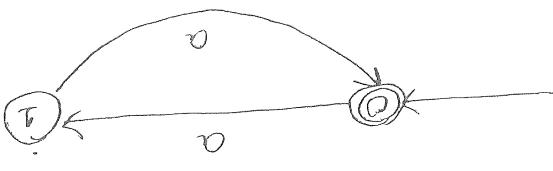
$E = \{(p, q) | p \in E_1 \text{ and } q \in E_2\}$

$S : Q \times E$ to Q is defined by $(S(p, q), a)$

$(S_1(p, a), S_2(q, a))$

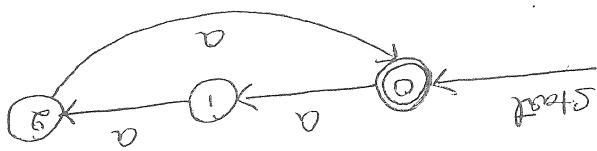
Closure under intersection:

$$A_1 = \{0, 1\}$$



The DFA to accept a string w that results in $|w| \bmod 2$ can be written as shown below.

$$A_1 = \{0, 1, 2\}$$



The DFA to accept a string w that results in $|w| \bmod 3$ can be written as shown below.

Q3 :-

Example :- Obtain a DFA to accept the following language $L = \{w : |w| \bmod 3 = |w| \bmod a\}$ where $w \in \Sigma^*$ and $\Sigma = \{a, b\}$.

Language is closed under intersection i.e., if and only if $w \in L_1 \cap L_2$ so, the result

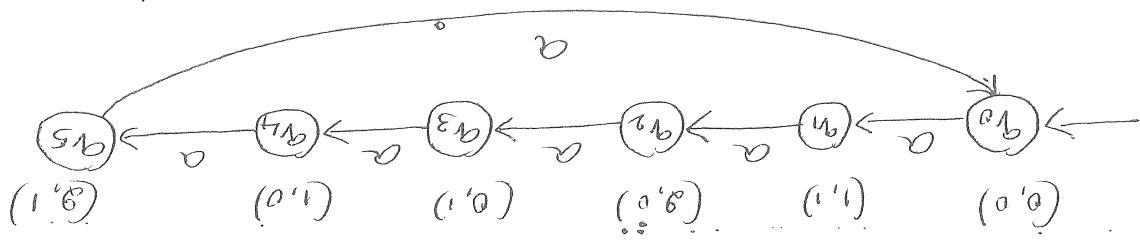
$$S_1(a_1, w) \in F_1 \text{ and } S_2(a_2, w) \in F_2$$

The result happens if and only if

$$\text{f.e. } (S_1(a_1, w), S_2(a_2, w)) \text{ is in } F$$

$$S((a_1, a_2), w) \text{ is in } F$$

The string w is accepted if and only if



$$(0,0) = \begin{matrix} 0 & \dots & 0 \\ ((0,1)S \cdot S(1,0))S & = & S((0,1), a) \end{matrix} \quad \text{a}_5$$

$$(1,0) = \begin{matrix} 1 & \dots & 0 \\ ((0,1)S \cdot S(1,0))S & = & S((0,1), a) \end{matrix} \quad \text{a}_4$$

$$(0,1) = \begin{matrix} 0 & \dots & 1 \\ ((0,1)S \cdot S(0,0))S & = & S((0,1), a) \end{matrix} \quad \text{a}_3$$

$$(1,1) = \begin{matrix} 1 & \dots & 1 \\ ((0,1)S \cdot S(1,0))S & = & S((0,1), a) \end{matrix} \quad \text{a}_2$$

$$(0,0) = \begin{matrix} 0 & \dots & 0 \\ ((0,1)S \cdot S(1,0))S & = & S((0,1), a) \end{matrix} \quad \text{a}_1$$

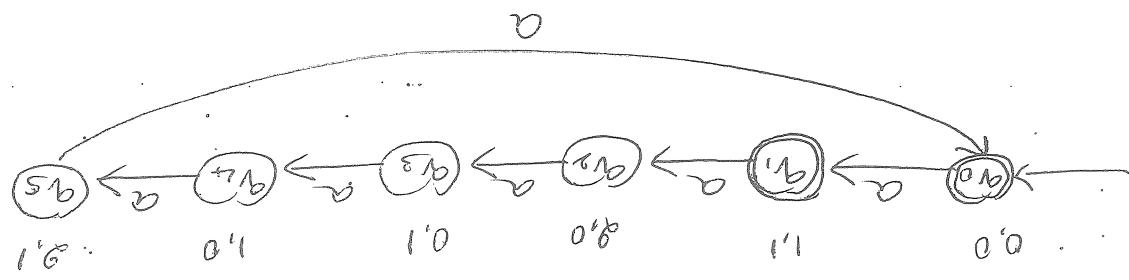
$$(1,1) = \begin{matrix} 1 & \dots & 1 \\ ((0,1)S \cdot S(0,0))S & = & S((0,1), a) \end{matrix} \quad \text{a}_0$$

The transitions in each pair (x, y) can be obtained as shown below:

$$A_1 \times A_2 = \{(0,0), (0,1), (1,0), (1,1), (0,0), (0,1)\}$$

taking the cross product of A_1 and A_2 as shown below:
 so with $|w| \bmod 3 = |w| \bmod 2$ can be obtained by

Transitions: Transitions of DFA which has states



So, the DFA to accept the given language is

$$E = \{(0,0), (1,1)\}$$

States are:

Final states. So, in the above DFA, the final word 2, the pairs (x,y) such that $x \equiv y \pmod{3}$ to accept strings of Σ such that $\Sigma =$

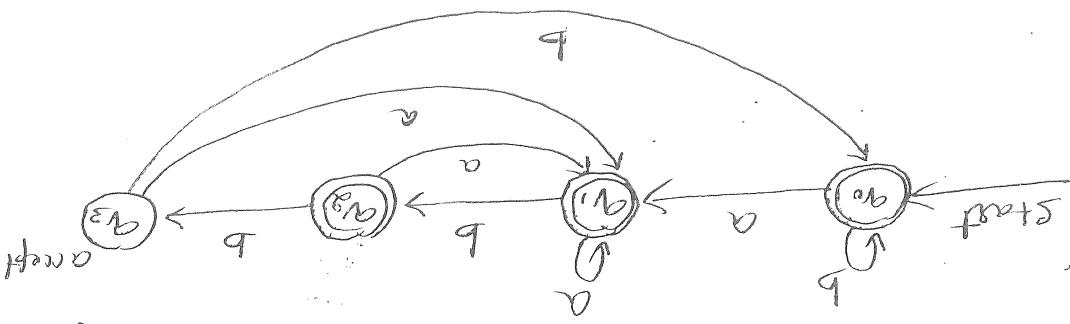
Closure under Complementation

The study of regular languages is done by means of finite automata.

Theorem: If L is a regular language, then the complement of L denoted by \bar{L} is also regular.

Proof: Let $M_1 = \{Q, S, q_0, F\}$ be a DFA which
accepts the language L . Since, the language is
acceptable by a DFA, the language is regular.
Now, if we define - the machine $M_2 = \{Q, S, S, q_0, F\}$
which accepts L .

the use of regular language is desired



Using the closure property of complement, the DFA is obtained by:

Making the new final state in above DFA as final state in new DFA.

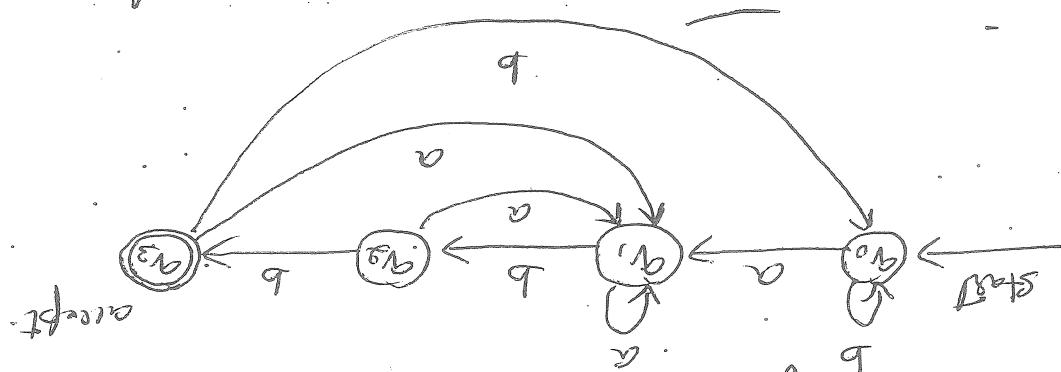
Making the final state in above DFA as final state in new DFA.

Making the new final state in above DFA as final state in new DFA.

The resulting DFA accepting strings of as and bs that do not end with abb is:

and bs that do not end with abb are found in new DFA.

According to the theorem, the regular language described under complement is the complement of the given language "strings of as and bs that are closed under complement". So, the complement are closed under complement of as and bs that do not end with abb.



Example :- Convert a DFA to accept strings of as and bs which do not end in abb.

The DFA to accept strings of as and bs that do not end with abb is:

bs ending with abb is

regular languages to closed under difference.
i.e. if and only if $w \in L_1 - L_2$. So, the

$$S_1(a_1, w) \cap F_1 \text{ and } S_2(a_2, w) \cap F_2$$

These will happen if and only if

$$S_1(a_1, w), S_2(a_2, w) \text{ or } S_1(a_1, w) \cap S_2(a_2, w) = \emptyset$$

$$S((a_1, a_2), w) \text{ in } E$$

The step is to prove closed under difference if

$$F = f(P, Q) \cap F_1 \text{ and } Q \notin F_2$$

start state of M_2

$a_1 \in S((a_1, a_2), w)$ and $a_2 \in S((a_1, a_2), w)$ is the start state where

$$S((P, Q), a) = (S_1(P, a), S_2(Q, a))$$

$S : Q \times E \rightarrow Q$ is defined by

$L = S$ same for both the machines

$$Q \rightarrow Q \times Q$$

beginning $L_1 - L_2$ as follows:

which accept L_2 . we define $M = (Q, E, S, Q_0, F)$ which accept L_1 and $M_2 = (Q_2, E, S_2, Q_2, F_2)$

Proof: Let us consider $M_1 = (Q_1, E, S_1, Q_1, F_1)$.

then $L_1 - L_2$ is also regular.
i.e. if L_1 and L_2 are regular languages,
difference.

Theorem: If L_1 and L_2 are regular languages,
then the regular language is closed under

Closure under difference

Closure under Reversal

What is closure under Reversal?

Let $w = a_1 a_2 a_3 \dots a_n$. The closure of w is denoted by w^R and is defined as a string w denoted by w^R is defined as a string which is written backwards i.e. $w^R = a_n a_{n-1} \dots a_3 a_1$.

String which is written backwards i.e.

What is reverse of a language?

Ex:- Reverse of 0111 denoted by 1110

Reverse of E denoted by E^R is E .

Ex:- Reverse of 0111 denoted by 0111R is 1110

What is reverse of a string?

Let $w = a_1 a_2 a_3 \dots a_n$. The closure of w is denoted by w^R and is defined as a string w denoted by w^R is defined as a string which is written backwards i.e.

Note:- Given a finite automata, the reversed of finite automata can be obtained by reversing all the arcs in the transition diagram.

Ex If $L = \{E, 0, 10, 011, 00111\}$ then

The reversal of a language L^R is defined as the set of all strings of L that are reversed as the set of all strings of L that are reversed

What is reverse of a language?

$$L(E^R) = L(E_1^R) \cup L(E_2^R)$$

languages :

If $E = E_1 + E_2$ is a regular expression denoting the language $L = L(E_1) \cup L(E_2)$

$$\text{Case 1: } E = E_1 + E_2$$

E_1 is a regular expression \leftarrow

$E_1 \cdot E_2$ is a regular expression \leftarrow

$E_1 + E_2$ is a regular expression \leftarrow

E_1 and E_2 are regular expressions, then:

Induction: Again, by definition of regular expression,

$$(ay)^R = (ay)$$

$$(\phi y)^R = \phi$$

$$(Ey)^R = (Ey)$$

So, the general of regular expression E^R is

a is a regular expression \leftarrow

E is a regular expression \leftarrow

ϕ is a regular expression \leftarrow

Base: By definition of regular expression E we have:

which is used as "language of E^R " is the same as of language of E

$$L(E^R) = L(E)$$

that

Proof:- Let L is the language corresponding to regular expression E . It is equivalent to form E^R such that there is another regular expression E^R such

Theorem: If L is regular, then L^R is also regular.

L^R is also regular.

Show that if the language L is regular, then

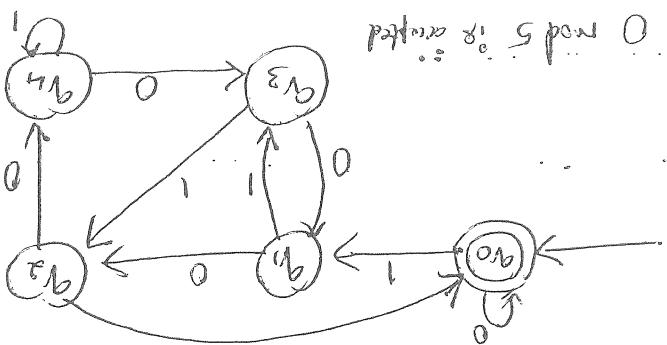
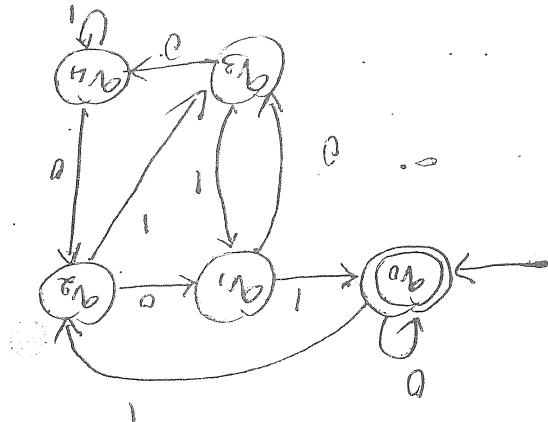
$$\begin{array}{r} 5 \\ \times 3 \\ \hline 15 \end{array}$$

0011001

$$k \equiv (p+q) \pmod{5}$$

$$f_k = g_k = s(a, a)$$

$a \pmod{5}$ if not accepted.



Example :- Obtain a DFA to accept the strings of all lengths that when interpreted as binary fractions, are divisible by 5. i.e. for example, the string 1001100 should be accepted. This is because even though 1001100 is acceptable, its reversal 001100 is not divisible by 5, as mod division by 5, it's remainder 001100 is accepted.

$$L(E_R) = L(E_i)$$

If $E = E_i$ is a regular expression denoting the language L :
Case 3:- $E = E^*$

$$L(E_R) = L(E_i) \cdot L(E_i)$$

the language:

If $E = E_1 \cdot E_2$ is a regular expression, then $E_R = E_1^* \cdot E_2^*$ is a regular expression denoting

$$\text{Case 2:- } E = E_1 \cdot E_2$$

$$(10 \text{ 0}110, 1010) \in L$$

$$((01)(11), (01)(01)) \in L$$

$$((010)(0), (00)(0)) \in L = (00, 01) \in L$$

$$1001110 \in L$$

$$(01)(11) (01) \in (010) \in L$$

$$h(w) = h(a_1) \dots h(a_n)$$

By definition we have

If $L = \{00, 010\}$ is called "homomorphic image of Σ ,
if Σ is made of alphabets from Σ , then $h(\Sigma) = L$

~~If $\Sigma = \{0, 1\}$ and $L = \{0, 1, 0\} = \Sigma$ by~~

~~If L is made of alphabets from Σ , then $h(\Sigma) = L$ is called homomorphic image~~

$$\overline{h(w)} = h(a_1) \overline{h(a_2)} \dots \overline{h(a_n)}$$

If $w = a_1 a_2 a_3 \dots a_n$

Subtracted by a shifting.

i.e. a substitution where a single letter is

function $h: \Sigma \rightarrow \Gamma$ is called homomorphism.

Let Σ and Γ are sets of alphabets. The homomorphism

What is homomorphism?

Closure under Homomorphism:

$$\begin{aligned}
 & h((a_1 + a_2)q) = h(a_1q) * h(a_2q) \\
 & h(a_1q) * h(a_2q) = ((a_1 + a_2)q) * h(a_1q) \\
 & h(a_1q) * h(a_2q) = h(a_1 + a_2)q
 \end{aligned}$$

$$\begin{aligned}
 & h(a_1 + a_2)q = ((a_1 + a_2)q) * q \\
 & h(a_1 + a_2)q = ((a_1 + a_2)q) * q = h(a_1 + a_2)q
 \end{aligned}$$

Limits of finite automaton

There are so many problems for which we cannot construct a DFA and still we want some solution to solve these problems. For example, there are so many grammars, push down automaton, Turing machines etc module-4, will see how these problems can be solved using these 8 types of languages. In the module-3 and machine code is more general than DFA which can have a solution for all these problems is to have a using grammar, push down automation, Turing machines etc module-4.

The solution for all these problems is calculating, finding the string categorizing various types of languages involving counting, 3. Finite automata as finite state machines have trouble in

2. Since it does not have memory, FA cannot remember long amount of information. does not have the capacity to remember a long string. For example, to check for matching parentheses - check whether the string is a palindrome or not etc are not possible using FA

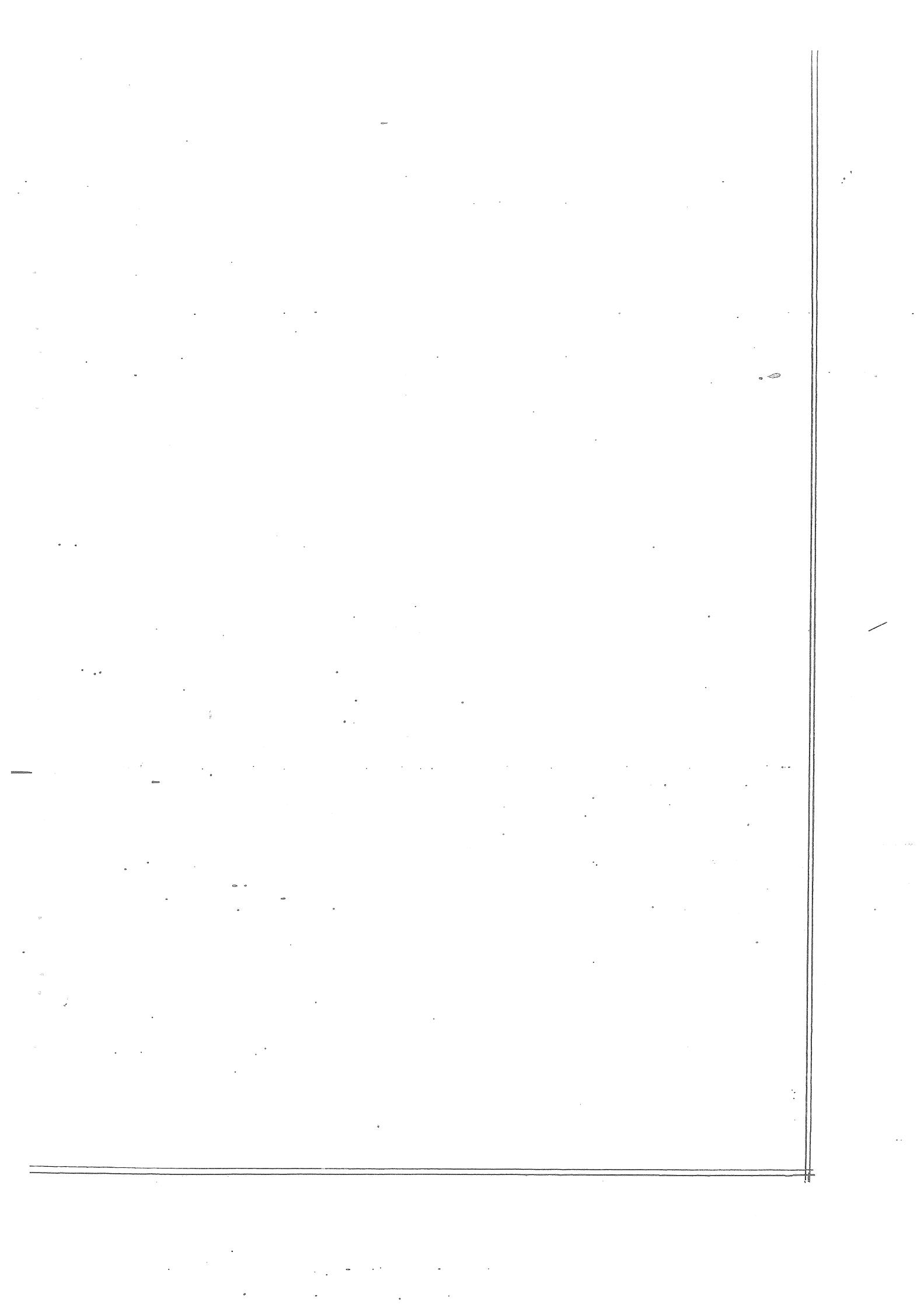
1. An FA has finite number of states and so it is not possible using DFA

a. Count numbers of a's and then the number of b's (not possible using DFA)

1. Check for the matching parentheses

These are so many problems for which we cannot construct a DFA and still we want some solution to solve these problems. For example,

Limits of finite automaton



Use

b \leftarrow stands

a \leftarrow stands

e \leftarrow stands

t \leftarrow stands

g \leftarrow stands

for a start word
for a start word
for else keyword
for then keyword
for if keyword

c \leftarrow cat | catse | a

below:

Grammar to recognize an if-startword as follows:-

s is the start symbol

p is part of products

T is part of terminals

v is part of variables

Use

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

A CFG is a 4-tuple (as quadruple)

Type-2 grammar

The context free grammar also called as

What is context free grammar?

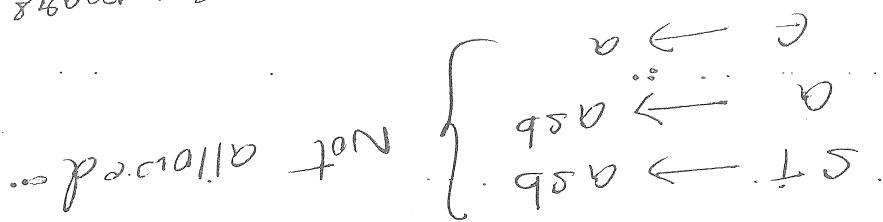
Context free grammar of Languages

45

RASHMI M
Ass't Prof,
Dept. of CSE,
RNSIT

Module - 3
HTC

Note:- The name for these grammar "Context free", makes sense because using these rules the context in word the non-terminal occurs. This sequence is made without looking at the decision to separate a non-terminal looking at the "free" makes sense because these rules



$S \leftarrow a, S \leftarrow b, S \leftarrow T, S \leftarrow ab,$ if RHS have no ~~single~~ strings are allowed

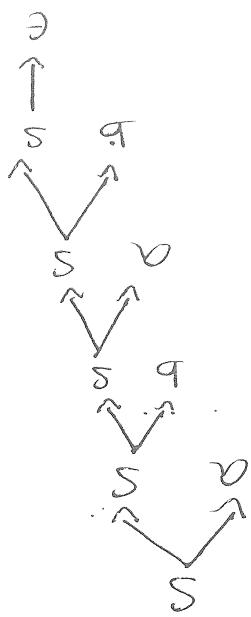
LHS should contain a single non-terminal
LHS \leftarrow RHS

How to make CFG?

of escape the language generated from this
pushdown automata (PDA) can be constructed
called type-2 language of CFG.
The language generated from this grammar is
side of any production.
The symbol ϵ can appear on the right-hand
terminal.

$A \leftarrow \alpha$ where $\alpha \in (\{ \cup T \})^*$ and $A \neq \text{non-}$
In a CFG, all productions are of the form

Note:- In every CFG, observe the following points:



$\Rightarrow abab \leftarrow ababs \leftarrow abas \leftarrow abs \leftarrow s$

$s \leftarrow as | bs | e$
(g)

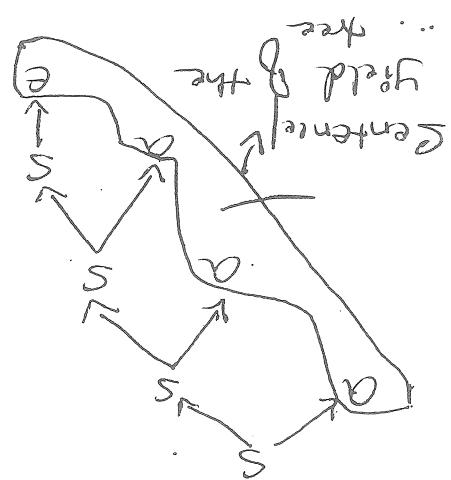
$e \leftarrow s$
 $bs \leftarrow s$
 $as \leftarrow s$

$R: R_E = (a \cup b)^*$

$s \leftarrow (a, b)$

2) Changes with any number of 'a's and 'b's.

Note: Terminal should be at the leaf nodes.



Resultant for aaa :

$s \leftarrow as$

$R: s \leftarrow e$

$R_E = a^*$

$s \leftarrow (a)^*$

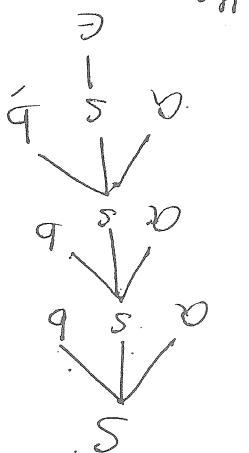
1) Obtain a grammar (G) to generate the starting consisting of any no. of 'a's.

$L = \{ab, aabb, aaabb, \dots\}$

$$59 \cdot 5) = 3 - \sqrt{105}$$

$$\{1, 2, 3, 4, 5\} = 7$$

5) Obtain a grammar to generate the



Ex-1) aaabbba
 Ans: aasbb
 LHS: aasbb
 RHS: aaabbba
 \Rightarrow aasbb
 \Rightarrow aaabbba
 \Rightarrow aaaaabb

n & S
950 ← 5

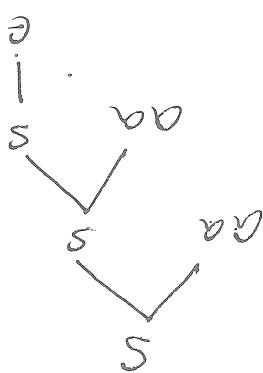
$\exists \leftarrow S$

2

$$L = \{ f, ab, aabb, aaabbb, \dots \}$$

$$\{q, v\} = 3 \quad \text{ipos}$$

(ii) Definition of a grammar for L_0 :



\Rightarrow aaaa
 \Rightarrow aaas
 \Rightarrow aas
 \Rightarrow aa
 \Rightarrow aaaa

$s \leftarrow \alpha s$

$\exists \leftarrow S$

8

$$P.E = (Aa)$$

$$6 \cdot 7 = 5 \cdot 1105$$

3) Obtain a grammar to generate even numbers

3) If w is a palindrome then the string $awaw$ and
 the string bwb are palindromes.
 So if s is a palindrome, asa and bsb are
 palindromes.

Note: E is a palindrome. The equivalent productions
 of $E \leftarrow e$
 of $E \leftarrow a|b$
 of $E \leftarrow s$

10) Between a grammar of generating set of all palindromes

over $\Sigma = \{a, b\}$
 $S \leftarrow SbS$
 $S \leftarrow aSa$

9) Between a grammar for $L = \{a^{m_1}b^{m_2}a^{m_3}b^{m_4}\dots\}$

over $\Sigma = \{a, b\}$
 $S \leftarrow SbS$
 $S \leftarrow aSa$

8) Between a grammar for $L = \{a^{m_1}b^{m_2}a^{m_3}b^{m_4}\dots\}$

over $\Sigma = \{a, b\}$
 $S \leftarrow SbS$
 $S \leftarrow aSa$

7) Between a grammar for $L = \{a^nb^n : n \geq 0\}$

over $\Sigma = \{a, b\}$
 $S \leftarrow SbS$
 $S \leftarrow aSa$

6) Between a grammar for $L = \{a^{n+1}b^n : n \geq 0\}$

- The right hand
- Show the derivation for
- $L = \{e, ab, ba, aabb, abba, aaab, bbbb, baab, \dots\}$
- S $\leftarrow S$
 $bsa \leftarrow S$
 $asb \leftarrow S$
 $e \leftarrow S$ \overline{SOL}
- $L = \{m^a n^b | m \in \{a, b\}^*, n \in \{a, b\}^*\} \quad (1)$
- * A product in numbers
 followed by a number
 is with the same
 followed by an number
 of products in numbers
- $S \leftarrow A B$
 $A \leftarrow \Omega_1 | \Omega_2$
 $B \leftarrow e | \Omega_3$
- $L = \{m^a n^b | m \geq 1 \text{ and } n \geq 0\} \quad \overline{SOL}$
- Obtain the grammar to generate
- Note: To generate the string suitable region that must occur in some fixed order and do not have to expand to each other, then use the rule:
- $S \leftarrow AB$ where $A \neq B$ are regions
- $S \leftarrow asa | bsa$
 $e \leftarrow S$ \overline{SOL}
- $L = \{aa, bb, abba, aaab, bbbb, baab, \dots\}$ \overline{SOL}
- (1) $L = \{\text{www where } w \in \{a, b\}^*\}$

Qd: $P = S \leftarrow S \leftarrow A \leftarrow B \leftarrow B \leftarrow B \leftarrow B$
 [Generates more noise than is
 needed to make it
 easier to find the
 solution] \rightarrow
 [To generate more noise
 than is
 needed to make it
 easier to find the
 solution] \rightarrow
 [At least one extra
 iteration is needed]
 [At least one extra
 iteration is needed]

5 go to the start symbol

→ ← 5

55. ← 5

$\{s\} \leftarrow s$

[s] ← 5

$$(s) \leftarrow s = j$$

$$sy = \wedge$$

$$(S^z)^2 = 9 \quad ; \quad \sqrt{105}$$

Advanced Features

13) Obtain a CFG to generate a string of

16) Obtain a grammar of generalise
 $L = \{a^m b^n \mid m \geq 0, n > m\}$
 by one of the following ways
 The language consists of strings of a 's followed by strings of b 's
 $L = \{EBB^*, ABB^*, AABBB^*, \dots\}$
 $\rightarrow m \geq 2 \quad m \geq 1 \quad m \geq 0 \quad m \geq 0$
 $S \leftarrow AB$
 $A \rightarrow AAB \mid E$
 $B \leftarrow BBE$
 [Generalise strings of more b 's]

$L = \{a^m b^n \mid m \geq 0, n > m\}$
 by one of the following ways
 $L = \{a^m b^n \mid m \geq 0, n > m\}$
 $\rightarrow m \geq 2 \quad m \geq 1 \quad m \geq 0 \quad m \geq 0$
 $S \leftarrow AB$
 $A \rightarrow AAB \mid ab$
 $B \leftarrow BB$
 [Generalise strings of more b 's]

$A \rightarrow B A C C | E$
 $S \rightarrow A S C | A$
 This grammar is
 $S \rightarrow A S C | A$
 can be written as below:
 because A and C are part of the grammar produced by
 the subfiring generated from a production of endstate
 $A \rightarrow B A C C | E$
 B_m^m can be generated from the following production,
~~production~~
 $\begin{array}{c} A \\ \diagdown \quad \diagup \\ a_m b_m c_m \\ \diagup \quad \diagdown \\ a_m b_m c_m \\ \vdots \\ \diagup \quad \diagdown \\ a_m b_m c_m \end{array} = L$
 $\therefore L = a_m b_m c_m$
~~as~~ given that $a + b + c = m$
 $L = \{a^m b^m c^m \mid m \geq 0, m \in \omega\} \quad (6)$

$A \rightarrow O A | I A | E$
 $S \rightarrow A O O A$
 generated using the production of the form:
 strings of O 's and I 's of any length which can be
 subfiring $O O O$ can be preceded and followed by
~~so~~ If it is clear from the definition that the
 strings of O 's and I 's having a subfiring $O O O$
 $(8) \text{Obtain a grammar to generate a language}$

$B \rightarrow a | ab$
 $A \rightarrow aa | baa | AA | Aa |$
 $S \rightarrow AB | BA | ABA$
 The grammar is

$B \rightarrow a | ab \text{ by defining } B$
 $\left. \begin{array}{l} A \rightarrow e \\ A \rightarrow AA \\ A \rightarrow bAa \\ A \rightarrow AAB \end{array} \right\} \text{Defining } A$

$S \rightarrow ABA$ [one as more is in the middle]
 $S \rightarrow BA$ [one as more is at the beginning]
 $S \rightarrow AB$ - [extra a generated at the end]

a) Both generate equal no. of a's and b's —
 b) Equal no. of a's as at the end is
 by same as in the middle

These will make the grammar by putting the
 language into two parts
 1 as inside a's
 since numbers of a's should be in a position to generate
 numbers of b's, we should be greater than

so $L = \{w | n_a(w) < n_b(w)\}$

$S \leftarrow a \mid b \mid I$
 followed by number N
 An integer can be a number as sign (of plus or minus)
 $(N \leftarrow D)$
 A digit is a number followed by a digit as a number
 $D \leftarrow N$ follows;
 A number N can be recursively defined as
 $I \leftarrow 0 \mid 1 \mid 2 \mid \dots \mid 9$
 digits can be written as:
 0, 1, 2, ..., 9. The product of any of the digits
 A no. can be formed from any of the digits
 numbers would not be generated.
 These, if it is derived from S, the sign for a
 The product for this can be written as:
 S1) The sign of a no. can be:
 $+ \mid - \mid \times \mid / \mid \div \mid =$
 1965 and 8965 numbers + 1965.
 Shows the division for the unsigned numbers
 98) Define a grammar to generate integers.

$$\begin{aligned}
 A &\leftarrow C \\
 A &\leftarrow AA \\
 A &\leftarrow b \mid a \\
 A &\leftarrow ab
 \end{aligned}$$

$$S \leftarrow bA \mid A\bar{b} \mid ABA$$

$$l_1 + (m_1 m_2) = (m_1 m_2) l_1 = l_1 \quad (1)$$

9961 ←
9960 ←
9962 ←
9963 ←
9964 ←
9965 ←
9966 ←
9967 ←
9968 ←
9969 ←
9970 ←
9971 ←

The unsigned number 1961 and signed number
1965 can be shown as below:

6 | ... | 8 | 1 | 0 ← D
5 | - | 1 | + | ← S
4 | 0 | 2 | N ← N
3 | 2 | 2 | ← T

The final examination

$$\{ \omega | \omega \in \{0\}^{\omega} \} \leq L$$

$$\{ \omega | \omega \in \{0,1\}^{\omega} \} \leq L$$

Both will show ϵ and strings consisting of 0's can be generated from the grammar.

Alternatively, we are applying the productions of $S \leftarrow e$, we get strings consisting of 0's and 1's and finally applying the production $S \leftarrow A$ and $A \leftarrow s$

$$\begin{array}{ll}
 [e \leftarrow S] & 1010 \Leftarrow \\
 [S \leftarrow A] & S1010 \Leftarrow \\
 [A \leftarrow s] & 01010 \Leftarrow \\
 [S \leftarrow B] & S10 \Leftarrow \\
 [B \leftarrow S] & B0 \Leftarrow S
 \end{array}$$

$$\begin{array}{l}
 A \leftarrow s \\
 S \leftarrow A | e
 \end{array}$$

1) What is the language generated by the grammar.

$L = \{w \cdot w^R \mid w \in (a+b)^*\}$
 Q we derived by w
 we get a string w followed by some

$$\begin{aligned}
 & [S \leftarrow S] \Leftrightarrow aabbbaa \\
 & [q_5 q \leftarrow S] \Leftrightarrow aaabbbaa \\
 & [q_5 q \leftarrow S] \Leftrightarrow aaabbaa \\
 & [S \leftarrow q_5 q] \Leftrightarrow aaasa \\
 & [S \leftarrow q_5 q] \Leftrightarrow asa
 \end{aligned}$$

$$S \Leftarrow S \quad \text{Top}$$

What is the language generated by this grammar?
 a) Consider the following grammar:

Deviation

3) Righthand deviation :- The process of obtaining a string of terminals from a sequence of symbols such that only rightmost non-terminal is replaced at each and every step.

4) lefthand deviation :- The process of obtaining a string of terminals from a sequence of symbols such that only leftmost non-terminal is replaced at each and every step.

Types of deviation :-

A $\rightarrow x_1 x_2 x_3 \dots x_n$ must be a product in P
 as all children of A from left, then
 If a vertex is a child of A and if $x_1 x_2 x_3 \dots x_n$
 in x_1 vertex has a label from V
 Every leaf node has a label from T and an
 Every vertex has a label which is in (VUT)
 The root has the label S

following properties :-

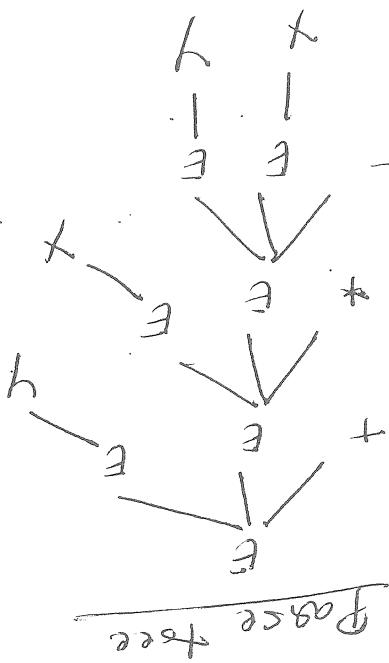
The deviation can be defined with the

grammar.

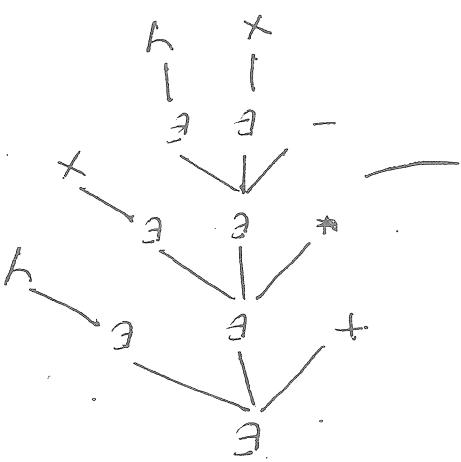
Definition :- Let $G = (N, T, P, S)$ be a context free grammar.
 A tree

Deviation can be shown in the form of

Parse Tree :-



Rightmost derivation

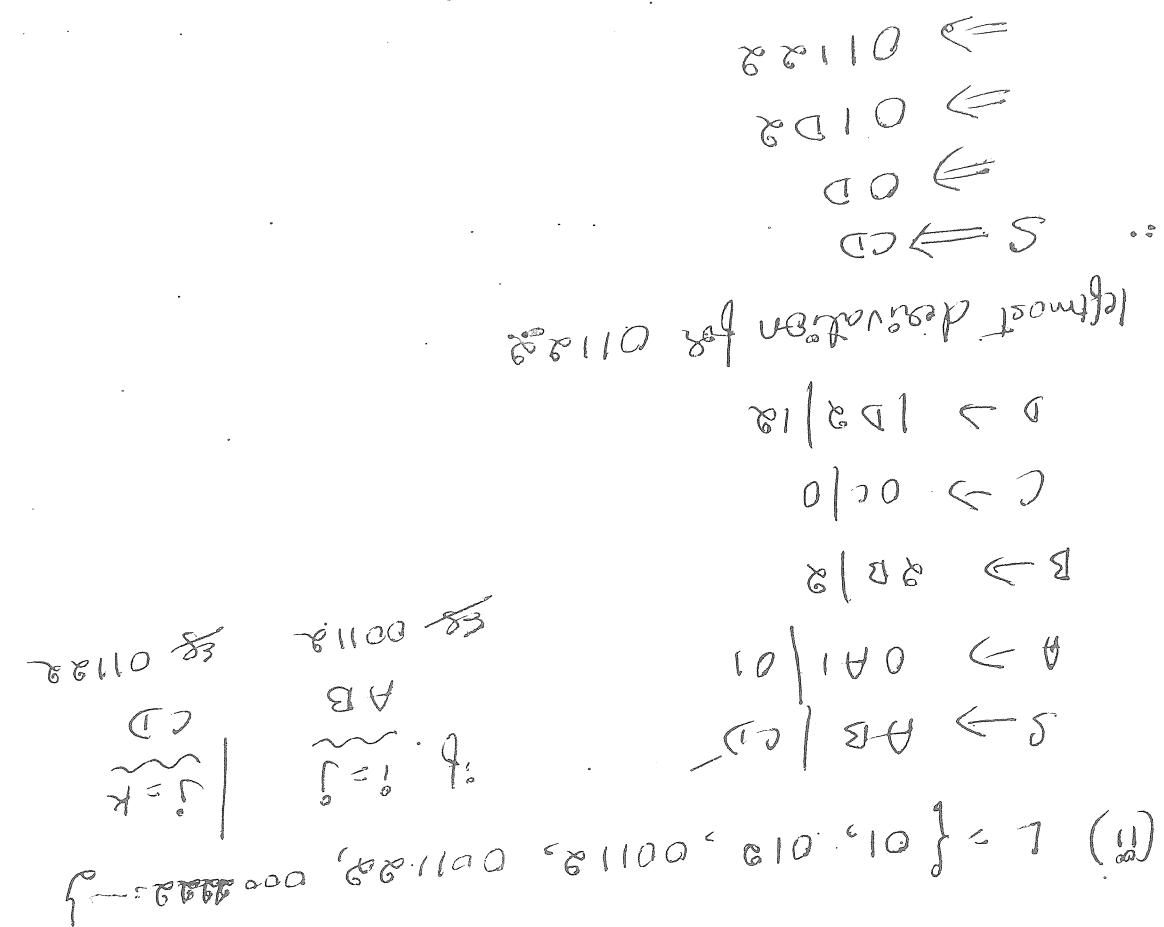
$$\begin{aligned} &L \xrightarrow{*} x y * \\ &\xrightarrow{*} x y E - \\ &\xrightarrow{*} x y E E \\ &\xrightarrow{*} x E x y \\ &\xrightarrow{*} x E E y \\ &\xrightarrow{*} x E y \\ &\xrightarrow{*} E y \\ &\xrightarrow{*} E E \end{aligned}$$


Leftmost derivation

$$\begin{aligned} &L \xrightarrow{*} x y * \\ &\xrightarrow{*} x y E - \\ &\xrightarrow{*} x y E E \\ &\xrightarrow{*} x E x y \\ &\xrightarrow{*} x E E y \\ &\xrightarrow{*} x E y \\ &\xrightarrow{*} E y \\ &\xrightarrow{*} E E \end{aligned}$$

Change the grammar
Find leftmost and rightmost derivation for the string
 $+ * - x y x y$ and write parse tree

$$E \leftarrow + E E | * E E | - E E | x | y$$



(i) $L = \{e, ab, aaaaabbb, aaaa...bb, \dots\}$

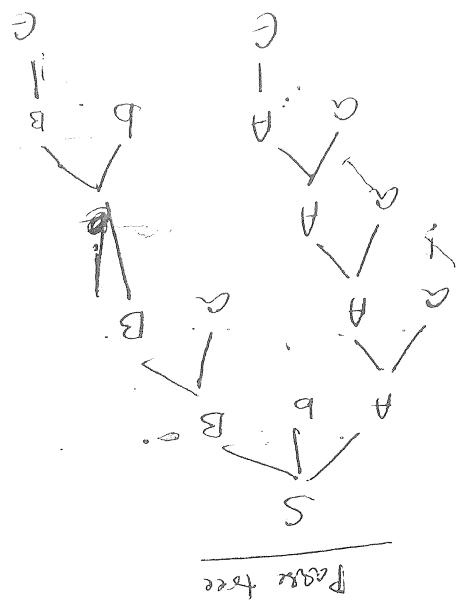
SL

generate leftmost derivation for the string 011122.

(ii) $L = \{0^i 1^j 2^k \mid i=j \text{ as } j=k\}$ and

(i) $L = \{a^n b^m \mid n \geq 0, m \geq 0\}$

a) construct CFG for the following language



Page 288

leftmost desolution

Give leftmost derivation, rightmost derivation and parse tree for the string $aabb$.

$$E \hookrightarrow \mathcal{A}B / \mathcal{B}B$$

$A \rightarrow \alpha A / e$

$S \leftarrow ABB$

3) Consider the gamma 6, with production:

$$c * (a+b) \Leftarrow$$

$$c * (b+e) \Leftarrow$$

$$c * (e+e) \Leftarrow$$

$$c * (e) \Leftarrow$$

$$c * e \Leftarrow$$

$$e * e \Leftarrow e$$

Postfix notation derivation

$$(c * a + b) \Leftarrow$$

$$(e * a + b * e) \Leftarrow$$

$$(e + e + a * e) \Leftarrow$$

$$(e + e + e) \Leftarrow$$

$$(e * e) \Leftarrow$$

$$(e) \Leftarrow e$$

Prefix notation derivation

$$c * (a + b)$$

$$(a + b * c)$$

(i) Observe the postfix notation derivation for the following

$$e \Leftarrow a | b | c$$

$$e \Leftarrow e$$

$$e \Leftarrow e | e * e$$

$$e \Leftarrow e + e | e - e$$

4) Generate the following grammar:

$$\begin{array}{ll}
 (0 - (0 * 1)) - 0 \Leftarrow & (0 - (0 * 1)) - 0 \Leftarrow \\
 (0 - (0 * 1)) - f \Leftarrow & (f - (0 * 1)) - 0 \Leftarrow \\
 (0 - (0 * j)) - j \Leftarrow & \vdots \quad (j - (0 * 1)) - 0 \Leftarrow \\
 (0 - (0 * \perp)) - \perp \Leftarrow & (\perp - (0 * 1)) - 0 \Leftarrow \\
 (0 - (0 * e)) - E \Leftarrow & (\perp - (E * 1)) - 0 \Leftarrow \\
 (0 - (E * 1)) - \perp \Leftarrow & (\perp - (\perp * 1)) - 0 \Leftarrow \\
 (0 - (\perp * e)) - \perp \Leftarrow & (\perp - (\perp * \perp)) - 0 \Leftarrow \\
 (0 - (e)) - \perp \Leftarrow & (\perp - (\perp * E)) - 0 \Leftarrow \\
 (0 - \perp) - \perp \Leftarrow & (\perp - (E)) - 0 \Leftarrow \\
 (\perp - \perp) - \perp \Leftarrow & (\perp - (\perp)) - 0 \Leftarrow \\
 (\perp - E) - \perp \Leftarrow & (\perp - (E)) - 0 \Leftarrow \\
 (\perp) - \perp \Leftarrow & (\perp) - 0 \Leftarrow \\
 (E) - \perp \Leftarrow & (E) - 0 \Leftarrow \\
 \perp - \perp \Leftarrow & \perp - 0 \Leftarrow \\
 \perp - E \Leftarrow & \perp - \perp \Leftarrow \\
 \perp \Leftarrow E & \perp \Leftarrow E
 \end{array}$$

Ed: leftmost derivation

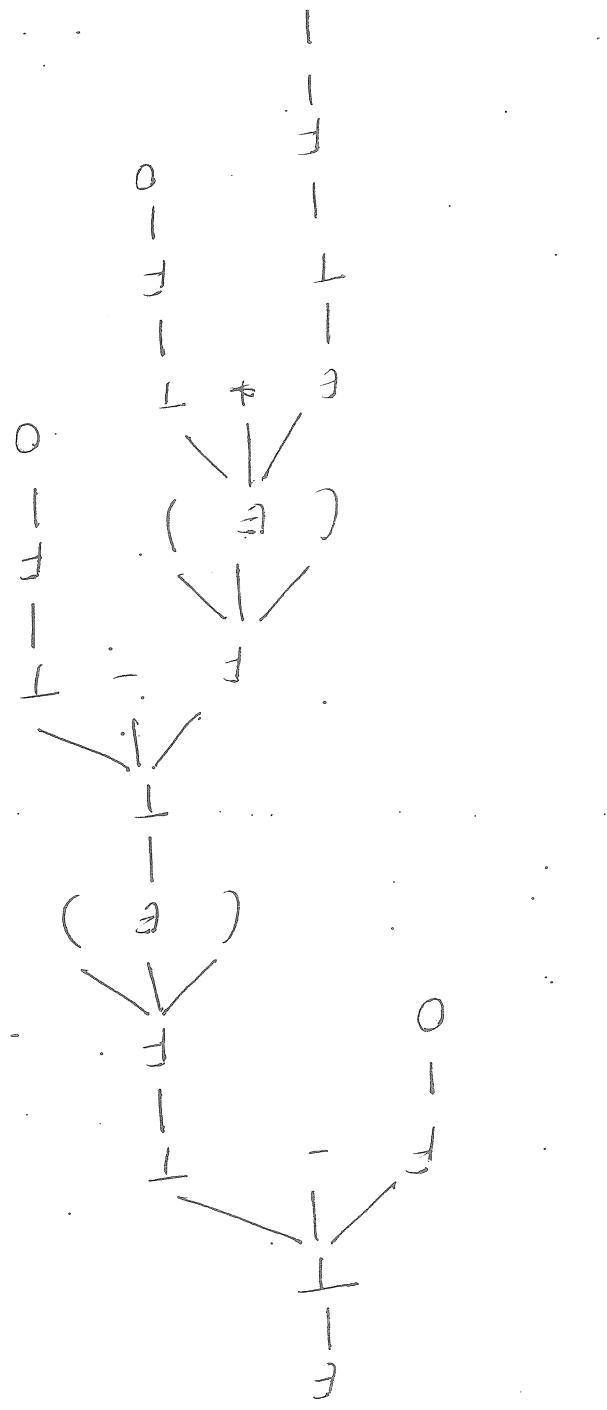
make the leftmost derivation, rightmost derivation and
passive tree for the string, $0 - ((10) - 0)$

$$F \leftarrow (E) | 0 |$$

$$T \leftarrow F - T | F$$

$$E \leftarrow E * T | T$$

5) Longest derivation with productions



Page tree

Q.7.0

Q.7.0

Show that the grammar is ambiguous.

$E \rightarrow E * E \mid E / E \mid id$

$E \rightarrow E + E \mid E - E$

Consider the following grammar:

[Apply the same procedure for rightmost derivation]

Obtain the string w by applying leftmost derivation

If there are two different parse trees for the same string w , then the parse trees are different, the grammar is ambiguous.

Obtain the leftmost derivation and get a string

Obtain the rightmost derivation and get a string

How to check the grammar is ambiguous or not?

If most derivation is right most derivation

Different parse trees exist by applying either the least fixup or E^{*} to see which tree is more

G is ambiguous if and only if there exists at least one string w E T^{*} for which two or more different parse trees exist by applying either the least fixup or E^{*} to E^{*} to see which tree is more

Let G = (V, E, P, S) be a CFG. A grammar

Ambiguous grammar

Since there are two different pages for the same subject, the ambiguous language of grammar is ambiguous.

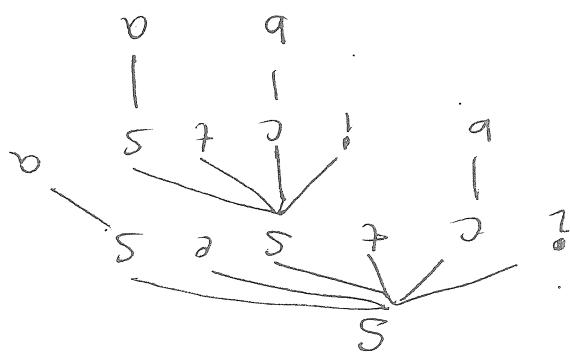
$$\begin{array}{c}
 p_0^! & p_0^! \\
 | & | \\
 | & | \\
 E + E & \\
 | & | \\
 | & | \\
 E & * & E \\
 | & | & | \\
 E & E & E
 \end{array}
 \quad
 \begin{array}{l}
 p_0^! * p_0^! + p_0^! \Leftarrow \\
 E * p_0^! + p_0^! \Leftarrow \\
 E + E + p_0^! \Leftarrow \\
 E + E + E \Leftarrow \\
 E * E \Leftarrow \\
 \hline
 \end{array}$$

The same spring $\frac{d}{dx} p! + p! \frac{dp}{dx}$ is obtained by applying different drugation and using different parameters as shown below

$$\begin{array}{c}
 p_1! \quad p_2! \\
 | \quad | \\
 E \quad E \\
 | \quad + \quad | \\
 | \quad \quad \quad | \\
 E \quad + \quad E \\
 | \quad \quad \quad | \\
 E \quad E
 \end{array}
 \qquad
 \begin{aligned}
 p_1! &= p_1 + p_1^? \leq \\
 E + p_1^? + p_1^? &\leq \\
 ? + E + p_1^? &\leq \\
 ?d + E + p_1^? &\leq \\
 ?d + E &\leq \\
 E &\leq E + E
 \end{aligned}$$

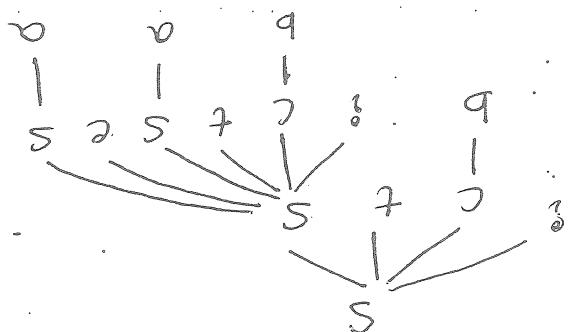
The spring $P_1 + P_2$ can be obtained by applying leftmost deletion as shown below.

Since there are two different paths free for the string, it can be obtained by applying leftmost derivation.



$S \Rightarrow S \cup S$
 $S \Rightarrow S \cup S$

The same string can be obtained by applying the leftmost derivation with the help of the following diagram:



$S \Rightarrow S \cup S$
 $S \Rightarrow S \cup S$

So, The string $abab$ can be obtained by applying the leftmost derivation as shown below:

$C \leftarrow b$

$S \leftarrow Idts | Idtses | a$

If the following grammar ambiguous?

The method ϕ means if statement has the past and intended part means if statement has past.

These can be matched part and intended because if.

Normally else is matched with closest elimination of ambiguity:

To solve this, consider unambiguous

How dangling else problem is solved?

"dangling else problem"

This ambiguity occurs to allocate else with first if-statement of second if-statement is called

The "second part free association else with"

The first part free association with if-statement

What is dangling-else problem? Different parts trees exist for same styling since two followings points:

The dangling else problem can be eliminated And thus ambiguity of the grammar can also be eliminated

The grammar cross-embedding of if-statement is ambiguous. This is due to dangling-else

The grammar cross-embedding of if-statement

if-ambiguity Rule:-

$$L_2 = \{a^n b^m c^l \mid n, m, l \geq 0\}$$

The language L_2 can be generated from non-terminal symbols S_1, A, B, C :

$$S_1 \rightarrow C B / E$$

$$A \rightarrow a A b / E$$

$$B \rightarrow A B$$

$$C \rightarrow A$$

$$L_2 = \{a^n b^m c^l \mid n, m, l \geq 0\}$$

The language L_1 can be generated from non-terminal symbols S_1, S_2, A, B, C :

$$S_1 \rightarrow S_2 / S$$

using the following production:

The language L_1 as L_2 can be generated

$$L_1$$

$$L = \{a^n b^m c^l \mid n, m, l \geq 0\} \cup \{a^n b^m c^l \mid n, m \geq 0\}$$

QED! The given language can also be written as:

$$L = \{(a^i b^j c^k \mid i, j, k \geq 0\} \text{ and } \{a^i b^j \mid i > j\} \cup \{a^i b^j c^k \mid i, j, k \geq 0\}$$

inherently ambiguous language

Obtain the inherently ambiguous grammar for the following:

Inherently ambiguous grammar :-

$$U \leftarrow ? c t m e$$

$$U \leftarrow ! c t s$$

$$W \leftarrow ? c t m e$$

$$S \leftarrow M U$$

Hence the ambiguous grammar will be:

$\rightarrow D \mid e$
 $C \leftarrow a \mid e$
 $B \leftarrow cB \mid e$
 $A \leftarrow aAb \mid e$
 $C_2 \leftarrow d_1$
 $S_1 \leftarrow AB$
 $S \leftarrow S_1 S_2$

The final Ambiguity ambiguous grammar can
be written as shown below:

$\rightarrow D \leftarrow bDc \mid e$
 $C \leftarrow aCc \mid e$
 $C \leftarrow CS$

$\Leftarrow abcd$
 $\Leftarrow abB$
 $S \Leftarrow AB$

The string $abcd$ can be derived from the grammar as shown below:

$\begin{array}{l} S \xrightarrow{} AB \\ A \xrightarrow{} AB \mid ab \\ B \xrightarrow{} CBd \mid cd \\ C \xrightarrow{} ADD \mid ADd \\ D \xrightarrow{} bDC \mid bc \end{array}$

So, The grammar to generate the language is:

$\{ a^n b^n c^n d^n \mid n \geq 1 \}$

$\{ a^n b^n c^n d^n \mid n \geq 1, y = 1 \}$

for the following inherently ambiguous language.

Example: Between the inherently ambiguous grammar and the inherently unambiguous grammar:

The language is derived as called inherently ambiguous language and the grammar is called inherently ambiguous, then the language is called inherently ambiguous language L , for which there is no unique derivation of more distinct derivation trees of this type.

Some strings in $L(G)$ for which there are two different derivations are these two strings w and x .

A grammar is said to be inherently ambiguous if these extra

Inherently ambiguous grammar:

The same string abc can be derived by applying different set of permutations as shown below:

So, if we write the same trees for both the derivations, the same trees are different and mutually
the gammas are ambiguous.
It is not possible to obtain the unambiguous gamma for this and so it is inherently ambiguous.

$$\begin{array}{c} \Leftarrow abcd \\ \Leftarrow abdc \\ \Leftarrow cdab \end{array}$$

The same string abc can be derived by applying different set of permutations as shown below:

Eliminating ambiguity using Precedence of Association

The grammar can be converted into unambiguous grammars using the precedence of operators as well as associativity of operators as shown below:

Operators		Associativity	Precedence
P	Right		✓
T	Left		/,*
E	Left		-,+

Step 1: Assign the operators to increasing precedence along with associativity as shown below:

Step 2: Associate the operators as shown below:

Step 3:- The basic units in expression are id (identifier) and parenthesis enclosed expressions (subexpression) and product operation can be used when as the production expanding to this can be

Step 3:- The next highest priority operator is as a basic unit in expression.

Step 3:- The next highest priority of expression is as a basic unit in expression.

And it is right associative. So the term P and it must start from the left. Term P and it should have right association.

Step 4: The next highest priority operators are * and / and they are left associative. So, the product term must start from the non-terminal T and it should have left recursion.

$$T \rightarrow T * P \mid T / P$$

Step 5: The next highest priority operators are + and - and they are left associative.

$$E \rightarrow E + T \mid E - T$$

So the product term must start from the non-terminal E and it should have left recursion.

Step 6: The final grammar looks like

$$\begin{aligned} E &\rightarrow (E) \mid id \\ P &\rightarrow f \mid p \mid E \\ T &\rightarrow T * P \mid T / P \\ E + T &\rightarrow E - T \end{aligned}$$

unambiguous can be written as :

Since there are three levels of precedence we can extra non-terminal F generalizing basic units in allude three non-terminals: E , P and T . Also we can continue expression.

	T	$Right$	$/$	$(logest)$
	P	$Left$	\vee	
	E	$Left$	$-$	$(logest)$
<u>precedence</u>	<u>operator</u>	<u>associativity</u>	<u>non-terminal</u>	

of the precedence along with associativity
Step 1: Rearrange the operators to increase reading order
associative.

A operator has precedence in between and it is left associative
+ and \times / operator have the logest priority and
they are left associative
by considering * and - operators lowest priority and

$$E \leftarrow (E)^\dagger$$

$$E \leftarrow E/E$$

$$E \leftarrow E+E$$

$$E \leftarrow E-E$$

$$E \leftarrow E+E$$

unambiguous grammar

1) Convert the following ambiguous grammar into

$$\begin{array}{c}
 E \xrightarrow{\text{id}} (E) \\
 | \quad | \\
 E \xrightarrow{\text{id}} E + T \\
 | \quad | \\
 T \xrightarrow{\text{id}} P \cdot T \\
 | \quad | \\
 E \xrightarrow{\text{id}} E * P
 \end{array}$$

ie written as :

The final grammar which is unambiguous can

$$E \xrightarrow{\text{id}} E * P \mid E - P \mid P$$

Step 5 :- The next highest priority operators are and - and they are left associative. So, the production must start from the non-terminal E

$$P \xrightarrow{\text{id}} E \cdot T \mid T$$

Step 6 :- The next highest priority operator is \star and it is left associative.

$$T \xrightarrow{\text{id}} E + T \mid E \cdot T \mid E$$

Step 3 :- The next highest priority operators are $+ \text{ and } \cdot$ and they are right associative. So,

$$E \xrightarrow{\text{id}} (E)$$

Step 8 :- The basic units in expression are id and parenthesis based expression.

3) $F \leftarrow (E) id$

4) $T \leftarrow F T$

5) $E \leftarrow T E$

$A_i \leftarrow \alpha_1 A_i | \alpha_2 A_i | \alpha_3 A_i | \dots | \alpha_n A_i | E$

$A \leftarrow B_1 A | B_2 A | B_3 A | \dots | B_m A$

Right Recursive Production General Form:

3) $F \leftarrow (E) id$

4) $T \leftarrow F \alpha_1 | \beta_1$
 ↑
 ↑
 ↑
 |
 5) $E \leftarrow T E$

1) $E \leftarrow T$
 ↑
 ↑
 ↑
 |
 2) $A \leftarrow A \alpha_1 | A \alpha_2 | A \alpha_3 | \dots | A \alpha_n | \beta_1 | \beta_2 | \beta_3 | \dots | \beta_m$

Left Recursive Production General Form:

It can be eliminated from the grammar as shown below:
Goal: The grammar has immediate left recursion.

3) $F \leftarrow (E) id$

4) $T \leftarrow F T \leftarrow T$

5) $E \leftarrow E + T T$

Eliminate left recursion from the foll. grammar.

Now, A-production has left recursion and can be eliminated as shown below:

$$A \leftarrow A b \mid A b a \mid a$$
$$S \leftarrow A b \mid a$$

So, the given grammar can be written as:
can eliminate the indicated left recursion from S.
Substituting for S in the A-production, we

$$S \leftarrow S a b$$

$$S \leftarrow A b \in S a b$$

Recursion. Since,

Solving immediate left recursion, it has left
SOL: The non-terminal S, even though is not
showing immediate left recursion, it has left

$$A \leftarrow A b \mid S a$$

$$S \leftarrow A b \mid a$$

3) Eliminate left recursion from the foll. grammar

$$F \mid (E) \leftarrow F$$

$$T \mid * F T \mid E$$

$$T \mid F T$$

$$E \mid + T E \mid E$$

$$E \leftarrow T E$$

left recursion can be written as shown below:
The final grammar obtained after eliminating

$$A \rightarrow AaA$$

$$S \leftarrow Ab|a$$

$$A \leftarrow bA|baA|e$$

left derivation can be written as:
 The final grammar obtained after eliminating

$$A \leftarrow \alpha_1 A | \alpha_2 A | e$$

$$A \leftarrow bA | baA | e$$

$$\uparrow \quad \uparrow \quad \uparrow$$

$$A \leftarrow AaA$$

$$A \leftarrow B_A$$

$$\uparrow \quad \uparrow$$

$$S \leftarrow Ab|a \quad // \text{No left recursion}$$

$$A \leftarrow \alpha_1 A | \alpha_2 A | \alpha_3 A | \dots | \alpha_n A | e$$

$$A \leftarrow B_1 A | B_2 A | B_3 A | \dots | B_m A$$

Right recursive products

$$A \leftarrow Aa \quad A \leftarrow A\alpha_1 | A\alpha_2 | \dots | A\alpha_n | B_1$$

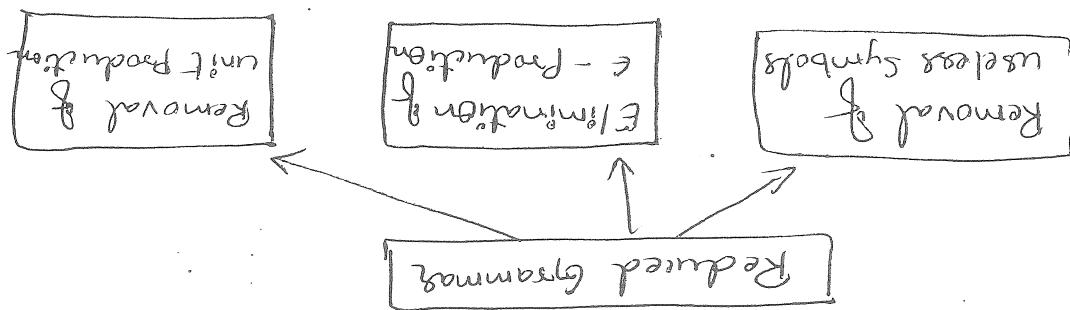
$$A \leftarrow Ab \quad A \leftarrow A\beta_1 | A\beta_2 | \dots | A\beta_m | B_1$$

$$S \leftarrow Ab|a \quad // \text{No left recursion}$$

$$A \leftarrow A\alpha_1 | A\alpha_2 | \dots | A\alpha_n | B_1 | \dots | B_m$$

Left recursive products

Two algorithms are used to eliminate useless variables



3) If E is not in the language L then there need not be the production $x \rightarrow E$

g) There should not be any production as $x \rightarrow y$ where x and y are non-terminals.

d) Each variable (non-terminal) and each terminal y appears in the derivation of some word in L .

The properties of reduced grammar are given below:-

by removing useless symbols.

Elimination of grammar means reduction of grammar by symbols unneccessary increases the length of grammar. Some extra symbols (non terminals). Having extra always offend. That means grammar may consist by context free grammar. All the grammars are not various languages can effectively be represented

Simplifying CFGs :-

1. $G' = G$
- remove unneeded symbols:
- a. mark every terminal symbol in G' as productive
 - b. mark every non-terminal symbol in G' as unproductive
 - c. until one entire pass has been made without any new symbol being marked do:
 - i) for each rule $X \rightarrow \alpha AB$ (where $A \in V - E$) in R do:
 - a) if X has been marked as productive and A has not then: Mark A as productive.
 - b) remove from G' , every unneeded symbol.
 - ii) for each rule $X \rightarrow \alpha A$ do:
 - a) if X has been marked as productive and A has not then: Mark A as productive.
 - b) remove from G' , every unneeded symbol.
 - d. until no new symbol is marked do:
 - i) for each rule $X \rightarrow \alpha A$ do:
 - a) if X has been marked as productive and A has not then: Mark A as productive.
 - b) remove from G' , every unneeded symbol.
 - ii) for each rule $X \rightarrow \alpha A B$ (where $A \in V - E$) in R do:
 - a) if X has been marked as productive and A has not then: Mark A as productive.
 - b) remove from G' , every unneeded symbol.

2. $G' = G$
- remove unproductive symbols:
- a. mark every terminal symbol in G' as productive
 - b. mark every non-terminal symbol in G' as unproductive
 - c. until one entire pass has been made without any new symbol being marked do:
 - i) for each rule $X \rightarrow \alpha A B$ (where $A \in V - E$) in R do:
 - a) if X has been marked as productive and A has not yet been marked as productive and X has not yet been marked as productive then:
 - i) if every symbol in α has been marked as productive then:
 - a) for each rule $X \rightarrow \alpha A$ do:
 - a) if X has been marked as productive and A has not yet been marked as productive then:
 - i) if every symbol in α has been marked as productive then:
 - a) for each rule $X \rightarrow \alpha$ do:
 - a) if X has been marked as productive then:
 - i) if every symbol in α has been marked as productive then:
 - a) remove from G' , every unneeded symbol.

remove unproductive symbols:

remove unproductive symbols ($G' \cap F(G)$):

Simplification of grammar:

Even though there is no substitution in the right-hand side of the production for any CFG, it is better in fact necessary to eliminate some of the useless symbols and products.

In the grammar G, some of the symbols do not need to derive a string. In the grammar, some symbols and products may not be used if products. Some symbols and products may not be used and will never be used as useless deriving a string. So, these symbols and products which will never be used are useless and the grammar G, some of the symbols can be removed and the corresponding products can be removed.

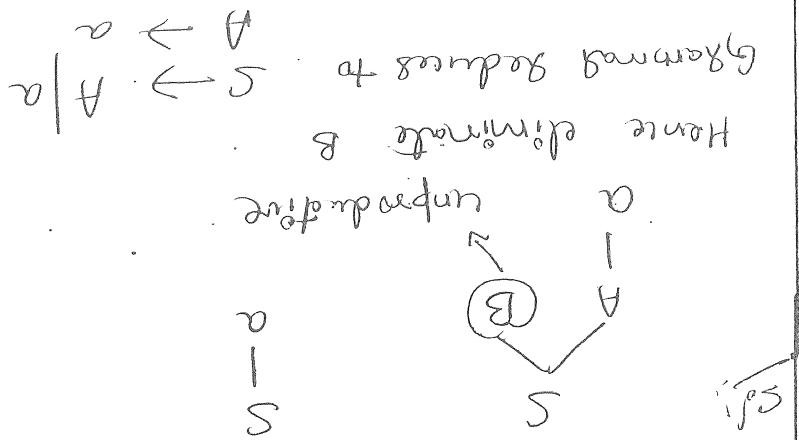
In two grammars, if we apply the production $S \rightarrow B$, A string can never be derived. So, $S \rightarrow B$, A symbol can never be derived. So, symbols in V must satisfy following conditions:

Now we'll see, how to eliminate symbols and can be eliminated.

Symbols in V must satisfy following conditions:

- Symbols in V must be derived.
- Symbols in V must not applying in any production of the form $A \rightarrow B$ if the symbol B is not derived.

Simplification of grammar:



a) Determine the useless symbol:

~~The parser can't reach the final state~~

and the following A-productions are shown below

The RHS of the production containing a non-terminal B can be replaced by the production $B \rightarrow ab/b$

So, consider the production:

Simplify the grammar by substitution method.

$$B \rightarrow ab/b$$

$$A \rightarrow aBa$$

i) Consider the products

$$S \rightarrow aB_a \quad | \quad BC$$

$$A \rightarrow a_c \quad | \quad BE$$

$$B \rightarrow bcc$$

$$C \rightarrow a$$

$$D \rightarrow E$$

$$E \rightarrow d$$

$$S \rightarrow AB \quad | \quad CA$$

$$B \rightarrow BC \quad | \quad AB$$

$$C \rightarrow AB \quad | \quad CB$$

$$A \rightarrow a$$

$$B \rightarrow b$$

$$S \rightarrow AS \quad | \quad A$$

The simplified rules are:

$$\begin{array}{c} A \\ | \\ S \leftarrow AS \quad | \quad A \\ | \\ S \leftarrow S \\ | \\ S \leftarrow S \end{array}$$

Productive

$\{B\}$ is unreachable

$$\begin{array}{c} A \\ | \\ S \leftarrow A \\ | \\ S \leftarrow AS \\ | \\ S \end{array}$$

Reachable

$\{S, A\}$ are reachable
others

\Rightarrow check for unproductive
~~or~~ check for unreachable

$$S \rightarrow AS \quad | \quad A$$

$$A \rightarrow a$$

$$B \rightarrow aa$$

Let $G = (V, T, P, S)$ be a CFG, if Production
 in P of the form $A \rightarrow E$ is called an
 E -Production. If it is also called Null Production.
 If we apply the production $A \rightarrow E$, then in the
 derivation, the variable A is replaced by E
 i.e. the variable A is erased. Each A in V
 there is a derivation of the form $A \Rightarrow^* E$, then
 E is a nullable variable.
 A nullable variable is defined as follows:
 \leftarrow If $A \rightarrow E$ is a production in P , then A
 is a nullable variable.
 \leftarrow If $A \rightarrow B_1 B_2 \dots B_n$ is a production in P ,
 and if B_1, B_2, \dots, B_n are nullable variables,
 then A is also a nullable variable.
 \rightarrow The variables for which there are products
 of the form $A \rightarrow E$ known in previous two steps are
 \rightarrow The variables for which there are products

nullable variables
 In a CFG, unless an empty string is derived
 from the start symbol. Suppose, the language
 generated from a grammar to does not derive
 any empty string & the grammar consists of
 productions. Such a production can be removed.
 What is an E -production? If so it is a nullable
 E -production. Suppose, the language

| | | |
|--|--|---|
| $D \rightarrow EA$
$A \rightarrow AA$
$S \rightarrow AA$

$E \rightarrow d$
$D \rightarrow ab$
$A \rightarrow a$

<u>Productions</u> | A, D, E, S

$\frac{m}{\Delta}$ | A, D, E, S

$\frac{\phi}{\Delta}$ |
|--|--|---|

From which we get only string of terminals.

Stage 1 :- Find the set of symbols and productions.

using too stages

Q1:- The useless symbols can be eliminated

$$\begin{array}{l} E \rightarrow ad | d \\ D \rightarrow ab | Ea \\ B \rightarrow bB \\ A \rightarrow AA | a \\ S \rightarrow AA | bB \end{array}$$

II) Eliminate useless symbols in the following

In other words, in a derivation, finally we should get string of terminals and all these symbols used in the derivation must be derivable from the start symbol \$.

These symbols used in the derivation must be derivable from the start symbol.

From each \$, we should be in a position to get string of terminals and \$ must be from each \$, we should be in a position to get string of terminals and \$ must be derivable from the start symbol.

In other words, if a derivation is

otherwise, the symbol \$ is useless. That is,

$$S \Leftarrow \alpha \times \beta \Leftarrow \in$$

derivation of the form

A symbol \$\times\$ is useful if there is a

such that is a useless symbol?

$$\begin{aligned}
 D &\leftarrow \text{add} \\
 C &\leftarrow \text{CD} \\
 B &\leftarrow a/Aa \\
 A &\rightarrow AB \\
 S &\leftarrow AA | a | Bb | CC
 \end{aligned}$$

Q) Simplify the following grammar
Assignment:

S is the start symbol

$$\begin{aligned}
 A &\leftarrow a/Aa \\
 S &\leftarrow AA \\
 P &= \{ \dots \} \\
 T &= \{ a \} \\
 \{ S, A \} &= \underline{\cup}
 \end{aligned}$$

$G' = (V, T, P, S)$ where

The resulting grammar

| | | | |
|--------|----------------------|----------|--------|
| | $A \rightarrow a/Aa$ | a | S, A |
| | $S \rightarrow AA$ | a | S, A |
| S | - | - | - |
| \cup | \vdash | \vdash | P |

Step 8 :- Using the above grammar we obtain
the symbols such that each symbol \times is
reachable from the start symbol S

| | |
|---------------------------|--|
| $S \rightarrow ABCa$ | $S \rightarrow ABCa \mid BCa \mid ACa \mid ABA \mid Ca \mid Aa \mid Ba \mid a$ |
| $A \rightarrow BC \mid b$ | $A \rightarrow BC \mid c \mid B \mid b$ |
| $B \rightarrow b \mid e$ | $B \rightarrow b \mid e$ |
| $C \rightarrow c \mid e$ | $C \rightarrow c \mid e$ |
| $D \rightarrow d$ | $D \rightarrow d$ |

double one by one and add the resulting products
Take all combinations of nullable variables

$\therefore V = \{B, C, A\}$ are all nullable variables.

| Ω_V | new nullable vars | Products |
|-------------|-------------------|--------------------|
| - | - | B, C, A |
| B, C | B, C, A | $A \rightarrow BC$ |
| \emptyset | B, C | $B \rightarrow e$ |
| - | - | $C \rightarrow e$ |

from the grammar:

So:- Obtain the set of nullable variables

$$\begin{aligned}
 D &\leftarrow d \\
 C &\leftarrow c \mid e \\
 B &\leftarrow b \mid e \\
 A &\leftarrow BC \mid b \\
 S &\leftarrow ABCa \mid bD
 \end{aligned}$$

1) Eliminate E-products from the foll. grammar.

Leftmost derivation E-products

Resulting products are the final grammar

| | | |
|--------------|--|--------------------|
| | $B \rightarrow 1B 1$ | |
| | $B \rightarrow AB BA$ | |
| | $A \rightarrow \alpha A \alpha \alpha \beta$ | |
| | $A \rightarrow \beta A \beta \beta \beta$ | |
| | $S \rightarrow BAA ABA BAB ABA BBS$ | Resulting products |
| $\alpha \nu$ | | |

Take all the combination of nullable variables in a production, delete subset of nullable variables one by one and add the resulting products

| | | |
|--------------|--------------------------|-------------|
| | A, B, S | A, B, S |
| | $S \rightarrow BAA$ | |
| | A, B | A, B |
| | $A \rightarrow \epsilon$ | \emptyset |
| $\alpha \nu$ | | |

Goal: Obtain the set of nullable variables from the grammar

$$\begin{aligned} S &\rightarrow BAA \\ A &\rightarrow \alpha A \beta | \beta A \beta | \epsilon \\ B &\rightarrow AB | 1B | \epsilon \end{aligned}$$

g) Implement all E-products from the grammar.

Eliminating unit products

Let $G = (V, T, P, S)$ be a CFG. Any what is an unit product?

production in G of the form:

where $A, B \in V$ is a unit production. In any grammar, the right products are undisable. This is because one variable is simply replaced by another variable.

Consider the foll. grammar:

In the above grammar, the production:

a. unit production

$A \rightarrow B$

$B \rightarrow AB|b$

The productions: $B \rightarrow AB$ are non-unit products

There :- Let $G = (V, T, P, S)$ be a CFG if has unit productions and no C-products. An equivalent grammar G' is obtained such that $L(G) = L(G')$.

can be defined such that

[Unit products]

It is clear from the dependency graph that all non-unit products from E can also be generated from D

The resulting D products are:

$$\begin{aligned} E &\leftarrow D/Ab \\ D &\leftarrow bc \\ B &\leftarrow b \\ A &\leftarrow a \\ C &\leftarrow AB \end{aligned}$$

The non-unit products of the gamma are:



Dependency graph for the unit products is:

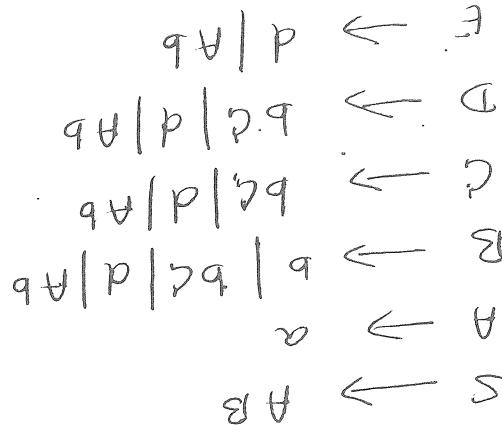
$$\begin{aligned} E &\leftarrow D \\ C &\leftarrow D \\ B &\leftarrow C \end{aligned}$$

shown below:

Sol: The unit products of the gamma are

$$\begin{aligned} E &\leftarrow a/Ab \\ D &\leftarrow e/bc \\ C &\leftarrow D \\ B &\leftarrow C/b \\ A &\leftarrow a \\ S &\leftarrow AB \end{aligned}$$

D) Eliminate all unit products from the gamma



∴ The final grammar obtained after eliminating unit productions are :

$$\begin{array}{l}
 B \leftarrow d | Ab \\
 B \leftarrow bc \\
 B \leftarrow b
 \end{array}$$

∴ The resulting B productions are :

$$\begin{array}{l}
 C \leftarrow d | Ab \\
 C \leftarrow bc
 \end{array}$$

∴ The resulting C productions are :

∴ If we clear from the dependency graph that, all non-unit productions from E, D, C can be generated from

$A \leftarrow 0|1|1$
 $B \leftarrow 1|0|1|2$
 $C \leftarrow A0|1|0|1|2$
 Can be obtained by combining (1) and (2)
 The simplifying grammar without productions

$\textcircled{2} \longrightarrow \left\{ \begin{array}{l} A \leftarrow 1|1 \\ B \leftarrow 0|1|2 \\ C \leftarrow 1|0|1|2 \end{array} \right.$
 The new productions from S, A, B are:

$\textcircled{1} \longrightarrow \left\{ \begin{array}{l} A \leftarrow 0|1|2 \\ B \leftarrow 1|1 \\ C \leftarrow A0 \end{array} \right.$
 The new unit productions are:

The unit productions are:
 $S \leftarrow A$
 $S \leftarrow B$
 $A \leftarrow 0|1|2$
 $B \leftarrow 1|1$

Q) Eliminate unit productions from the grammar
 $S \leftarrow A0|B$
 $B \leftarrow A|1|1$
 $A \leftarrow 0|1|2|B$

$$\begin{array}{l}
 S \xrightarrow{} Aa | Ca | aB | b \\
 B \xrightarrow{} aB | b \\
 C \xrightarrow{} DB | D | ab \\
 D \xrightarrow{} d | ab \\
 E \xrightarrow{} ab
 \end{array}$$

Combining ① and ②

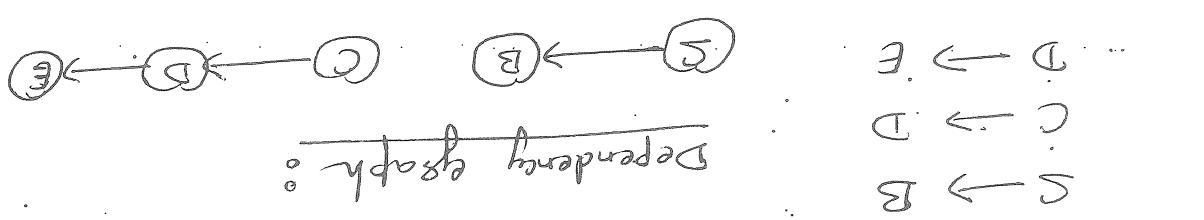
The non-unit productions can be obtained by

$$\textcircled{2} \quad \left. \begin{array}{l} D \xrightarrow{} ab \\ C \xrightarrow{} d | ab \\ S \xrightarrow{} aB | b \end{array} \right\}$$

The new productions from S, C, D are:

$$\textcircled{1} \quad \left. \begin{array}{l} E \xrightarrow{} ab \\ D \xrightarrow{} d \\ C \xrightarrow{} DB \\ B \xrightarrow{} aB | b \\ S \xrightarrow{} Aa | Ca \end{array} \right\}$$

The non-unit productions are:



Q1: The unit productions are:

$$\begin{array}{l}
 E \xrightarrow{} ab \\
 D \xrightarrow{} e | d \\
 C \xrightarrow{} DB | D \\
 B \xrightarrow{} aB | b \\
 S \xrightarrow{} Aa | B | Ca
 \end{array}$$

3) Eliminate unit productions from the grammar.

Normal forms

In a CFG, there is no restriction on the right-hand side of a production. The left-hand side of a production can impose conditions on the RHS of productions in various normal forms resulting in various normal forms.

The different normal forms that we discuss are:

- * Chomsky Normal form (CNF)
- * Greibach Normal form (GNF)

Let $G = (V, T, P, S)$ be a CFG. The grammar is said to be in CNF if all productions are of the form

$\text{where } A \in V \text{ and } a \in T$

$A \rightarrow a$

or

$A \rightarrow BC$

the form

Note that if a grammar is in CNF the RHS of a production should contain two symbols as one symbol must be non-terminal.

If there are two symbols on the RHS, those two symbols must be terminal symbols.

a terminal.

Note Before converting the grammar into CNF it should be in reduced form. That means remove all the useless symbols, E-productions and unit productions from it. Thus the reduced grammar can be converted to CNF.

$T \leftarrow b$
 $R \leftarrow QT$
 $A \leftarrow QR$
 $Q \leftarrow a$
 $P \leftarrow BR$
 $S \leftarrow AP$
CNF

$A \leftarrow ab$
 $S \leftarrow ABa$
SoL Producten

$B \leftarrow Ac$
 $A \leftarrow ab$
 $S \leftarrow ABa$

g) Convert the foll grammar into CNF

$A \leftarrow a$
 $P_4 \leftarrow AA$
 $S \leftarrow P_4 P_4$
 $P_3 \leftarrow AS$
 $P_2 \leftarrow AP_3$
 $P_1 \leftarrow AP_2$
 $S \leftarrow AP_1$

CNF

$S \leftarrow Aaaa$
Producten

$N.T. \leftarrow T$
 $N.T \leftarrow N.T \cdot N.T$

g) write the rule for CNF as

$S \leftarrow Aaaa$
 $S \leftarrow Aaaa$

i) convert the foll. CFG into CNF

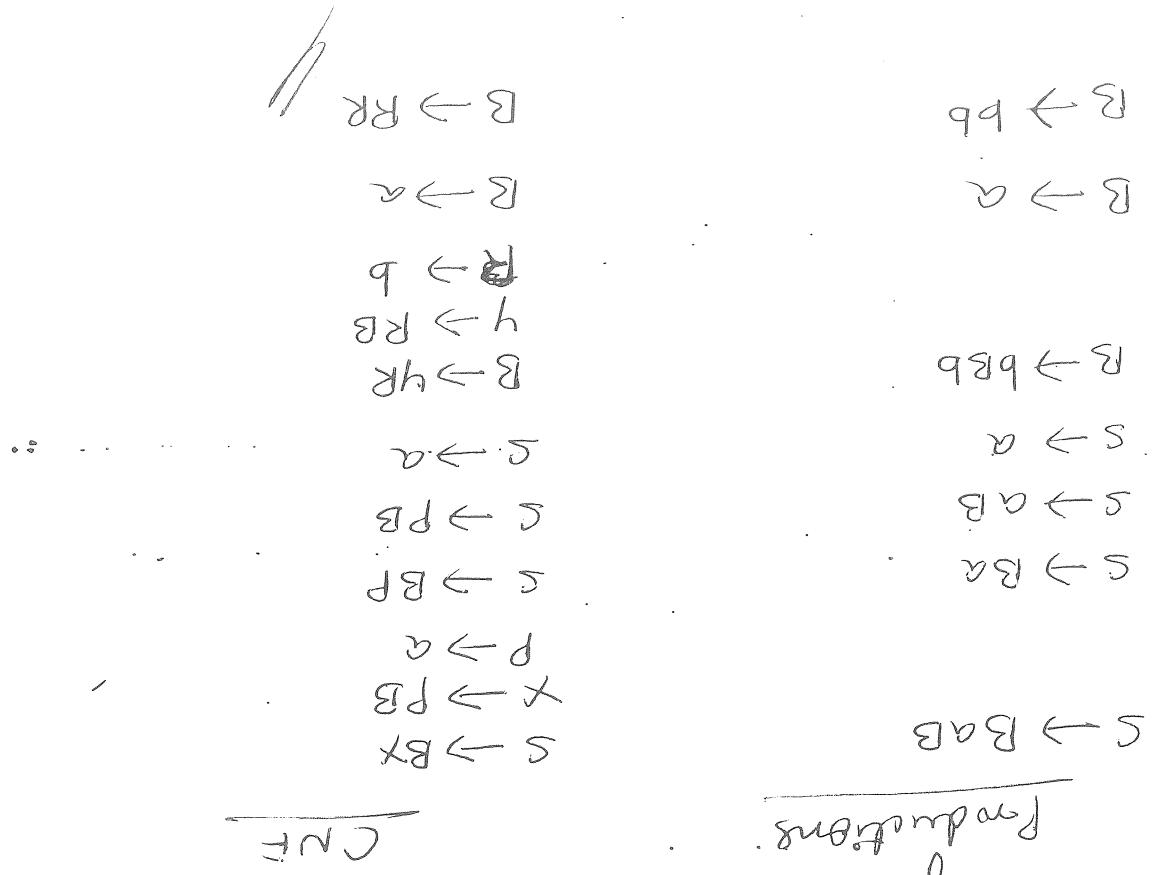
| | |
|-------------------|-----------------------------------|
| | $C \rightarrow CA AC$ |
| | $B \rightarrow bB a bBb$ |
| | $A \rightarrow Aaa$ |
| | $A \rightarrow BAC AC$ |
| | $A \rightarrow AA$ |
| | $S \rightarrow BAB BA AB a$ |
| | $S \rightarrow ABC AC$ |
| <u>Productive</u> | |

| | |
|-----------------------|-------------|
| | B, D |
| | D, B |
| | \emptyset |
| <u>Non Productive</u> | |

Ex:- Eliminating C - Productive

| | |
|--|--------------------------------|
| | $D \rightarrow E$ |
| | $C \rightarrow CA AC$ |
| | $B \rightarrow bBb a D$ |
| | $A \rightarrow Aa BAC Aaa$ |
| | $S \rightarrow ABC BAB$ |
| <u>Non Productive</u> | |
| From foll. grammar and put the resulting | |
| grammar rule C is | |

Ex:- Eliminate E, until productive & useless productive



Converting the above grammar into CNF:

$$B \rightarrow BBb | a | bb$$

$$S \rightarrow B.aB | Ba | aB | a$$

The resulting grammar is -

In the grammar, again if ~~as~~ not reachable from ~~as~~ go eliminate production A.

$$B \rightarrow BBb | a | bb$$

$$A \rightarrow AA | aA | aA$$

$$S \rightarrow B.aB | Ba | aB | a$$

The reduced grammar is :

The non terminal C generates the infinite length string. Hence C is closed instead of useless symbol. we will eliminate C and the productions in bold C appear.

Grammar Normal Form
 The rule for GNF is:
 In the rule $A \rightarrow A_1 A_2 \dots A_k$ if $i < j$ then the grammar is said to be left recursive. Remove the left recursion.
 When $i = j$, i.e. $A \rightarrow A_1 A_1 \dots A_k$ then the grammar is said to be right recursive. Remove the right recursion.
 If $i < j$ then process the rule to convert it.
 That must be $i < j$.
 $A \rightarrow A_1 A_2 \dots$
 4) In the rule $A \rightarrow A_1 A_2 \dots$ order i.e. A_1, A_2, A_3 and so on.
 3) Number the non-terminal symbols in ascending order.
 2) Bring the grammar to CNF.
 Procedure to convert CFG into GNF is:
 1) Convert the given grammar into simplified form by eliminating E-productions and useless symbols.
 2) Bring the grammar to CNF.
 $\{ \begin{array}{l} S \leftarrow Aa \\ S \leftarrow a \\ S \leftarrow AA \\ C \leftarrow AA \\ C \leftarrow a \\ C \leftarrow Aa \end{array} \} \in \text{in GNF}$
 $\boxed{NT \leftarrow t \cdot NT}$
 In short
 Non-terminal \rightarrow the terminal. Any number of
 The rule for GNF is:
 Grammar Normal Form
 In short, formula

3) Rename non-terminals in ascending order

$$\begin{array}{l}
 S \leftarrow b \\
 A_5 \leftarrow b \\
 A_4 \leftarrow a \\
 A_3 \leftarrow A_1 A_5 \\
 A_2 \leftarrow A_4 A_5 \\
 C \leftarrow xy \\
 Y \leftarrow AB \\
 Y \leftarrow A_1 A_2 A_3 \\
 A \leftarrow a \\
 B \leftarrow b \\
 S_B \leftarrow y \\
 S \leftarrow aa
 \end{array}$$

$C \leftarrow xy$ can be connected to $A_1 \leftarrow A_2 A_3$

3) Convert the grammar to CNF

$$\begin{array}{l}
 S \leftarrow AA \\
 S \leftarrow abSb \\
 C \leftarrow xy \\
 C \leftarrow A_2 B \\
 C \leftarrow A_1 A_2 A_3 \\
 A \leftarrow a \\
 B \leftarrow b \\
 S_B \leftarrow y
 \end{array}$$

2) There is no ϵ , as unit production. So there is no need to simplify the grammar.

$$NT \leftarrow t \cdot (set \cup NT)$$

Goal: - The CNF is

1) Convert the following grammar to CNF

$$\begin{array}{l}
 S \leftarrow aa \\
 S \leftarrow abSb
 \end{array}$$

Problem! -

3) Then convert each rule in the form

$$\text{Non-terminal} \rightarrow \text{The terminal. If y number of non-terminals}$$

Then convert above grammar in the following form

$$\begin{array}{l}
 A' \leftarrow \alpha | A \\
 A \leftarrow BA' | \beta
 \end{array}$$

$$5) A_5 \leftarrow b$$

$$4) A_4 \leftarrow a$$

$$3) A_3 \rightarrow A_5 A_3 A_5$$

$$2) A_2 \rightarrow A_5$$

$$1) A_1 \rightarrow A_2 A_3$$

The rules then become

The A_4 in RHS is replaced by terminal symbol a
Now consider rules for non terminals A_2 and A_3 .

$$5) A_5 \leftarrow b$$

$$4) A_4 \leftarrow a$$

$$3) A_3 \rightarrow A_4 A_5 A_3 A_5$$

$$2) A_2 \rightarrow A_4 A_5$$

$$1) A_1 \rightarrow A_2 A_3$$

To Summary

$$A_3 \rightarrow A_4 A_5 A_3 A_5$$

$$A_3 \rightarrow \boxed{A_2 A_3} A_5$$

$$A_3 \rightarrow \boxed{A} A_5$$

Therefore we will replace A_3 in RHS

$$A_5 \leftarrow b$$

$$A_4 \leftarrow a$$

$$A_2 \rightarrow A_4 A_5$$

$$A_1 \rightarrow A_2 A_3$$

$$\text{The rules are}$$

$$6) A_2 < A_0 ?$$

4) Now with newly the condition $\{ A_2, A_0 \}$ as

Now if we observe all the above rules, all the rules are in GNF except rule (i).
 Now if we observe all the above rules, all the rules are in GNF except rule (i).
 To summarize all, we get the following rules
 that are in GNF:
 1) Convert the grammar
 2) In CNF, So will numbers out the non terminals
 Step 2: The grammar is simplified and is also
 3) So
 as A_1, A_2, A_3, \dots
 in CNF.
 $S \rightarrow AB$ can be converted to $A_3 \rightarrow A_1 A_2 | b$
 $A \rightarrow BS | b$ can be converted to $A_3 \rightarrow A_1 A_2 | a$
 $B \rightarrow SA | a$ can be converted to $A_3 \rightarrow A_1 A_2 | a$
 $A_1 \rightarrow A_2 A_3 \dots, i \in J$
 Now will verify the condition $A_i < A_j$.

Now if we observe all the above rules, all the rules are in GNF except rule (i).
 To summarize all, we get the following rules
 that are in GNF:
 1) Convert the grammar
 2) In CNF, So will numbers out the non terminals
 Step 2: The grammar is simplified and is also
 3) So
 as A_1, A_2, A_3, \dots
 in CNF.
 $S \rightarrow AB$ can be converted to $A_3 \rightarrow A_1 A_2 | b$
 $A \rightarrow BS | b$ can be converted to $A_3 \rightarrow A_1 A_2 | a$
 $B \rightarrow SA | a$ can be converted to $A_3 \rightarrow A_1 A_2 | a$
 $A_1 \rightarrow A_2 A_3 \dots, i \in J$

Now A_2 is also in GNF

$$A_2 \leftarrow b A_3 A_2 A_1 | A_1 A_2 | b$$

Now consider rule for A_2

Now rule for A_3 is in GNF

$$A_1 \leftarrow A_1 A_3 A_2 A_1 | A_1 A_3 A_2$$

Hence

$$A_3 \leftarrow b A_3 A_2 A_1 | A_1 A_3 A_2 | b A_3 A_2$$

$$A \leftarrow B A_1 | B$$

left recursion

$$A_3 \leftarrow A_3 A_1 A_3 A_2 | b A_3 A_2 | a$$

The left recursion can be eliminated as

$$A_3 \leftarrow A_3 A_1 A_3 A_2 | b A_3 A_2 | a \quad \therefore A_2 \leftarrow A_3 A_1 b$$

$$A_3 \leftarrow A_3 A_2 | a \quad \therefore A_1 \leftarrow A_3 A_2$$

$$A_3 \leftarrow A_1 A_2 | a$$

Hence will process A_3 rule first

$$\text{But for } A_3 \leftarrow A_1 A_2 | a \quad i > j \text{ i.e. } 3 > 1$$

(3)

Assignment of GNF to CNF where $N = \{S, A\}$
 Converse of given Cfg
 $T = \{0, 1\}$ and $p \in$
 $S \leftarrow AA^0$
 $A \leftarrow SS^1$

Thus A' is now in GNF. The rules can be summarized as:

$$\begin{array}{l}
 A_1 \leftarrow bA^3 A^2 A^1 A^1 A^3 | aA^1 A^1 A^3 | bA^3 \\
 A_2 \leftarrow bA^3 A^2 A^1 A^1 A^3 | aA^1 A^1 A^3 | aA^1 A^1 A^3 \\
 A_3 \leftarrow bA^3 A^2 A^1 A^1 A^3 | aA^1 A^1 A^3 \\
 A^1 \leftarrow bA^3 A^2 A^1 A^1 A^3 | aA^1 A^1 A^3 | bA^3 \\
 A^2 \leftarrow bA^3 A^2 A^1 A^1 A^3 | aA^1 A^1 A^3 \\
 A^3 \leftarrow bA^3 A^2 A^1 A^1 A^3 | aA^1 A^1 A^3 \\
 A \leftarrow bA^3 A^2 A^1 A^1 A^3 | aA^1 A^1 A^3 | bA^3 A^2 A^1 A^1 A^3 \\
 A' \leftarrow bA^3 A^2 A^1 A^1 A^3 | aA^1 A^1 A^3 | aA^1 A^1 A^3 \\
 A'' \leftarrow bA^3 A^2 A^1 A^1 A^3 | bA^3 A^2 A^1 A^1 A^3 | bA^3 A^2 A^1 A^1 A^3
 \end{array}$$

6) Now the only remaining non-terminal in A

$$A \xrightarrow{\quad} A_1 A_3 A_2 A \mid \bar{A}_1 A_3 A_2$$

$$A \xrightarrow{\quad} b A_3 A_2 A \mid \cancel{b A_3 A_2 A} A_1 A_3 A_2$$

$$A \xrightarrow{\quad} b A_3 A_2 A \mid b A_3 A_2 A_1 A_3 A_2$$

$$A \xrightarrow{\quad} A_1 A_3 A_2 \mid b A_3 A_2$$

$$A \xrightarrow{\quad} A_1 A_3 A_2 \mid b A_3 A_2$$

(5) Now consider rule for A_1

$$A_1 \rightarrow A_3 A_3$$

$$A_1 \rightarrow b A_3 A_3 A_1 A_3 | a A_1 A_1 A_3 | b A_3$$

$A_1 \in \text{L}(G)$ in GNF.

L is set of final states
 Q is the start state
 S is a transition function denoted by
 Q * (Σ ∪ {ε}) * T * to Q + T *
 Σ is a stack alphabet
 Z is set of input alphabet
 Q is set of finite states
 where,
 $M = (Q, Z, T, S, Q_0, F)$
 The PDA is defined as a collection of six components
 what is PDA?

PDA \Rightarrow Finite State machine + Stack alphabet
 This PSM with stack is called Push Down Automaton
 Extra stack to generate accept the above languages
 languages. But we can convert PSM with
 finite state machines do not exist for the above
 pass through it
 4) $L = \{ w \mid w \in \{a, b\}^* \text{ and } w \text{ has balanced parentheses} \}$
 3) $L = \{ w \mid n_a(w) = n_b(w) \mid w \in \{a, b\}^* \}$
 2) $L = \{ w \cdot w' \mid w \in \{a, b\}^* \}$
 1) $L = \{ a^n b^n \mid n \geq 1 \}$
 not regular

Consider the following languages which are

Pushdown Automata

- a , z_0 is the crossed symbol
- a , z_0 is the symbol on the top of the stack during execution of the recently pushed symbol.
- Where,

$$a, z_0 / a$$

stack a and the acc is labelled with
then, there will be an acc from stack P to

$$S(p, a, z_0) = (q, \alpha)$$

If there is a transition of the form

stack symbol.

$$S(\text{state}, \text{input-symbol}, \text{stack-symbol}) = (\text{next-state},$$

In general,
the transition function accepts three parameters
namely a state, an input symbol and stack
symbol and returns a new state after
changing the top of the stack.

The action performed by the machine consists of:
1) changing the states from one state to another.
2) Replacing the symbol in the stack.

3) The symbol on top of the stack

4) The next input symbol

5) The current state

The transitions performed by the PDA depends on:

and a_0 is accept state
where a_0 is a start state.

$$g(a_1, \epsilon, z_0) = (a_2, \epsilon)$$

$$g(a_1, b, a) = (a_3, \epsilon)$$

$$g(a_0, b, a) = (a_1, \epsilon)$$

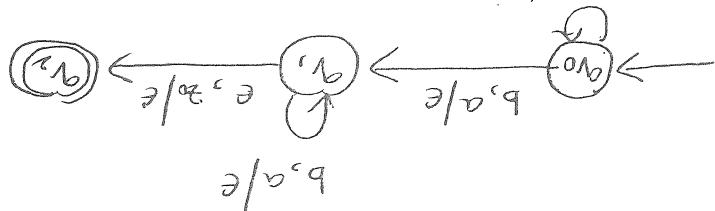
$$g(a_0, a, a) = (a_0, aa)$$

$$g(a_0, a, z_0) = (a_0, a z_0)$$

The instantaneous description can be given as:

$$(a_3, a | aa)$$

$$(a, z_0 | a z_0)$$



PDA:

stacking will be accepted:
all a 's and if we get stack empty then that
from the stack. If we read all b and remove
en reading every single b each a is popped
so:- First, will push all a 's onto the stack. Then
all a 's and if we get stack empty then that

descriotion of the PDA in the file stacking "aabbbb",
for PDA defined. Show the instantaneous
L = {aabⁿ | n ≥ 1}. Give the graphical representation.
2) construct a PDA that accepts the language
problems.

entire content of the stack.
the remaining stacking to be processed and the
An ID uses the current stack of the PDA,
what is instantaneous description?

In this language, a number of a 's should be followed by an number of b 's.

So if we read single 'a', we can push two 'a's into the stack. As soon as we read 'b', then for every single 'b', only one a should get popped from the stack. This basically maintains the $a \in b$ count and sequence.

Ques: Here, if we read single 'a', we can push string: aabbba

Show the moves made by PDA for the above transition diagram for PDA. Also

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

by final state:

Q) Design a PDA to accept the foll. language

$$\text{Accept state} + (q_2, \epsilon)$$

$$+ (q_1, \epsilon, z_0)$$

$$+ (q_1, a, a z_0)$$

$$+ (q_1, b, a z_0)$$

$$+ (q_0, bbb, a a z_0)$$

$$+ (q_0, abbb, aa z_0)$$

$$(q_0, aaaa, z_0) + (q_0, aaab, a z_0)$$

Simulate: we will simulate this PDA for the foll. string:

Accept

$$\begin{aligned}
 & + (a_0, e) \\
 & + (a_1, e, z_0) \\
 & + (a_1, b, a z_0) \\
 & + (a_1, b b, a a z_0) \\
 & + (a_1, b b b, a a a z_0) \\
 & + (a_0, b b b, a a a z_0) \\
 & + (a_0, a a a a z_0) \\
 & + (a_0, a a a a a z_0) \\
 & + (a_0, a a a a a a z_0)
 \end{aligned}$$

Gambleton for "aabb"

$$\begin{aligned}
 S(a_0, a a b b b b, z_0) &= S(a_0, a b b b b, a a z_0) \\
 S(a_0, a a b b b b, z_0) &= S(a_0, a a a a a z_0) \\
 S(a_0, a a a a a z_0) &= S(a_0, a a a a z_0) \\
 S(a_0, a a a a z_0) &= S(a_0, a a a z_0) \\
 S(a_0, a a a z_0) &= S(a_0, a a z_0) \\
 S(a_0, a a z_0) &= S(a_0, a z_0) \\
 S(a_0, a z_0) &= S(a_0, z_0) \\
 S(a_0, z_0) &= S(e, z_0 | e) \\
 S(a_1, e, z_0) &= S(a_1, e) \\
 S(a_1, e) &= S(a_1, e, z_0)
 \end{aligned}$$

ID:



The infinite sequence description is :

$$(a, z_0 | a a a)$$

$$(b, a | e)$$

The infinite sequence description is :

Sol: The meaning of well formed passenger is always (L, f left passenger come fast and then passenger's left passenger is gone fast and then passenger's right passenger is appear.

3) Design a PDA that accepts a regular expression of well formed Parenthesis. Consider the parenthesis's

4) Design a PDA for the language $L = \{www / weba, b\}$

Sol: This PDA is for generating even palindromes. This is a non-deterministic PDA as there is no way to find the position in between w and w' considered to be e.

Here the steps after between w and w' considered

$aa = aea$

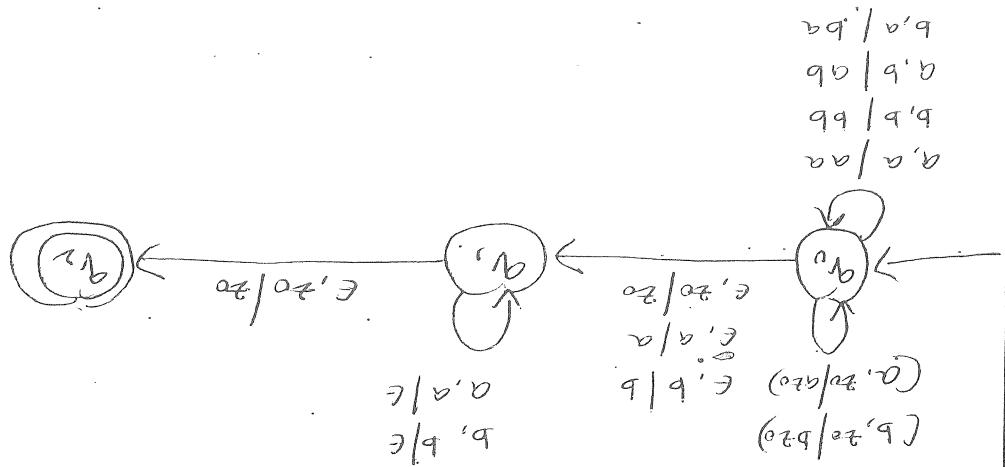
$abba = abeba$

$baab = baseba$

→ The string appearing before e is character will be pushed

→ The stack character by character. The each character after e is read and a single step of reading

→ Finally the stack must be empty and we must read by Jitter as popped from the stack.



$g(q_1, \epsilon, z_0) = (q_2, z_0)$ Accept

$$\left\{ \begin{array}{l}
 g(q_1, a, b) = (q_1, \epsilon) \\
 g(q_1, a, a) = (q_1, e) \\
 g(q_1, b, b) = (q_1, \epsilon) \\
 g(q_1, b, a) = (q_1, a) \\
 g(q_1, a, b) = (q_1, ab) \\
 g(q_1, a, bb) = (q_1, bba) \\
 g(q_1, b, aa) = (q_1, aab) \\
 g(q_1, b, ba) = (q_1, bab) \\
 g(q_1, a, ab) = (q_1, aza) \\
 g(q_1, a, z0) = (q_1, za) \\
 g(q_1, a, z0) = (q_1, z0a)
 \end{array} \right. \text{Push as } g, b, \text{ is } \text{onto the stack}$$

Ques:- The ID for this PDA will be :-

Algebra

[Real class note for more example]

$$+ (a_2, z_2) - \text{Accept}$$

$$+ (a_1, e, z_2)$$

$$+ (a_1, a, a_{20})$$

$$+ (a_1, ba, ba_{20})$$

$$+ (a_0, eba, ba_{20})$$

$$+ (a_0, ba, ba_{20})$$

$$8(a_0, abba, z_0) + 8(a_0, bba, a_{20})$$

Simulation for abba :-

RNS/T
Dept. of CSE,
Asst. Prof.,
RASHMI. M

Module - 4

Proof :- we know that every regular language is context free language.

Let us construct a context free language using regular language also. Now, we should show that context free since we can write a PDA to accept these exist atleast one context free language that regular language also. Now, we should show that by some DFA $M = (Q, \Sigma, S, q_0, F)$ from M , we construct a PDA $m = (Q, \Sigma, T, S, q_0, F)$ to accept L . In this case, M is simply a pumping lemma S_i is constructed as shown below:

where S_i is constructed as follows:

the stack as shown below. Let $m = (Q, \Sigma, \Delta, S, q_0, F)$

For every transition $S_i(q_i, a_j) = c$ introduce the transition $S_i = (q_i, a_j, c, \epsilon) = (q_i, c)$.

Theorem :- The context free languages properly contain the regular languages.

Where do CFLs fit? In this, we see the relationship between the regular languages and context free languages.

This is because, using stack it is not possible to count all these regulars and compare them. we know that, $L = \{w\#w \mid w \in \{a, b\}^*\}$ is context free language. But, language $L = \{ww \mid w \in \{a, b\}^*\}$ is not context free ..

Properties of CFLs

Now, M' behave identically to M and hence $L(M) = L(M')$. So, the regular languages are a subset of the CFL.

\Rightarrow The regular languages are a proper subset of the CFL because, there exists at least one language which is context free but not CFL because, it may hold in context free but not context free but not regular.

Pumping lemma for CFLs:

State and prove pumping lemma for context free languages.

$L = \{a^n b^n c^n\}$ may hold in context free but not regular.

\Rightarrow The regular languages are a proper subset of the CFL because, there exists at least one language which is context free but not CFL because, it may hold in context free but not context free but not regular.

Such that $|wxy| \leq n$, $|xy| \leq 1$, then $wxy \in L$

$$x = wxy$$

Combination of strings

single. If the string z can be decomposed into xz that $|z| \geq n$ whence some positive

length z as. Subsequently long string and $z \in L$

Statement: Let L . If the CFL and infinite.

Such that $|wxy| \leq n$, $|xy| \leq 1$, then $wxy \in L$

such that $|wxy| \leq n$, $|xy| \leq 1$, then $wxy \in L$

Same non-terminal occurs twice on the same path

are generated by the parser tree where the

is the length of the longest string that can

$$p \in Q, 1, 2, \dots \text{ where}$$

\hookrightarrow The string z is sufficiently long so that it can

through the tree

and y in that sequence

to decomposed into various substrings w, v, u, x

\hookrightarrow The two substrings v and x are present somewhere

in z

two cases, we have all the facts for pumping lemma
formulas and the derivation steps. If we can prove this.
Instructors some numbers of times, we get a string of
Case 3. Σ EL implies that after applying some rule

for some x and ϵ) and should it be applied more than

$$A \Leftarrow x A B$$

some non-terminal A such that
uniform strings can be generated if the grammar has
variables (non terminals) must be accurate (Not the
assumed that the string is uniform), or a more
Case 1: To generate a uniformly long string Σ (if it is

to the following two cases

applying rules of production. Proof of this theorem uses
know that Σ is strings of terminals which is derived by
string Σ is finite and is result for language. We
Proof: According to pumping lemma, it is assumed that

of course, the string Σ is not context free.
difficult to be in L and the string Σ is context free.
number of times, the easiest string will be
and if we duplicate substring v and x same
 \Rightarrow If all the points mentioned above are satisfied,
Since $|v| \geq 1$, one of them can be empty.

\Rightarrow Both the substrings v and x cannot be empty
 \leftarrow we do not know since $|vwx| \leq n$ for some positive integers n

\leftarrow The string w in between v and x cannot
appear after x .

\leftarrow The substring w appears before v , the substring
to Σ is between v and x and the substring

Proof of case 1: To prove a sufficiently long string
Z (at 1) is assumed that the string is sufficient,
one set more variables (non terminals) must be
available and each production has finite length. The
grammar and the grammar has a finite number of
variables. Let us assume that the language is
countable and the grammar is countable. Then
only way to derive sufficiently long strings using
such productions is that the grammar should have
only a finite number of variables. Assume that no
such production is available. Then we can
say no variable is recursive, such variable must be
diffined only in term of terminal / other variables.
Since there variables are also non - recursive, they
have to be defined in term of terminals. and the
variables said so on if we keep applying the productions
like this, then all no variable at all in the final
derivation and finally we get a string of terminals
and the generated string is finite. From this, we
can conclude that there is a limit on the length of the
string that is generated from the start symbols.
This contradicts the assumption that there are more
variables than non - recursive and it measures
finite. Thus first, the assumption that there are more
variables than non - recursive is incorrect. It means
that one more variable Vocabularies are recursive and
thus the first form

Proof of case 2: Z EL infinite that after
achieves same / all productions some number of times
we go finally. Using of terminals and the derivation
steps. Let Z EL is sufficiently long strings and so
the duration must have involved recursive use of
steps. Proof of case 2: Z EL infinite that after

some more - terminal

that UV_{Ay} occurs in the derivation, and

$$S \xrightarrow{*} UAY \xleftarrow{*} UVAY$$

Note from the derivation

$$|VX| \geq 1$$

Used column clearly shows that

$$A \xleftarrow{*} VAX$$

or introduces a terminal. The derivation

$$\text{length of the structural form (using successor variable)}$$

shows that every derivation step either introduces the

derives next certain E-structure or with precedences. At

can be easily derived since CFG that generates CFL

succession shows it is assumed that $|VWX| \leq n$. This

implies that longest string VWX is generated without

must also be possible. Next, we have to prove

$$A \xleftarrow{*} VWX$$

that the derivation

is also possible, from this we can easily conclude

$$A \xleftarrow{*} W$$

and

$$A \xleftarrow{*} VAX$$

it implies that the following derivations

$$S \xrightarrow{*} UAY \xleftarrow{*} UVAY \xleftarrow{*} Z$$

and the final derivation should be of the form

$$S \xrightarrow{*} UAY \xleftarrow{*} UVAY$$

last term the following form:

Note that any derivation should start from the start symbol S. Since A is used successively, the derivation

$$S \xrightarrow{*} UAY$$

A and the derivations must have the form

Note: Pumping Theorem for CFL
 ← The pumping lemma for a regular set satisfies
 that every sufficiently long string in a regular set contains a short string that can be pumped. In other words, if a long string is pumped and if we push up any number of steps, then we get back the original string. But this can be repeated as many times as required.

Context free using pumping lemma.
 The language can be proved to be non-regular if the closure properties are followed.
 If a push down automata can be designed for the given language.
 If it belongs to context free grammar.
 If the closure properties are followed as CFL.

Note: Showing a language is context-free
 The given language can be represented as
 $uvwy \in L$
 and hence the form

also possible, it follows that

$$A \xrightarrow{*} w$$

$$A \xrightarrow{*} vAx$$

and

so that $uv^ixy \in L$ for $i = 0, 1, 2, \dots$
such that $|uvx| \leq n$ and $|vxy| \geq 1$
 \Rightarrow if the string $z = uvwxy = a$
 \Rightarrow since $|z| \geq n$, $a \in z$ to Pumping lemma we can

Let z be any string $\in L$

so, we can apply Pumping-lemma.

Assume that L is context-free and $z \in \text{infinite}$
Goal: Shows that $L = \{a^p b^q c^r\}$ is not context-free.

Example:

language L is not context-free.
the language is context-free. Through, the given
result is a contradiction to the assumption that
according to pumping lemma, $uvwxy \in L$. So, the
find any z such that $uv^ixy \notin L$.

$z = uvwxy$ where $|uvx| \leq n$ and $|vxy| \geq 1$
but strings $u, v, w, x,$ and y such that
select the string $z/v/w$ and break at v/w

of z is context-free.

Assume that the language L is infinite &
below:

given language is not context-free is shown

The general strategy used to prove that a
language is not context-free is to prove that it
cannot be used to prove that certain languages
free languages. Note that Pumping lemma
proves that certain languages are not context-
free context-free.

Applications of Pumping lemma for CFLs:

[Refer class notes for more examples]

Context free

\therefore So, the language $L = \{a^p \mid p = q^2 \text{ for some integer } q\} \cup \{a^5 \mid 5 \leq 5 \text{ for any numbers}\}$

$$w = a a a a a = a^5 \text{ dL}$$

$$w = \overbrace{a a a a}^{\text{here}} \times \overbrace{y}^{\text{is absent}}$$

$$w = u v w x y = u v w x y$$

$$\text{for } q = 2$$

$$w = u v w x y$$

$$\therefore w = a^2 = a^4 = a a a a$$

$$\text{Let } q = 2$$

$$\text{Let } w = a^p \mid p = q^2$$

\Rightarrow This proves that the given language L is not a context free.

$$w = a^2 b^3 c^3 \notin L \quad [\text{Assumption: } w \in L]$$

$$w = a a b b b c c c$$

$$w = a a b b b c c c$$

$$w = u v v w x x y$$

$$w = u v i w x y = u v i w x y$$

Case 2: Consider $i = 0$ then,

writing

Hence our assumption of L being DFA is

$$\text{i.e., } w' = a a b c \notin L$$

$$w' = u v w y$$

If $i = 0$, then v^0 and x^0 . That means v and x are absent then,

$$w = a a b b c c c$$

$$w = u v i w x y$$

Case 1: Consider $i = 0$ then,

Consider Vague Case

i.e. we have additional recurrence of v and x .

$$w = u v i w x y$$

Now let us consider

$$\text{Let } |v x| \geq 1 \text{ and } |w v x| \leq n$$

$$w = u v w x y$$

Let

Let, w be any string such that $w \in L$

$L = a^n b^n c^n$ is a context free language

So let us assume that

a context free language

f) Show that $L = \{a^n b^n c^n \mid n \geq 0\}$ is not

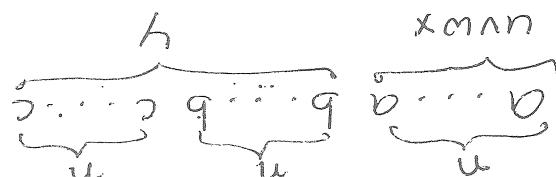
as followed by equal numbers of a 's and b 's and c 's
 Note that the following should have same number of
 But w.r.t pumping lemma, $uv^kwx^y \in L$, which is if the
 language is not context free.

Note that $uv^kwx^y = a^{n+j+k}b^m \notin L$ when



A/c to pumping lemma, $uv^kwx^y \in L$ for $j > 0$ and
 the language generated is as shown below:

Let $v = a^j$, $x = a^k$ where $|v| = j+k \geq 1$ and



and so $uv^kwx^y \in L$ for $j = 0, 1, 2, \dots$

$|vwx| \leq n$ and $|vx| \geq 1$.

$uvwx$ should thus

Note that $z \leq n$ and so we can split z into

$z = a^pb^m \in L$

So, If L is context free and it is infinite.

∴ S.T $L = \{a^nb^m \mid n \geq 0\}$ is not context free.

Notes on pp. 5-6 consider the language as generated by and assume that it is

the $G_1 = (v_1, T_1, P_1, S_1)$ and $G_2 = (v_2, T_2, P_2, S_2)$ keep ϵ -distance
and assume that v_1 and v_2 are disjoint.

Theorem 3: - Show that if L_1 and L_2 are context free languages then $L_1 \cap L_2$ is also context free.

i.e., the CFLs are closed under intersection.

Proof: - Let L_1 and L_2 are the two CFLs generated by

1. The CFLs are closed under union.
 2. The CFLs are closed under concatenation.
 3. The CFLs are closed under Kleene closure.
 4. The CFLs are not closed under intersection.
 5. The CFLs are not closed under complement.

The Context free languages are closed under some closure measures of its performing that part of a speech act in those CFLs the sequential language is context free language. These properties are as below:

Impact of Closeness Properties of CFLs

$$\therefore L_3 = L_1 \cup L_2$$

Why it is proved that CFS are closed under union.

The above derivation uses only the products in \mathcal{M} .
 $S_2 \Leftarrow w$
 $S_1 \Leftarrow w$

In the second case, all the variables in V_2 and all the terminals in T_2 may be used to get the derivation.

The above derivation uses only the products in \mathcal{M} .

$$S_1 \Leftarrow w$$

derivation

And all the terminals in T_1 may be used to get the as possible. In the first case, all the variables in V_1

$$S_3 \Leftarrow S_2 (q)$$

$$S_1 \Leftarrow S_2$$

So, if we assume $w \in L_3$, one of the derivations is possible.

$$S_3 \Leftarrow S_2 \Leftarrow w$$

derivation from S_3 is

if we assume $w \in L_2$, then the possible

$$S_3 \Leftarrow S_1 \Leftarrow w$$

derivation from S_3 is

If we assume $w \in L_1$, then the possible

$L_3 = \{V, UVU^{-1}U^2V^2, TUU^{-1}, PUPU^{-1}S^2, S^2\}$

L_3 is the language generated from $L(G_3)$

$$G_3 = (V_3, T_3, P_3, S_3)$$

$$\therefore G_3 = (V_1, T_1, P_1, S_1)$$

L_1 is the language generated from $L(G_1)$

$$\begin{aligned} S_2 &\leftarrow abS_2ba \\ S_1 &\leftarrow aS_1bS_1ab \\ S_3 &\leftarrow S_1S_2 \end{aligned}$$

To generate L_3 , the products are:

$$S_2 \leftarrow abS_2ba$$

To generate L_2 , the products are:

$$S_1 \leftarrow aS_1bS_1ab$$

So, To generate L_1 , the products are:

Show that $L_3 = L_1L_2$ is a CFL

If $L_1 = \{a^n b^n : n \geq 1\}$ and $L_2 = \{(ab)^m : m \geq 1\}$

Example :-

Theorem 9:

Show that if L_1 and L_2 are two CLEs, then

• 730 05/0 88 87-17

The Context free languages are closed under

Concave notation.

Proof: Let L_1 and L_2 be two CFLs generated

From the City

from the City

$$G_1 = (v_1, T_1, P_1, S_1) \text{ and } G_2 = (v_2, T_2, P_2, S_2)$$

young ~~seas~~ off by

Now, if we consider the language to be generated

$G_4 = (u, u_1, u_2, u_3, T_1 \cup T_2, P_4, S_4)$

$$\text{source} \rightarrow p_4 = p_1 \cup p_2 \cup \{s_4\} \leftarrow s_1 s_2 \}$$

and so (v_1, v_2)

44 → is the best symbol for the gamma 45 ←

by
all
and
free
central
gas
station
as
the

• If it is clear from this that the specimen is
centrifuge generated by the gamma as also centrifuge.

under consolidation.

Thus, it is proved that CFLs are closed

If we take L_1 as grammar $G_1 = (V_1, U_1, S_1, P_1)$ and L_2 as grammar $G_2 = (V_2, U_2, S_2, P_2)$ then thus that the grammar G_5 is generated by the context free and the language generated by this grammar is also context free.

$\Rightarrow G_5 = P_1 \cup \{S_1 S_2\} \rightarrow S_5 \leftarrow S_1 S_2$
Where, S_5 is the start symbol for the grammar G_5

Let us consider the language L_5 generated by the grammar $G_5 = (V_1 \cup V_2, U_1 \cup U_2, S_1 S_2, P_1 \cup P_2)$

Proof:- Let L_1 be the CFL generated from the grammar $G_1 = (V_1, U_1, P_1, S_1)$ and

$\therefore L_1 = L_1 \cup L_2 \cup \{S_1 S_2\}$ where $L_1 = L_1 \cup L_2$ and $L_2 = L_2 \cup L_1$

$\therefore G_4 = (V_1 \cup V_2, U_1 \cup U_2, S_1 S_2, P_1 \cup P_2 \cup \{S_1 S_2\})$

$$S_4 \rightarrow ab \mid S_2$$

$$S_1 \rightarrow aS_1b \mid ab$$

$$S_2 \rightarrow S_1 \mid S_2$$

To generate L_4 , the products are:

$$S_2 \rightarrow ab \mid S_2$$

To generate L_2 , the products are:

$$S_1 \rightarrow aS_1b \mid ab$$

CFL To generate L_1 , the products are:

$$S_1 T \quad L_1 = L_1 \cdot L_2 \text{ is a CFL.}$$

$\therefore L_1 = \{ab^m \mid m \geq 1\}$ and $L_2 = \{(ab)^m \mid m \geq 1\}$

Example :-

Proof: Let us prove this theorem by taking free language.

If L_1 is not always true that $L_1 \cup L_2 \Rightarrow$ context free language,

Statement: If L_1 and L_2 are context free languages,

CFLs are met closed under intersection.

Theorem: $L = \{a^n b^n | n \geq 0\}$

Countless examples. Consider the two languages

$L_1 = \{a^n b^n | n \geq 0, m \geq 0\}$ and

$L_2 = \{a^n b^n | n \geq 0, m \geq 0\}$

The two languages are context free, as we can easily obtain the corresponding CFLs

$S \rightarrow S_1 S_2$

$S_1 \rightarrow a S_1 b$

$S_2 \rightarrow S_2 S_1$

$S_2 \rightarrow S_2 S_1$

$S_2 \rightarrow S_2 S_1$

$S_2 \rightarrow S_2 S_1$

and

$S \rightarrow S_2 S_1$

$S_1 \rightarrow a S_1 b$

$S_2 \rightarrow S_2 S_1$

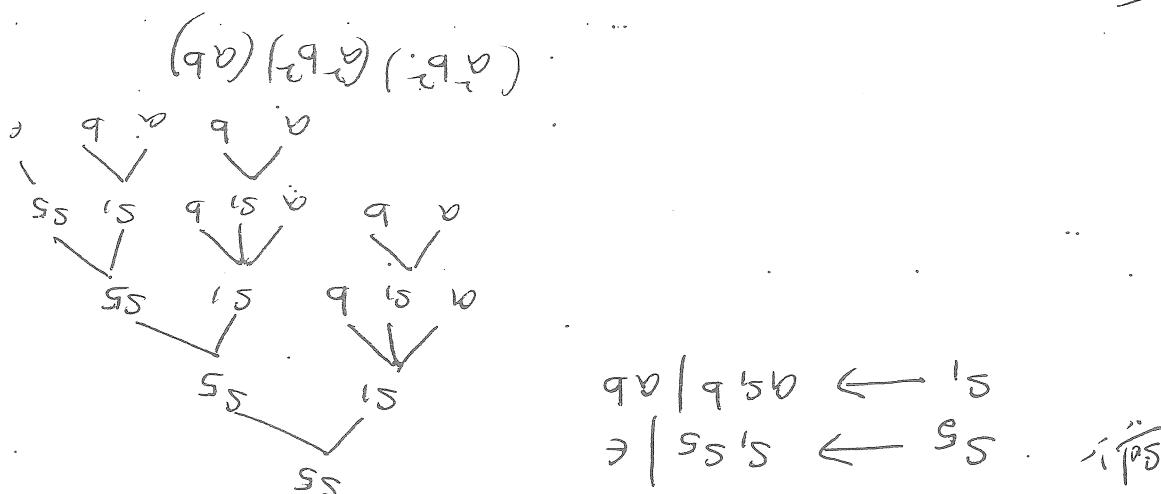
can easily obtain the corresponding CFLs

$L_1 = \{a^n b^n | n \geq 0, m \geq 0\}$

$L_2 = \{a^n b^n | n \geq 0, m \geq 0\}$

and

It is not always true that $L_1 \cup L_2 \Rightarrow$ context free language.



Example: If $L = \{a^n b^n | n \geq 0\}$. Show that $L = L_1 \cup L_2$ is also a CFL.

Now, let us take $L_1 = \{a^n b^n c^n | n \geq 0\}$ and $L_2 = \{a^m b^m | m \geq 0\}$.
 we have already proved earlier that this language is not context free. Thus we can prove that the family of CFLs is not closed under intersection.
 $L_1 \cap L_2 = \{a^n b^n c^n | n \geq 0\}$ and $L_3 = \{a^n b^n | n \geq 0\}$ and $L_4 = \{a^n b^n c^n | n \geq 0\}$
 we have already proved earlier that this language is not context free. Thus we can prove that the family of CFLs is not closed under intersection.
Theorem 5: If L is context free language, then L' is context free if and only if $L' \cap L$ is context free.

Proof:- Let us prove this theorem by contradiction.
 Suppose that CFLs are closed under complementation. So, if L and L' are CFLs, then $L' \cap L$ must also be closed under union. So, $L \cup L'$ must also be closed under union. Since we have assumed that the context free. Since we have assumed that the context free. Since we have already proved that CFLs are also context free.

So, if L and L' are CFLs, then L and L' are CFLs and $L \cup L'$ are CFLs. But we prove that this theorem is not true that CFLs are closed under complementation.

A.e. An $a^n b^n c^n$ language is followed. But,
 $L_1 \cap L_2 = \{a^n b^n c^n | n \geq 0\}$ and $n \geq 4$
 $L_2 = \{a^m b^m | m \geq 0\}$ and $m \geq 20$
 $L_1 = \{a^4 b^4 c^4\} = \{aaaa bbbb cc\}$
 $L_1 \cap L_2 = \{aaaa bbbb cc\}$ and $n \geq 4$

Now, let us take $L_1 = \{a^n b^n | n \geq 0\}$ and $L_2 = \{a^n b^n c^n | n \geq 0\}$.
 we have already proved earlier that this language is not context free. Thus we can prove that the family of CFLs is not closed under intersection.

$$L_1 \cap L_2 = \{a^n b^n c^n | n \geq 0\}$$

Now, let us take

PDA's are more powerful than the finite automaton.
 The above statement automatically implies that
 languages

The regular languages are the subset of CFL
 and so CFL are more powerful than the regular
 languages. So we can make a of these stages are:
 languages. Even then some of the important point we
 cannot have the corresponding PDA for these types of
 context using context free grammars and so we
 are not context free and it is not possible to
 5. $L = \{a^p b^q | p = q\}$
 4. $L = \{a^n | n \geq 0\}$
 3. $L = \{ww | w \in \{a, b\}^*\}$

$\therefore L = \{w | w \in \{a, b, c\}^*\text{ where } n_a(w) = n_b(w)\}$
 2. $L = \{a^n b^n | n \geq 0\}$

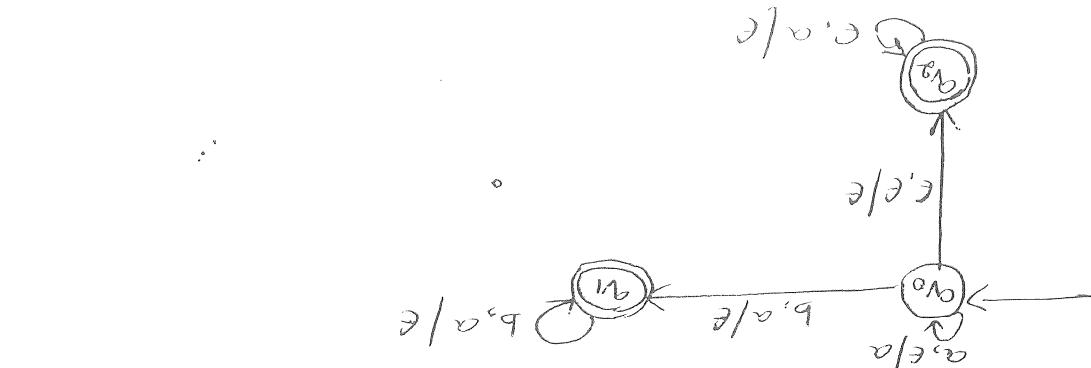
Note: we have seen that the following languages

So, the family of CFLs are not closed under
 concatenation, union, intersection & not even
 under inclusion, our assumption that the CFLs
 are closed under complementation is not true.
 So, $L_1 \cup L_2$ must be context free which is a
 contradiction. Since the CFLs are not closed

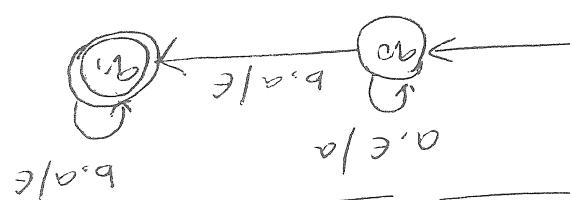
$$\underline{L_1 \cup L_2} = L_1 \cup \underline{L_2}$$

according to De-Morgan's law we have:

$$\underline{L_1 \cup L_2} \text{ must be context free. But,}$$



we delete all a 's followed onto the stack as shown:
 transition from state q_0 to final state q_3 is deleted. In such situation, we can have one more followed by b 's, then also the string has to be followed by a 's and not b 's. If we input only a 's and not b 's, in state q_0 , if we push a 's and not b 's, then also the string has to be accepted.



Non-deterministic PDA:

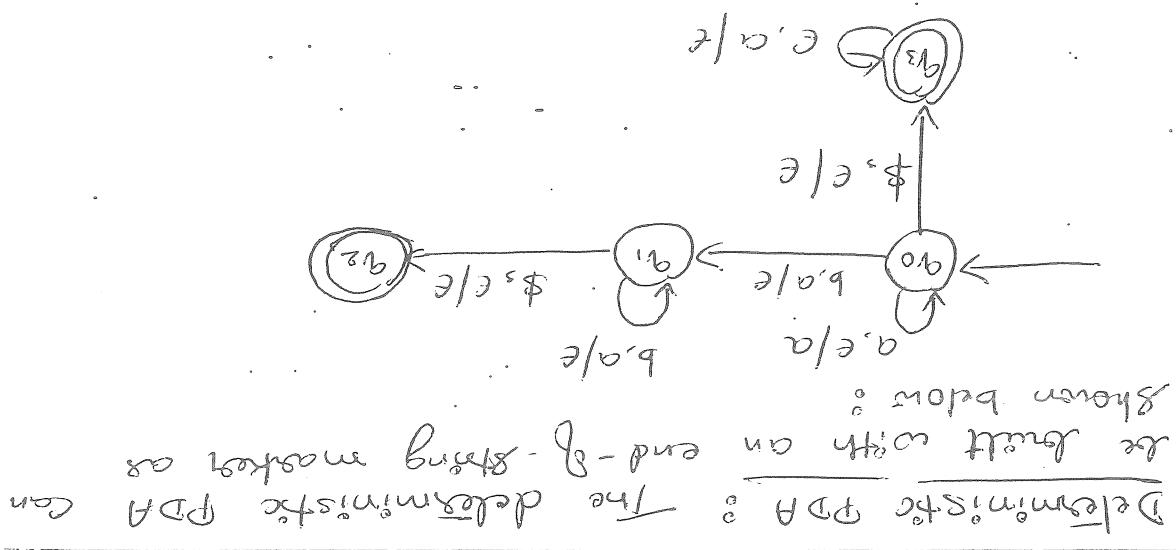
$L = a^* \cup \{a^n b^n \mid n > 0\}$ to accept the language: Between a deterministic and non-deterministic PDA

if L can be accepted by some PDA where \neq is end-of-string marker. Instead of we can use some feed old configuration return.

Linea~~r~~ Bound~~d~~ Alternation is most powerful Turing machine.

Some of the languages as pointed out earlier that are not context free and so we need much more powerful automaton than the PDA such as Linear Bounded Alternation or most powerful

\Rightarrow But, the PDAs are not strong enough to accept



The Turing machine provides an ideal form. the model of a computer. Turing machine are useful in several ways: if it is used for determining the undecidability of certain languages. As an automaton, the turing machine is the most general model, if accepted type-0 languages can also be used for computing functions. It turns out to be a model of computation. It is part of all recursive functions. measuring the space and time complexity of problems.

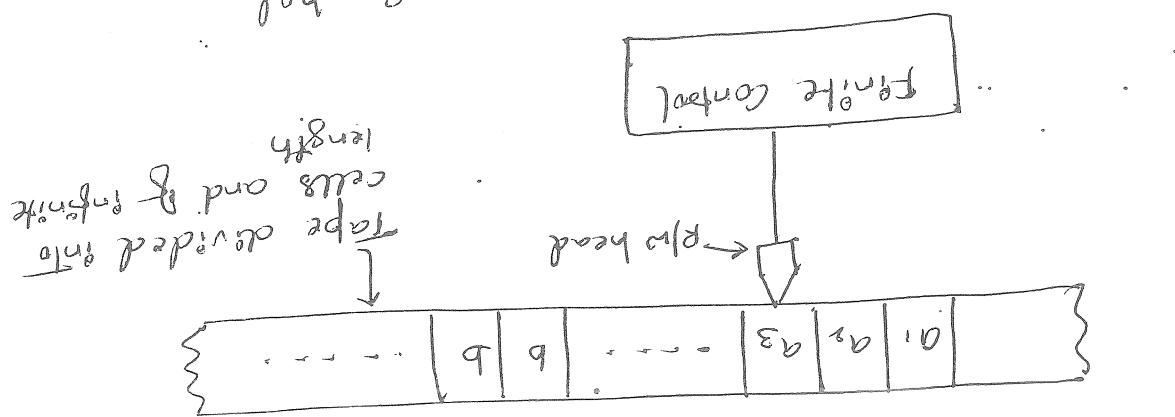
Turing machine moves in several ways:

- (i) writing a new symbol in the cell being
- (ii) moving to the cell left of the present cell
- (iii) moving to the cell right of the present cell

the three simple operations, namely cells one at a time and usually performs one of three cell. In turing machine one scans the paper, which can be viewed as a tape divided person writes symbols in a one-dimensional turing assumed that while computing, a

Turing Machine

In the move, the machine examine examples the present symbol under the R/W head on the tape and the present state of an automaton to determine which can execute the cell at a time. ←
 Each cell can step only one symbol
 and the output from the ~~function~~ R/W head →
 In the move, the machine examine examples the present symbol under the R/W head on the tape and the present state of an automaton to determine which can execute the cell at a time.
In the move, the machine examine examples the present state of an automaton to determine which can execute the cell at a time.



Turing machine model:

Definition:

Translating machine M is a 7-tuple, namely

(Q, Z, T, S, q_0, b, F) , where

1. Q is a finite non-empty set of states

2. T is a finite non-empty set of tape symbols

3. $b \in T$ is the blank character

and is a subset of T and $b \notin Z$.

4. Z is a non empty set of input symbols

5. S is the transition function mapping

$(q, x) \rightarrow (q', y, D)$ where D denotes the

direction of movement of the head.

$D = L \text{ or } R$ (accordingly as the movement is to the left or right)

6. $q_0 \in Q$ is the initial state

7. $F \subseteq Q$ is the set of final states

8) S may not be defined for some elements of the eachable set from the initial state to some by a steping is decided by

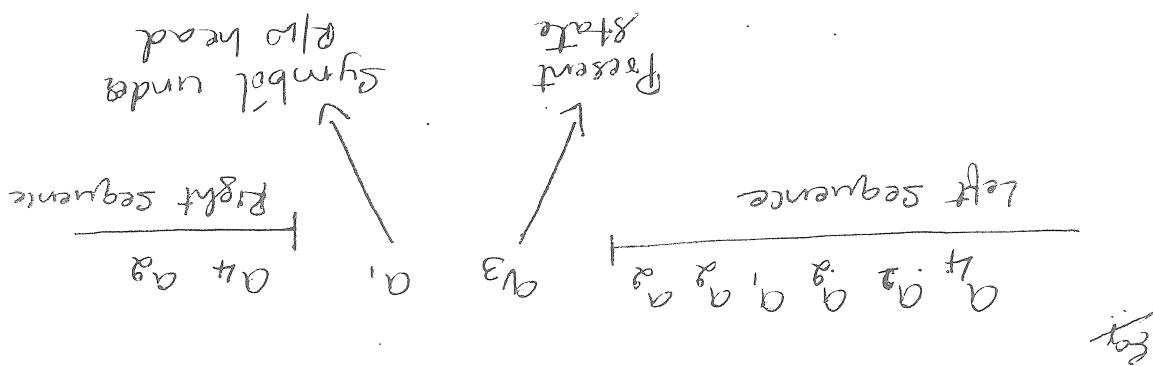
Note:

* If $transition$ is not defined for any input in a particular state, then it means that the

a particular state leads to the trap state.

Final state \rightarrow $A \times T$

- as part of the left of a right substring.
- 2) we observe that the blank symbol may occur symbol under the R/W head.
 - 3) For constructing the ID, we simply insert the cassette state in the input string to the left of the symbol under the R/W head.
- Note:



Repetition: In ID of a turing machine M is a string a^k , where k is the present state of M .
 Representation by Inglantamico descriptions: The entire input string is split as a^k , the first symbol of a^k is the current symbol a under the R/W head and it has all the subsequent symbols under the R/W head and it has all the subsequent symbols to the left of a .

! Transitions: instantaneous descriptions using move-relations
 We can describe a Turing machine employing
Representation of Turing Machine:

(iii) Transition diagram

(ii) Transition table

• *apple cake*

- The machine changes the state from a_0 to a_1 . The edge is labeled with (a, x, b) .
- Where a is the current input symbol read from a tape word has to be replaced by b in the tape moving the read/write head to soft / right.
- The transition from one state to another state is indicated by a dashed edge.
- The start state is a state which has an incoming arrow not originating from any node and ending in the state. This is labelled with "i".
- The final states of accepting states which can be reached by a double edges. The states are separated by a dot.
- The whole machine is self-reduced by a self-loop at the final state.



Representation by transfiguration diagram

↳ gives the movement of the head (eye)

$\alpha \leftarrow$ If $i < 8$ swap them in the current cell

Where γ \leftarrow denotes the move state into which the following machine enters the current cell

$$S(\alpha, \beta) = (\alpha, \beta)$$

fallie called the translation table.

we give the definition & see the form of a

Representation by transmission route

Language acceptance by a Tuning machine:
The tuning machine can do one of the following things:
 \hookrightarrow Hart and accept by entering into final state
 \rightarrow Hart and reject. This is possible if the transition is not defined i.e. $S(a,a)$ is not defined.

Standard Turing Machine :- Since these can be succinct statements of Tm , the Tm that we are studying now can be called as standard Turing machine with the following features :

→ The Turing machine has a tape that is divided into numbers of cells with each cell capable of holding only one symbol. The shape is unbounded (i.e., no boundary in the left as well as in the right) with any numbers of left as right moves.

→ The Turing machine as a tape that is made up of determinate. It can have either zero or one description for each configuration → Some of the symbols in the tape can be considered as the blank symbol (Some symbols of all the symbols in the tape) . The symbols are considered as the input .. The symbols can be considered as the output of the configuration. Some of the symbols in the tape can be considered as the control symbols.

Can be considered as the configuration of the automaton in the halts.

What is a recursively enumerable enumerable language?

A language L is recursively enumerable if there is a Turing machine which is accepted by a TM, the machine has input and output No.

Output Yes if it belongs to the language. If it is accepted by a TM i.e., given a string which is input to TM, the machine halts and

say that the string is accepted by TM.

After some sequence of moves, if the TM enters into final state and halts, then we say that the string is accepted by TM.

In short, the machine will be in the start bank.

The string to which it is the string to be scanned should end with infinite numbers of symbols to form left bracket to the head pointing to the first symbol.

Recursively enumerable language is RE language.

← The language accepted by TM is called

Note :-

where now is the initial ID and accepted as the final ID. The set of all these words to be formed causes no move from start state to the final state p.

$$L(w) = \{w \mid w_0 + w_1 \text{ is where}$$

defined as -

TM. The language $L(w)$ accepted by M is defined as -

What is the language accepted by a Turing machine?

The numbers of state in TM must be
minimized. It is achieved by changing the
state only when take symbol is changed
as there is a change in the movement
of the head.

After scanning the symbol below the P10
head is to know what activity has to be
done in future. The machine must
remember all the scanned symbols by moving
symbols to be scanned. The machine can
remember all the scanned symbols and the
done in future. The machine must
to next state.

What are the basic guidelines for designing
a Turing machine?

After scanning the symbol below the P10
head is to know what activity has to be
done in future. The machine must
remember all the scanned symbols by moving
symbols to be scanned. The machine can
remember all the scanned symbols and the
done in future. The machine must
to next state.

The TM that always halt irrespective of
whether they are not a good model for
an algorithm. If an algorithm exists to solve a
given problem, then the problem is decidable otherwise
it is undecidable problem.

The languages which TM always halt output "no" if it
belongs to the language of output "yes". If it
does not belongs to the language are called
decidable languages or recursive languages.

$a_1 a_3 a_5 a_7 a_9 \dots$ | $a_1 a_3 a_5 a_7 a_9 \dots$

Move i_8

Sol: The next ID is $a_1 a_3 a_5 a_7 a_9 \dots$

(a_3, b_1, L) is the transition, what is the next ID?
 $a_1 a_3 a_5 a_7 a_9 \dots$ and $S(a_3, a_5) =$

Q) Given the following ID is a TM:

$a_1 a_3 a_5 a_7 a_9 \dots$ | $a_1 a_3 a_5 a_7 a_9 \dots$

This can be represented by a move as shown below:

$a_1 a_3 a_5 a_7 a_9 \dots$

is obtained:

\leftarrow R indicates that the R/W head is moved one symbol towards right and the following ID

by b_1

\leftarrow The current input symbol a_5 is replaced

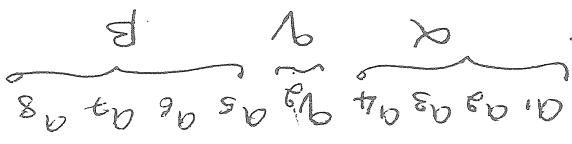
\leftarrow The machine enters into state a_3

\leftarrow applied the following transitions are performed:
 when the transition $S(a_3, a_5) = (a_3, b_1, R)$

\leftarrow The next symbol to be scanned is a_5

\leftarrow The machine is in state a_3

The above ID indicates that:



Sol: The ID: $a_1 a_3 a_5 a_7 a_9 \dots$

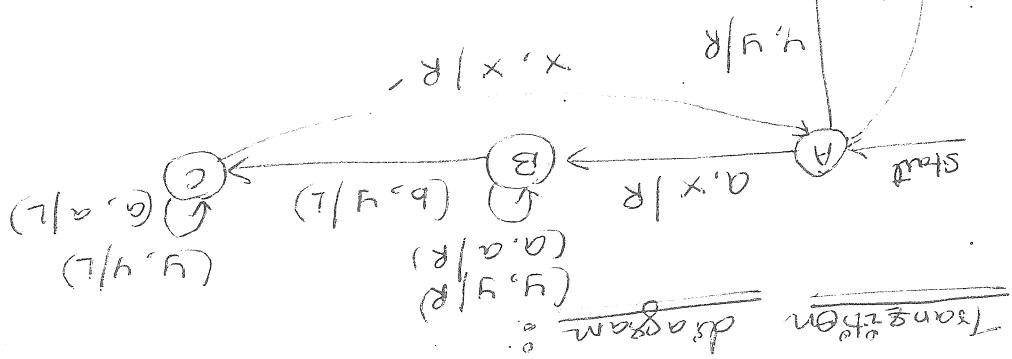
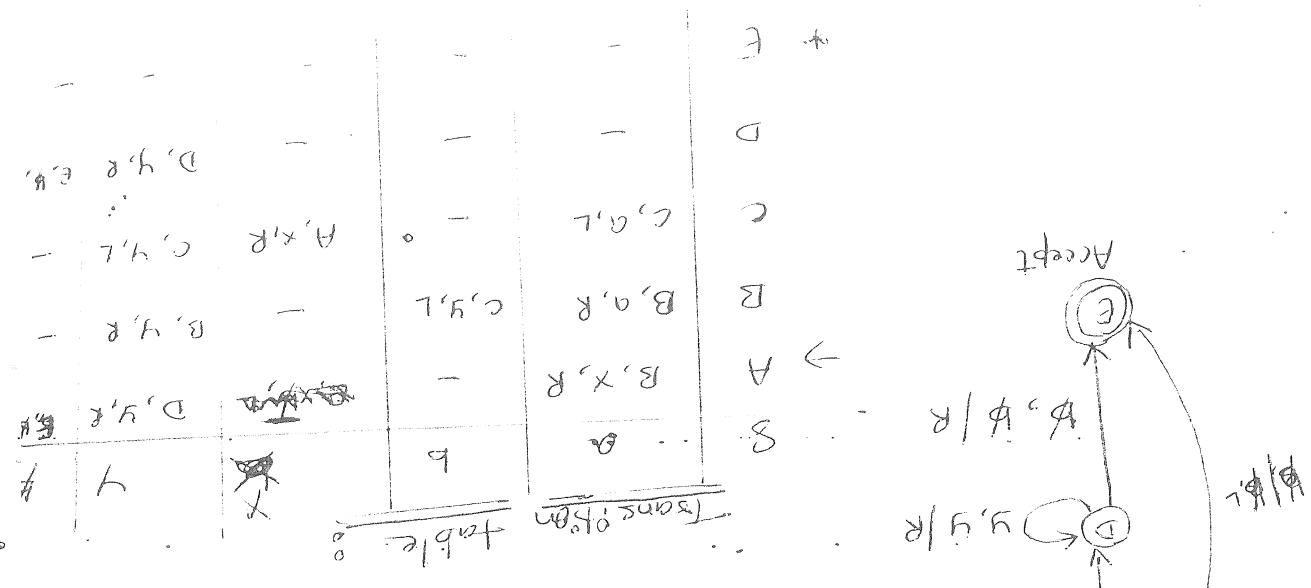
next ID?

(a_3, b_1, R) is the transition, what is the

$a_1 a_3 a_5 a_7 a_9 \dots$ and $S(a_3, a_5) =$

Q) Given the following ID is a Turing machine:

In general, the actions performed by the defences in depends on the easiest step to be scanned to be the whole skipping to be the current position of the head. The action performed by the machine consists of the easiest step to change the state from one state to another. Replacing the symbol pointed to by the head - write head. Movement of the lead - write head towards left or right.



3. change b to y , to be blamed.
4. move left to leftmost a .
5. reflect the above steps until no move is.
6. make sure no more b 's remain.

General Procedure

language: $L = \{ ab^n | n \geq 0 \}$. Also write the instantaneous description for the string "abb".

Design a Turing Machine to accept the following

Sol:-

Read/write head point to the first symbol of the string

Get a all procedure

To state A1. Transposition is not
permitted for the
bank character. So it is leading
state and the starting is reflected.

in h x x +

$$h^{16} \times x^{-1}$$

6.0% X +

box 8 -

$\text{L} \propto \rho^{\frac{1}{2}} N^{\frac{1}{3}}$

卷之二

14 ✓

— 6 —

Accepted

h h x x -

$\lambda h x x t$

b h x -

196 x 1

12

100 X. T

b h x x

16 x 1

$\alpha \times q_0$

$$= a^2 \times a^3$$

$$= \alpha^{\beta_1} \times \dots \times \alpha^{\beta_n}$$

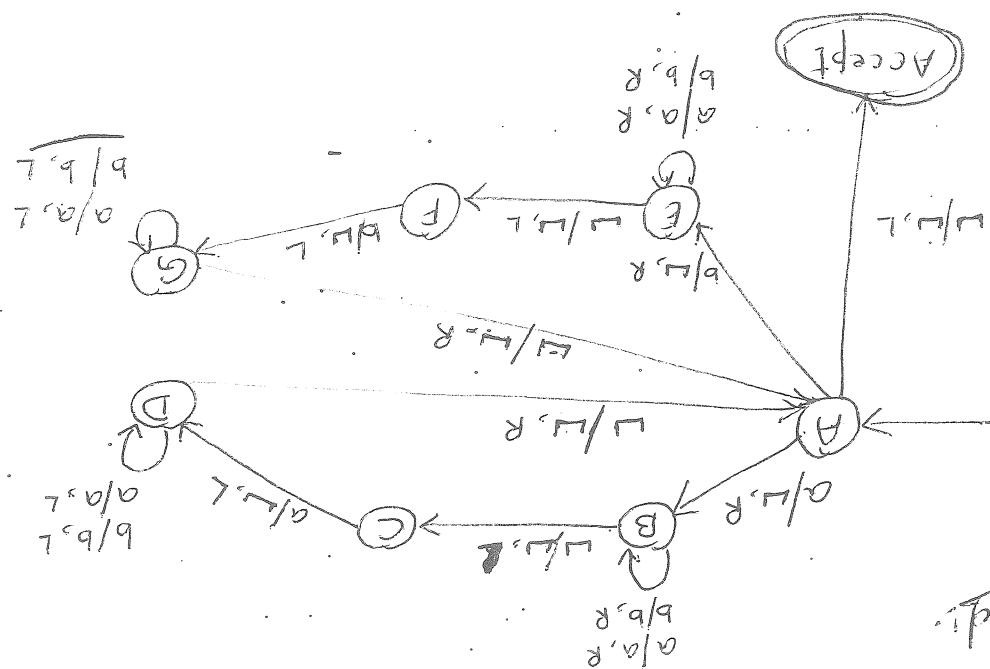
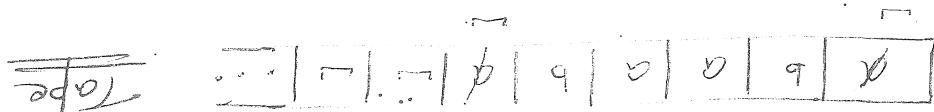
$$H \times a_{\alpha}$$

$$+ x_{\alpha_1} a$$

• 998

•

The procedure continues for the entire string until the
 tape becomes blank. If all cells were zero, then the
 symbol is reflected otherwise it is reflected.
 As the first symbol it was read before. If it is same,
 when it encounters the black character should be same
 as the symbol it is initial if encounter the black character
 i.e. a. When it reads the first symbol, it places it on
 to begin tape head will be pointing to the first symbol
 looks like as shown in the above fig. For the first/
 as already present in the tape. In the tape, it
 Apparatus :- Consider the string "ababa". Assuming it
 to be



Note the turing machine to accept even
 palindromes uses the alphabet $\Sigma = \{a, b\}$

I) ~~for the string abaaba~~

| Table symbol | | | Accept |
|--------------|-----------|-----------|--------|
| → | → | → | → |
| (A, →, R) | (B, →, L) | (C, →, L) | G |
| (D, →, R) | (E, →, L) | (F, →, L) | E |
| (G, →, R) | (H, →, L) | (I, →, L) | D |
| (J, →, R) | (K, →, L) | (L, →, L) | C |
| (M, →, R) | (N, →, L) | (O, →, L) | B |
| (P, →, R) | (Q, →, L) | (R, →, L) | A |
| — | — | — | — |

String accepted:

+ (L M A N U)

+ (L M C A N)

+ (L M A B N)

+ (L M B A N)

(different format) .
 3) So, the transition table for DFA and TM remains same

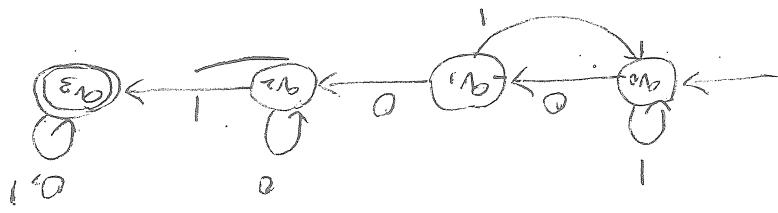
for each scanned input symbol (either 0 or 1),
 to say if 0 and 1 by 1 and the tape head moves
 to the same state in same input symbol, reflecting
 the behaviour of the DFA was in TM also in this
 way.

As the DFA processes the input string from left
 to right in only one direction, TM also processes

Approach :-

| | | | | | |
|--|--|--|----|----|---|
| | | | A3 | A3 | * |
| | | | A2 | A2 | |
| | | | A1 | A1 | |
| | | | A0 | A0 | |
| | | | 0 | 0 | 1 |
| | | | | | |

Transition table



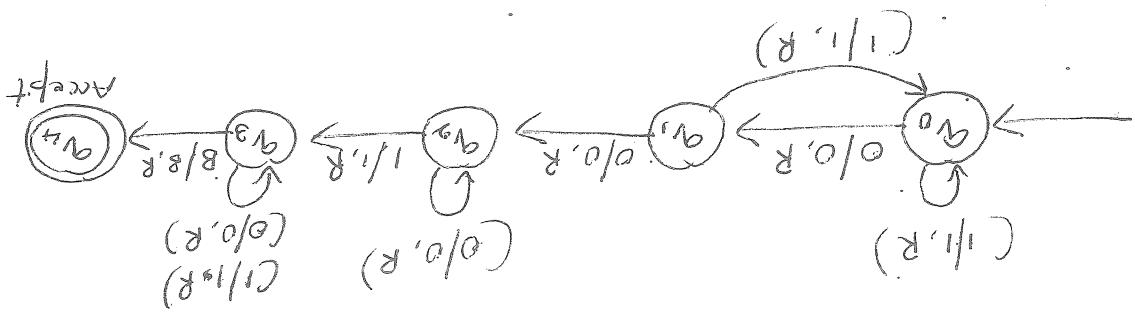
Transition diagram for DFA :-

The DFA which accepts the language consisting of strings of 0's and 1's showing a substring 001
 is shown below:-
 the substring 001
 Design a Turing machine to accept the full language? L = {w | w ∈ {0+1}* containing
 the substring 001} (3)

$\Sigma = \{a_0, a_1, a_2, a_3, a_4\}$...
 $Q = \{q_0, q_1, q_2, q_3, q_4\}$...
 δ is shown in the form of transition table above
 q_0 is the start state
 B is the blank character
 $F = \{q_4\}$ is the final state
 where,
 The TM is given by, $M = (Q, \Sigma, \delta, q_0, B, F)$

| | | | | a_4 |
|--|-------------|-------------|-------------|-------|
| | | | | $-$ |
| | | | $a_3, 0, R$ | a_3 |
| | | $a_3, 1, R$ | a_4, B, R | $-$ |
| | $a_3, 0, R$ | $a_3, 1, R$ | $-$ | a_2 |
| | $a_3, 1, R$ | $-$ | $a_2, 0, R$ | a_2 |
| | $a_2, 0, R$ | $a_2, 1, R$ | $-$ | a_1 |
| | $a_2, 1, R$ | $a_1, 0, R$ | $a_1, 1, R$ | $-$ |
| | $a_1, 0, R$ | $a_1, 1, R$ | $a_0, 0, R$ | a_0 |
| | $a_1, 1, R$ | $a_0, 0, R$ | $a_0, 1, R$ | $-$ |
| | $a_0, 0, R$ | $a_0, 1, R$ | $-$ | S |

Transition table:



Transition diagram for TM:

Design a Tm over $\{1, 3\}$ which can compute

path of words (w_1, w_2) as the input, the output, the output

has to be done

difficult topic

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| B | I | I | I | E | I | I | B | I | B |
|---|---|---|---|---|---|---|---|---|---|

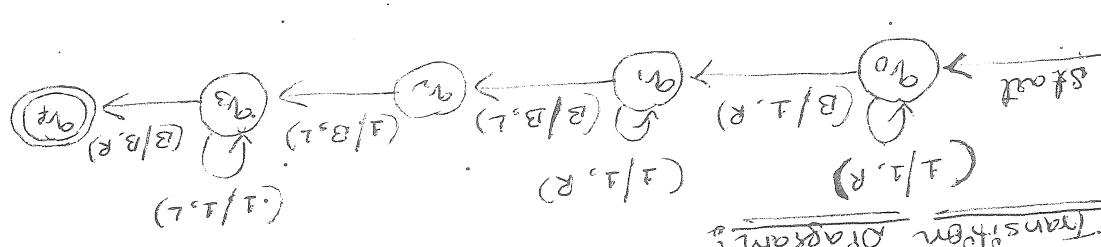
A diagram showing a stack of 10 boxes labeled B1 through B10. Box B5 is highlighted with a bracket above it and an arrow pointing to the word "box".

Impairment

Sol: Design considers the following as $= 11$. $\omega_1 = \omega_2 = 11$.

a generalization function over \mathcal{X} and \mathcal{Y} based on pairs of words (w_i, w_j) as the input, the output

has to be done.



Transcription of aogram

end

because that in the first fig., B is in between head and tail. In the second fig., B is moved to the

1920

-6as 1m

二二二二二二二二

John M. Utley

Consider the design of

100% of the time, I am not able to do what I want to do.

has to be written.

Patent & Trademark Office

83-8

Design a Tm Qvs

[View all posts by admin](#) | [View all posts in category](#)

(q_3, A, R)

(q_2, A, R)

(q_1, A, R)

(q_0, A, R)

$(q_0, 1, R)$

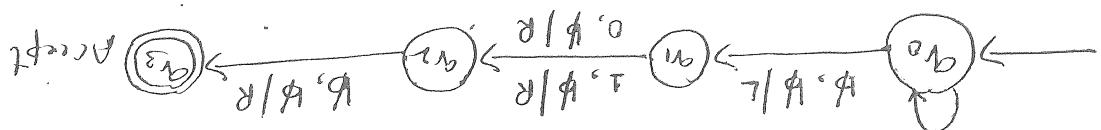
$(q_1, 1, R)$

$\#$

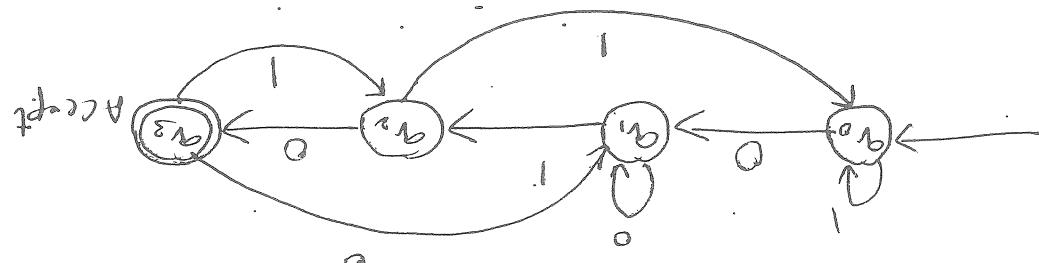
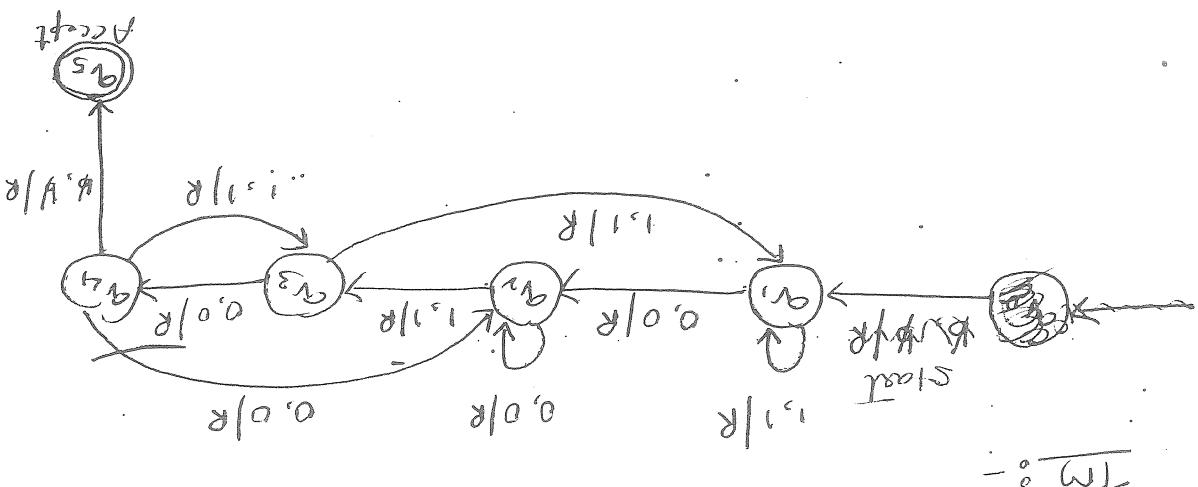
1

0

S



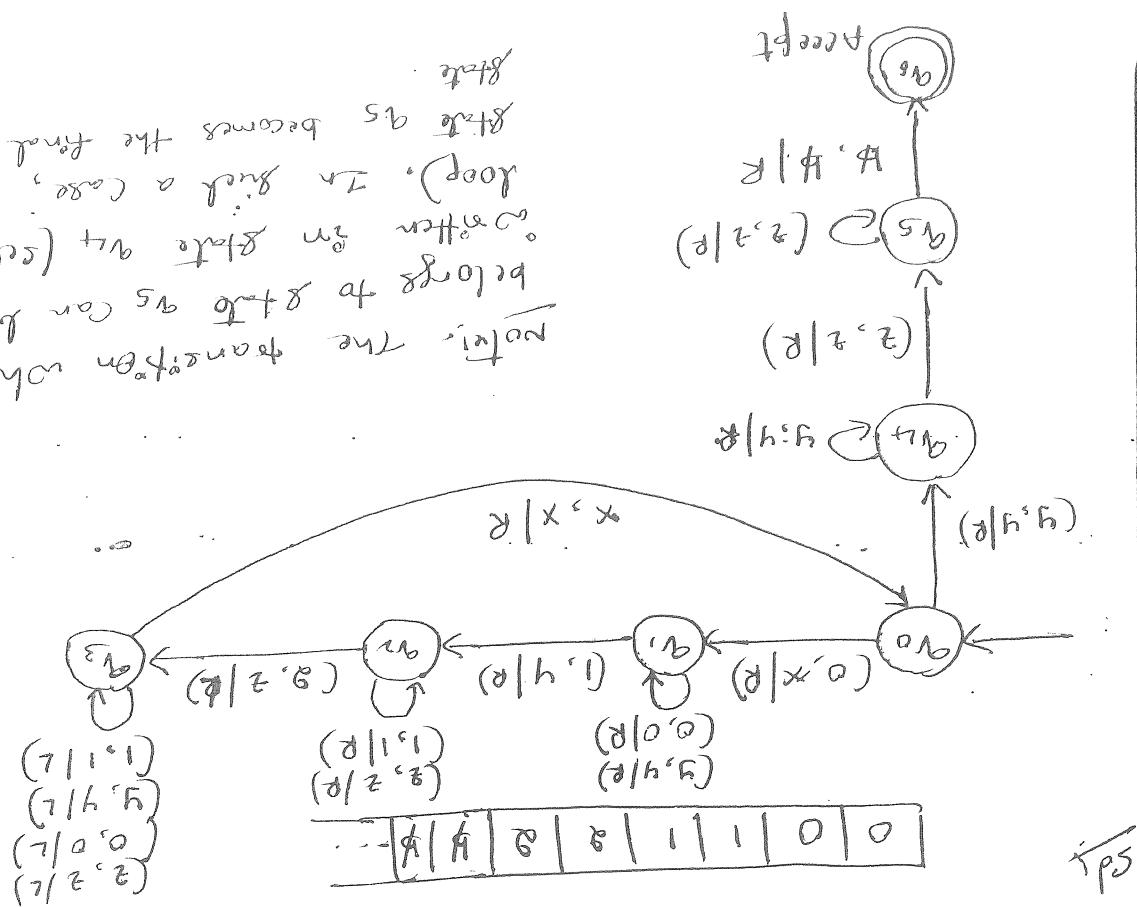
Q) Design a TM on $\Sigma = \{0, 1\}$ which accepts strings of length most



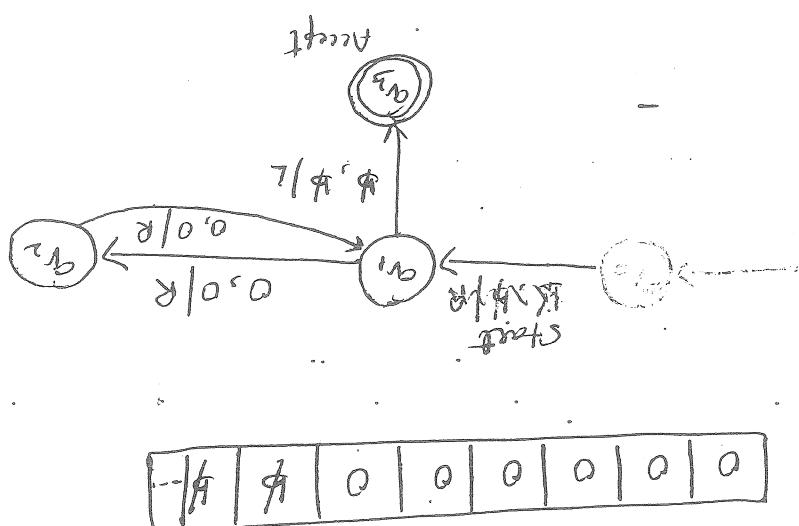
Q) DFSM :-

5) Construct a TM to accept strings ending with 010
giving DFA $\Sigma = \{0, 1\}$ ending with 010

After the transition to state q_4 , the transition which belongs to q_4 as q_4 is a cause, another one in state q_4 (say q_5) becomes the final state. It will be a loop. As q_5 becomes the final state, the transition to q_5 as q_5 is a cause, and so on. Thus, the transition which belongs to q_5 as q_5 becomes the final state. It will be a loop. As q_5 becomes the final state, the transition to q_5 as q_5 is a cause, and so on.



(8) Design a TM that accepts $L = \{0^n | n \geq 1\}$



| | | | | | | |
|---------|---------|---------|---------|---------|---------|---------|
| $m > 0$ | $m = 1$ | $m = 2$ | $m = 3$ | $m = 4$ | $m = 5$ | $m = 6$ |
| 0^m | 0^1 | 0^2 | 0^3 | 0^4 | 0^5 | 0^6 |

(a) Design a TM that accepts $L = \{0^n | n \geq 1\}$

$M = (Q, \Sigma, \Gamma, S, q_0, B, F)$ where
given by

The final difference of TM with start symbol is
head either towards left or towards right.
and changes its state from a to p without updating P/I
the input symbol a , replace the input symbol a by p .
This means that, the TM in state a , after consuming

$$S(a, a) = (p, q, S)$$

implemented using the transitions?

for some input symbols can be implemented
by head such that the P/I head remains in the same
place, move either left or towards right. This can be
done after consuming the input symbol a , the P/I
so p can be left as right denoted by l or r
where l stands for direction.

$$S(a, a) = (p, q, \Sigma)$$

1. Turing machine with start symbol head :
In standard TM S is defined as :

↳ couples

↳ multiple track TM

↳ storage in the state

↳ TM with start symbol head

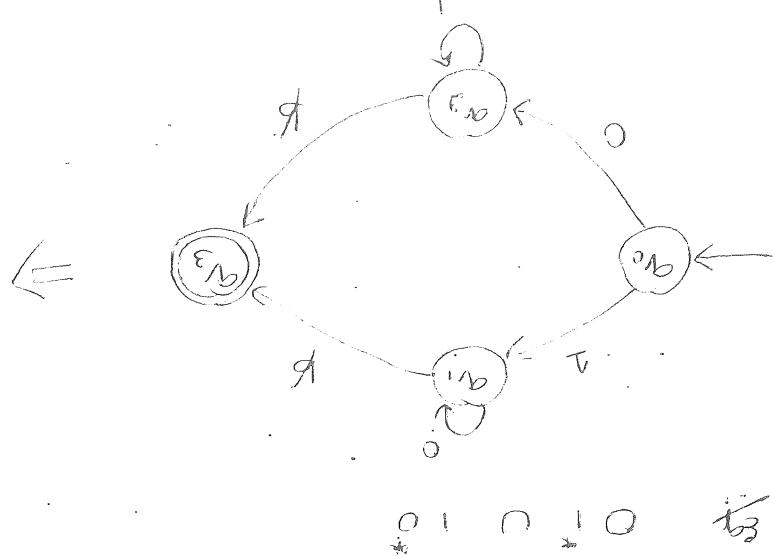
What are the various techniques for TM construction?

Now, let us discuss some high-level conceptual tools using which we can construct TMs very easily.

far is called "standard turing machine".

The TM that we have discussed so

Techniques for TM construction



Q, O, I, U, I, Q

In all our machines such as FA / PDA / TM, we used states to remember things. Apart from this, we can use a state to store a symbol as well. So, in TM where we use state in a tape, the state and below it a pair (a, a) where a is the state and a is the tape symbol stored in the state (a, a) . So, the new set of states is $Q \times T$, where we use state a in a tape in a state b .

g. Storage in the state

\leftarrow set of final states

\leftarrow special symbol indicating blank character

\leftarrow start state

\leftarrow update symbol on the tape

\leftarrow symbol at the same position in the tape

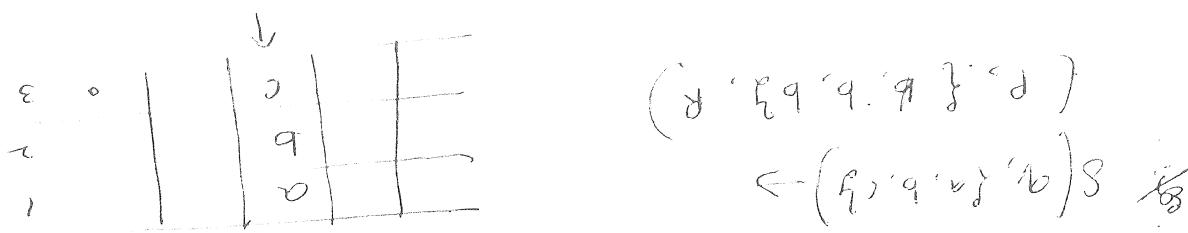
\leftarrow indicating the TM moves towards left

\leftarrow transition function from $Q \times T$ to $Q \times T \times \{L, R\}$

\leftarrow set of tape symbols

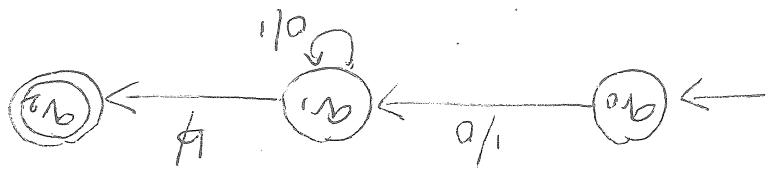
\leftarrow input alphabet

\leftarrow set of final states



terminates same as that. In standard Turing machine, the tape symbols are denoted by $\frac{1}{T_k}$, the reading symbol T whereas in case of TM, with multiple tracks, of standard TM, tape symbols are denoted by multiple tracks as the set of tape symbols. Only difference between standard TM and TM is with alphabet consists of tuples of tape symbols. In case of single tape of divided into n tracks, then the tape to be divided into a number of tracks. If a TM only one tape was used to store the data. In multiple track Turing machine: In a standard

$$\begin{aligned}
 & S(a_0, a_1, a_2, a_3) \leftarrow S(a_0, a_1, a_2, a_3) \\
 & S(a_0, a_1, a_2, a_3) \leftarrow S(a_0, a_1, a_2, a_3) \\
 & S(a_0, a_1, a_2, a_3) \leftarrow S(a_0, a_1, a_2, a_3) \\
 & S(a_0, a_1, a_2, a_3) \leftarrow S(a_0, a_1, a_2, a_3) \\
 & S(a_0, a_1, a_2, a_3) \leftarrow S(a_0, a_1, a_2, a_3)
 \end{aligned}$$



4. Subroutines: we know that subroutines are used in programming languages whereas a task has to be done repeatedly. The same function can be used in TM and complicated TMs can be built using subroutines. A TM subroutine is a set of states that performs some pre-defined tasks. The TM subroutine has a start state and passes the control to the start which has no moves because of the reason. and a state without any move. This state is separated by the delimiter. So if we assume we have two unary numbers x and y such that x has m number of 1's and y has n number of 1's and x and y should be stored in the board of x and y should be stored in the tape and the original numbers should be displayed. This can be visualized as shown below:

If $x = 00$ and $y = 0000$ and are separated by commas then it is followed by blank spaces at the ends with which can be assumed to be input string of blankes (B) as the delimiter for the output string which is given by infinite sequence of numbers at the following lines.

Now let us assume we have two unary numbers x and y such that x has m number of 1's and y has n number of 1's and x and y should be stored in the board of x and y should be stored in the tape and the original numbers should be displayed. This can be visualized as shown below:

Call the subroutine. Design a TM to multiply two unary numbers separated by the delimiter. Let us assume we have two unary numbers x and y such that x has m number of 1's and y has n number of 1's and x and y should be stored in the board of x and y should be stored in the tape and the original numbers should be displayed. This can be visualized as the following lines.

Step 2: Now, change C to P to E

Note that all B's will be replaced by the result [Qm (also)] .

(gross III) (gross II) (gross I)
ab ab ab
~~~~~ ~~~~~ ~~~~~  
m m m B B B . . .

To start with let us assume a shape such that a zigzag of the form  $m_1 m_2 B B B \dots$  which has decided info that it goes as shown below:

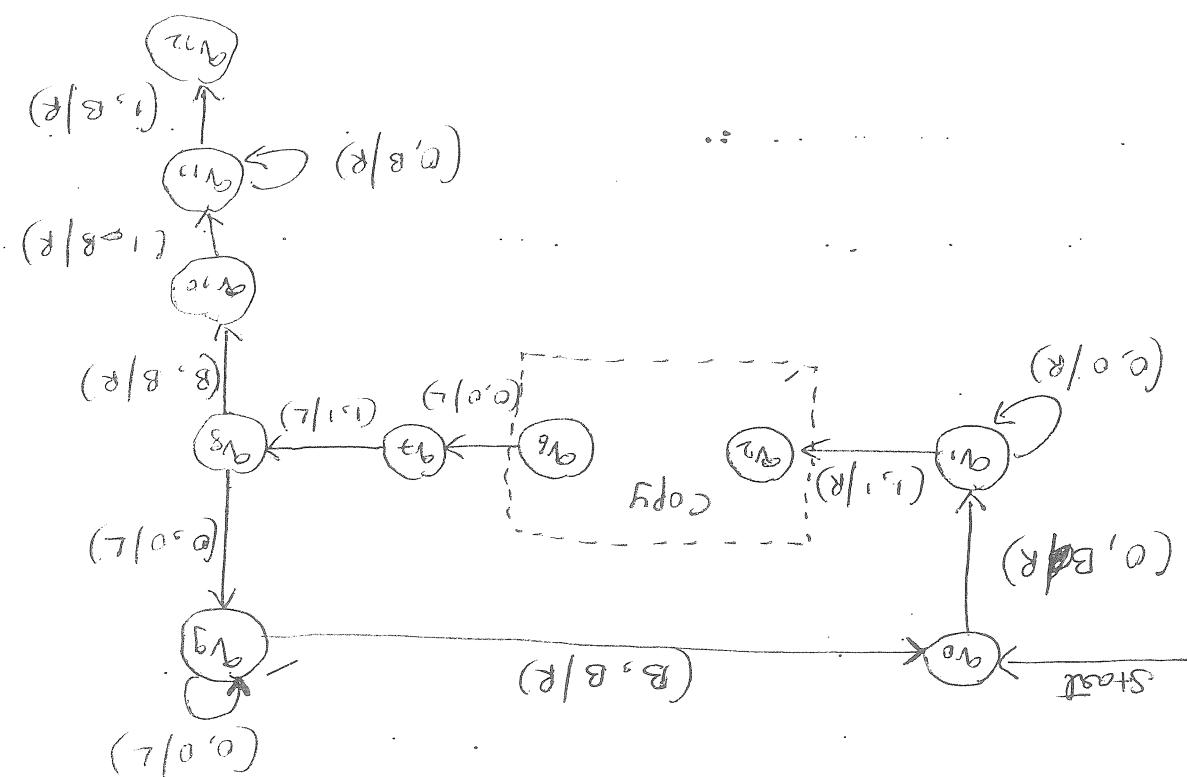
General procedure: (Algorythm)

— The output should be 00000000 //  $4 \times 2 = 8$   
So, to start. With we think 8 hours to copy the  
unary numbers by hand so that we can call this  
term (which is expressed as a 8 digit unary)  
separately in times to achieve the result.

Note: It is clearly visible from the above figure that the unary number  $y$  is equal to the numbers  $y$  in the numbers  $x$  in the result  $X$ .

$$\overbrace{00000000}^{\text{fix}} + \overbrace{00000}^F + \overbrace{00}^x = 888$$

The outfit should be fit for the few



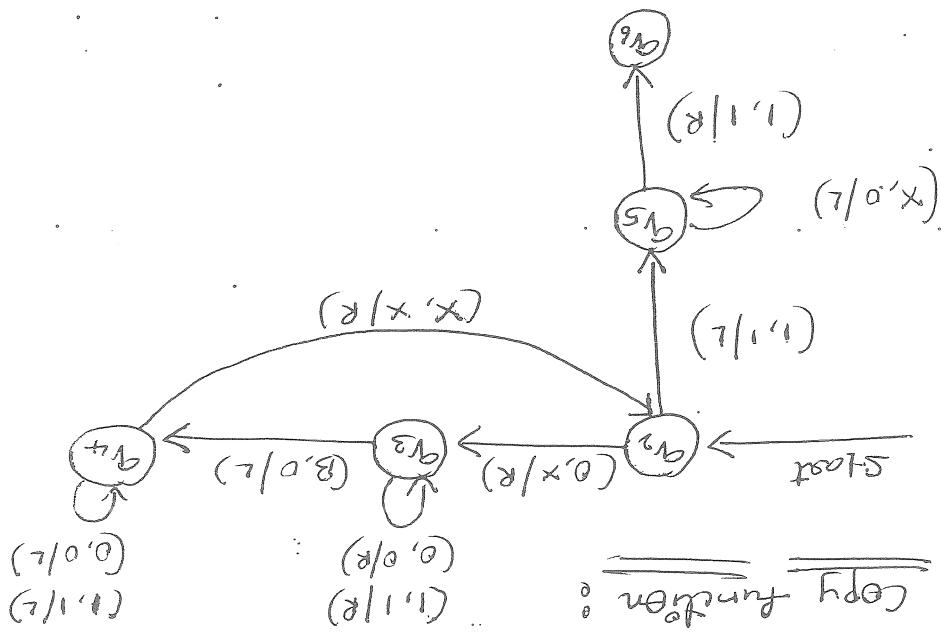
Note: If we clear from second step that "we copy a group & a is from a by reflecting it in B's in B's in A to n as when a is reflected to B". When all this is in P. are changed to B's when all this is in P. are changed to B's. These will be definitely be the same number of B's as shown below.

appears in p.

Stage 3: Perfect above two steps till no change.

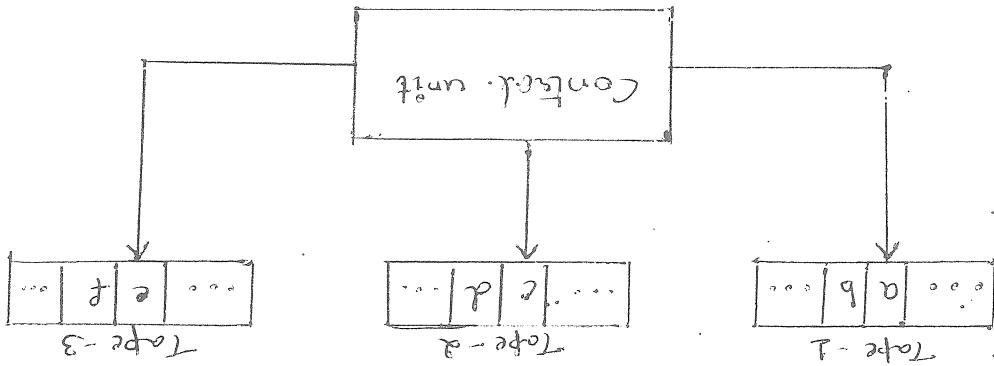
So as shown using the transition diagram  
 $\Delta = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}, q_{11}, q_{12}\}$   
 $\Sigma = \{0, 1, x, B\}$   
 $Z = \{0, 1\}$   
 $L = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}, q_{11}, q_{12}\}$   
 where,  
 $M = (\Delta, \Sigma, \Gamma, S, q_0, B, F)$

The Tm to accept the given language is given



← Each tape is divided into cells above the following points from above  
← Each symbol from the given alphabet  
← The start with, the TM should be

The value components of multiple-type in use are? → Finite control ← Multiple type heads. where each type having its own symbols and rule head. ←



A. Multis-tape TM :  
A multis-tape Turing machine is nothing but a standard turing machine with more number of tapes. Each tape is controlled independently with independent read-write head.  
The potential representation of multitape TM with n tapes is shown below:

- g. N.B. - deterministic

1. multi-tape Tm

Two variables of the are -

Two variables of the are -

Vasanta's Flying Machine :

If the R/W head is pointing to tape 1 moves towards right, the R/W head pointing to tape 2 and towards left, the move made towards right and the move made towards left depends on the move 3 made by the R/W head.

The move of the number-tape in TM is 3 and if there is a transposition then the move 3 is

$S(a, a, b, c) = (p, x, y, z, l, g)$

where  $a$  is the current state.

The transposition can be interpreted as follows:

The TM in state  $a$  will be moved to state  $p$  only when the first R/W head points to  $b$  and the second R/W head points to  $c$ .

Second R/W head moves to  $b$  and the first R/W head moves to  $c$ . The head moves to  $b$  and the head moves to  $c$  only when the second R/W head points to  $a$ ; the case and except in the second case. But, the R/W head with respect to the third tape will not be altered.

At the same time, the symbols  $a$ ,  $b$  and  $c$  can be replaced by  $x$ ,  $y$  and  $z$ .

The formal definition of number-tape in TM can be defined as follows:

Keenly the number-tape in TM is an n-tape machine  $M = (Q, Z, T, S, V_0, B^3, F)$ .

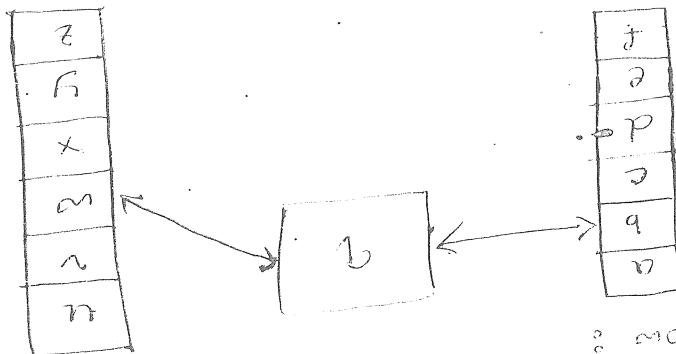
These 3 tapes are of type 1 and 2 and 3 respectively.

Q is the set of states, S is the input alphabet,  $V_0$  is the initial state,  $B^3$  is the blank symbol,  $F$  is the final configuration of the n-tape machine.

shown below:

|   |   |   |   |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| o | 5 | 6 | c |
| o | x | o | d |
| o | m | l | l |
| o | v | o | b |
| o | z | o | a |

using single tape in which this can be simulated as shown above two-tape (V) can do the same tasks as multiple tape single tape takes multiple tape - single tape.



For example, consider a TM with three tape heads as shown below:  
 These can be shown by simulation.  
 The power is by constructing a new machine.  
 Proof:-

Theorem: Every language acceptable by standard Turing machine with single tape is accepted by a multi-tape TM.

In fact it is equivalent to the single tape if it can be easily shown that the one-tape TM

$E \leftarrow \text{ed. of final state}$   
 $B \leftarrow \text{Blank states}$   
 $A_0 \leftarrow \text{Start state}$

$S \leftarrow \text{Transition function from } Q \times P \text{ to } Q \times T^*$   
 $\{L, R\}$

The first and third tracks consist of symbols from first and second tape heads respectively. The second and fourth tracks consist of symbols from second and fourth tape heads respectively. The value of each track is the sum of the values of the symbols it contains. The machine can enter from one state to another, if and only if there is a transition for mult-tape TM, if we apply the same definition for the new machine that we did for mult-tape TM. So, what happens to the new machine that we did for the same machine? The difference between new-deterministic TM and deterministic TM lies only in the definition of the formal definition of new-deterministic TM is the set of finite states  $M = (Q, \Sigma, \Delta, \delta, q_0, F)$  where  $\delta$  is a function of the input alphabet  $\Sigma$ , set of finite states  $Q$  and initial state  $q_0$ . The set of tape symbols  $\Delta$  is a subset of the set of input alphabets  $\Sigma$ . The set of tape symbols  $\Delta$  is a subset of the set of input alphabets  $\Sigma$ .

S is a translation function which is a

g

- ulges of  $\mathbb{Q} \times \mathbb{T} \times \{L, R\}$

$a_0 \hookrightarrow$  start state

B  $\hookrightarrow$  blank character

E  $\hookrightarrow$  set of final states

It is clear from the definition of S that for each state q and tape symbol  $x$ ,  $S(q, x)$  is a triple:

$\{ (q_1, x_1, D), (q_2, x_2, D), \dots, (q_n, x_n, D) \}$

where,

$q \hookrightarrow$  finite interface

D  $\hookrightarrow$  transition function Left / Right

The machine can choose any of the triples as the next move.

RNS17  
Dept. of CSE,  
Ass't. Prof.,  
RASHMI. M

Module - 5

Decidability of Turing Machine

What is an algorithm? Is the algorithm can be executed by Turing machine?

If an algorithm terminates eventually, the TM Turing machine halts in foll. two situations:

→ When a TM reaches a final state; then it halts.

→ When a TM reaches a symbol  $s(a, a)$  as next

↳ When a TM reaches a state  $q$  and the input symbol is a and if the transition  $S(q, a)$  is not defined, then it halts.

Decidability:

The Church-Turing thesis states that any problem solvable with all unknowns having integer values with the help of a computer and a finite number of general algorithm which, for any given polynomial equation has a unique solution can decide whether the equation has a finite number of solutions as a challenge to provide a feasible method to attack Hilbert's tenth problem.

Hilbert's tenth problem is a challenge to provide a method for an algorithm. The ten was therefore called undecidable model for an algorithm. Thus, the TM is considered as an ideal machine.

A computer, can also be considered as a Turing algorithm that can be solved out by a running procedure to solve a given problem. The procedure is to complete a task. The algorithm is terminated after finite sequence of instructions that have to be executed to complete a step.

The Church-Turing thesis states that any algorithm that any

## Decidability of Complexity

design problems, we can design design algorithms, called decision problems. These may satisfy the answer yes/no, accept/reject, finite/infinite, finite or infinite. All such problems with the same have to identify whether the language is going to be yes/no. Given a language and a string  $x$ , does it accept  $x$ ? The output problem can be solved as "Given a machine M membership problem. Formally, the membership may be accept/reject. This problem is called a language. So, the output of the machine rejects a language. Therefore the machine accepts a language if it meets

If  $w \in L$ , then  $w$  is accepted by TM in reaching the final state and the machine halts  
If  $w \notin L$ , then  $w$  is rejected by TM without reaching the final state and the machine halts  
for two conditions:  
if these exists a TM that satisfies the  
if language  $L \subseteq \Sigma^*$  is recursive if and  
only if we L then  $w$  is accepted by TM in  
reaching the final state and the machine halts

What is recursive language?  
language accepted by turing machine.

(w) such that  $L = T(w)$  where  $T(w)$  is the enumerable if and only if there exists a TM  
A language  $L \subseteq \Sigma^*$  is recursive if  
What is recursively enumerable language?  
some inputs and a TM that meets halts on some  
languages that are accepted by TM and halts on  
we have to make a distinction between the  
on some inputs in any of the above situations. So,  
But, there are some TMs that meet that

Define decidable language and undecidable language

A decision problem of decidable language with yes/no answer is decidable if and only if the corresponding language is recursive. In other words, a decidable language is a recursive language with words, a decidable language is a language which has to consist of a finite number of strings and accepts L(M). We know that DFA always ends in a same state after executing the string to then  $(M, w)$  is part of the language made by DFA.

Let us define TM as shown below

4. If  $w$  is the string to be solved by DFA then  $(M, w)$  is part of the language made by DFA.

3. Simulate  $M$  and report  $w$  to TM in. Here,  $w$  is simulated in and input  $(M, w)$  is valid input.

2. Check whether input  $(M, w)$  is valid input.

1. If  $M$  is invalid, the TM is also invalid.

It updates the state and reading the next character. Otherwise, if  $M$  is valid input then  $(M, w)$  is invalid. The TM is also invalid.

2. Simulate  $M$  and report  $w$  to TM in. Here,  $w$  is simulated in and output  $(M, w)$  is valid input.

3. If  $M$  is valid state and reading the next character. If  $M$  is invalid state and reading the symbol or and  $L(M)$  is valid input, then  $(M, w)$  is valid input.

4. If  $M$  is valid state and reading the next character. If  $M$  is invalid state and reading the symbol or and  $L(M)$  is invalid, then  $(M, w)$  is invalid.

3. The simulation ends in a final state and  $L(M)$  is accepted.

2. If  $M$  is accepted then  $(M, w)$  is accepted.

1. If  $M$  is rejected then  $(M, w)$  is rejected.

Proof :- Let  $M = (Q, \Sigma, S, q_0, F)$  be a DFA. we have to construct a TM  $M'$  such that always halts and accepts  $L(M)$ . We know that DFA always ends in a same state after executing the string to then  $(M, w)$  is part of the language made by DFA.

Now, we will construct a TM in which simulate now, we will construct a TM in which simulate the string  $w$  and accept it if it belongs to  $L(M)$ .

Let us define  $TM$  as shown below

Prove that DFA is decidable language.

Decidable languages

A decision problem of decidable language with yes/no answer is decidable if and only if the corresponding language is recursive. In other words, a decidable language is a recursive language with words, a decidable language is a language which has to consist of a finite number of strings and accepts  $L(M)$ . Otherwise, it is undecidable language.

yes/no answer is decidable if and only if the corresponding language is recursive. In other words, a decidable language is a recursive language which has to consist of a finite number of strings and accepts  $L(M)$ . Otherwise, it is undecidable language.

A decision problem of decidable language with yes/no answer is decidable if and only if the corresponding language is recursive. In other words, a decidable language is a recursive language which has to consist of a finite number of strings and accepts  $L(M)$ . Otherwise, it is undecidable language.

If a language is not accepted by a Turing machine, then the language is not recursively enumerable. One simple problem is given a Turing machine. If it halts on input  $x$ , then we can say that  $x$  is in the language. If it does not halt, then we can say that  $x$  is not in the language. This is called a many-one reduction. In other words, if  $M$  is a Turing machine and  $N$  is another machine, then  $M \leq_m N$  if there is a many-one reduction from  $M$  to  $N$ . This means that if  $x \in L(M)$ , then  $\phi(x) \in L(N)$ . If  $x \notin L(M)$ , then  $\phi(x) \notin L(N)$ .

Halting Problem:

Given a Turing machine  $M$  and an input  $x$ , determine whether  $M$  halts on  $x$ . This is an undecidable problem. We can prove this by contradiction. Assume that there exists a Turing machine  $H$  that can decide the halting problem. Then we can construct a new Turing machine  $M'$  such that  $M'$  takes an input  $x$  and runs  $H$  on  $x$ . If  $H$  halts, then  $M'$  accepts  $x$ . If  $H$  does not halt, then  $M'$  rejects  $x$ . Now consider what happens when  $M'$  runs on its own description. If  $M'$  accepts its own description, then  $H$  must not halt on its own description, which contradicts our assumption that  $H$  decides the halting problem. If  $M'$  rejects its own description, then  $H$  must halt on its own description, which also contradicts our assumption that  $H$  decides the halting problem.

Proof: Let  $L$  be a decidable and get a contradiction  
 since that halting problem of Turing machine is undecidable.  
 Then  $A$  is undecidable  
 $\Leftrightarrow$  If  $A$  is reducible to  $B$  and  $B$  is undecidable,  
 then,  $A$  is decidable  
 $\Leftrightarrow$  If  $A$  is reducible to  $B$  and  $B$  is decidable,  
 $L(A) = L(B)$  and let  $M_1$  that eventually halts  
 in all  $(m, w)$ . Let us consider turing machine  
 $f$  for  $Tm\ m_2, (m, w)$  is input  
 as follows:  
 1) If  $Tm\ m_1$  accept  $(m, w)$ , simulate  $Tm\ m_2$   
 on the input  $g(m)$  to until  $m_2$  halts  
 2) If  $Tm\ m_1$  accept  $(m, w)$ , then  $Tm\ m_2$   
 rejects  $(m, w)$ .  
 3) If  $Tm\ m_1$  rejects  $(m, w)$  then  $Tm\ m_2$   
 accepts  $(m, w)$ ; otherwise we  
 4) If  $Tm\ m_1$  rejects  $(m, w)$ , then  $Tm\ m_2$   
 rejects  $(m, w)$ .  
 5) If Turing machine  $m$  has accepted or  
 rejected, so  $Tm\ m_2$  accept or  
 rejects the following pair of pairs:  
 When  $m_1$  accept  $(m, w)$  (see step 4), the  
 in the first case (first a halting in step 5)  
 in  $m$  halts in  $w$ .  
 in the second case (second a halting in step 5)  
 in  $m$  accept  $(m, w)$ .

$V_1 = 111$ ,  $V_2 = 100$ ,  $V_3 = 111$

$w_1 = 11$ ,  $w_2 = 100$ ,  $w_3 = 11$

Given two lists A, B. Where A is even, and B is odd. Find whether these two lists are a Post Correspondence Problem solution or not.

Example :-

The Post Correspondence Problem will follow if there exists a pair-solution such that the algorithm finds a pair-solution for each pair of consecutive pairs in the given two lists.

Solution for pair (A, B) is achievable if we say that these exists a sequence of integers such that  $w_1 w_2 \dots w_k = v_1 v_2 \dots v_n$ .

Definition :- The Post Correspondence Problem can be stated as follows. Given two sequences  $A = w_1, w_2, \dots, w_m$  and  $B = v_1, v_2, \dots, v_n$ . Then there exists a sequence of integers  $k$  such that  $w_1 w_2 \dots w_k = v_1 v_2 \dots v_n$  and  $v_1 v_2 \dots v_n = w_1 w_2 \dots w_k$ .

Post Correspondence Problem :-

So, it follows from definition of the Turing machine that eventually and hence,  $L(M) = \{(m, n)\}$  : The following machine is acceptable to which is undecidable. This is a contradiction. So, the turing machine accepts an input in the undecidable.

Algorithm is a polynomial time  
problem can be solved by a deterministic  
standard polynomial time algorithm

Observe the following points:

This is still open and any one can characterize this  
theory, many scientists believe that P ≠ NP. But  
it's necessary to solve it completely. In complexity  
amount of computation time and memory requirements  
solvable in practice. But, it may require enormous  
resources that the problem is computationally  
when a problem is manageable & decidable, it

### Complexity

longer than the computation time  
any string composed of elements of A will be  
these cannot be any PC-calculator simply because

$$w_1 = 0, w_2 = 11, w_3 = 011$$

$$w_4 = 00, w_5 = 001, w_6 = 1000$$

If we take

$$\begin{array}{r} \overline{11100111} \\ - \quad \textcircled{3} \quad \textcircled{3} \quad \textcircled{3} \\ \hline 11100111 \\ \quad \quad \quad \boxed{\begin{array}{c} 11 \\ \hline 111 \end{array}} \quad \boxed{\begin{array}{c} 100 \\ \hline 001 \end{array}} \quad \boxed{\begin{array}{c} 111 \\ \hline 11 \end{array}} \\ \quad \quad \quad 0-01 \end{array}$$

as shown below:

For this case, there exist a PC-solution

$$\begin{array}{r} \textcircled{3}. \quad \textcircled{3}. \quad \textcircled{3} \\ \boxed{\begin{array}{c} 11 \\ \hline 111 \end{array}} \quad \boxed{\begin{array}{c} 100 \\ \hline 001 \end{array}} \quad \boxed{\begin{array}{c} 111 \\ \hline 11 \end{array}} \\ \quad \quad \quad \frac{100}{100} \end{array}$$

in which case it is called defensive TM.  
 making maximum of  $T(n)$  measure. In this case,  
 $T(n) \cdot T(n)$  is added to the TM that starts after  
 time complexity of a Turing machine is given by  
Definition: Given an input string  $w$  length  $n$ , the  
 length of a string to be output  
 which satisfies  $C + g(n)$  is  
What is the time complexity of a Turing machine

### Class P and NP:

$f(n) \leq C + g(n)$  for all  $n \geq n_0$   
 and positive  $n$  to get no satisfying the constraint  
 if and only if these exists a positive constant  $c$   
 $(f(n)) \leq C + g(n)$   
 $O(g(n))$  denoted by

Algorithm: The function  $f(n)$  is said to be  
Efficient: Let  $f(b)$  be the time efficiency by an

word is defined as follows  
 efficient. so normally we use big-O notation  
 algorithm to find out what is the most efficient  
 problem. it is necessary to compare more  
 given two algorithms to solve the same  
Same the Rate of Functions

Time: This class of problems can be solved  
 by a non-deferential algorithm in a  
 polynomial time  
NP stands for non-deterministic polynomial

Define class P with respect to Turing machine

Let  $L(n)$  be the language accepted by  $\hat{M}(n)$ . Then  $\hat{M}(n)$  with time complexity  $p(n)$  such that  $L = L(n)$  for some language  $L$  as in class NP if there exists some Turing machine with time complexity  $T(n)$ .

Step 1 :- In Step 2 :- we constructed a machine, use the above procedure that the number of moves =  $q_n$  =  $n^2$  (By neglecting the constant).

So, the time complexity is given by  $o(n^2)$ .

Step 2 :- When we consider the following machine for the above language, every letter a is replaced by  $x$  and every letter b is replaced by  $y$  and every letter c is replaced by  $z$ .

The total number of moves =  $q_n$ .

Step 3 :- Let  $L = f(a, b, c)$  be a Turing machine defined over  $\{a, b, c\}$ . Then we consider the following machine with time complexity  $T(n)$ .

Let  $L(n)$  be the language accepted by  $\hat{M}(n)$ . Then  $\hat{M}(n)$  with time complexity  $p(n)$  such that  $L = L(n)$  for some language  $L$  as in class P if there exists some Turing machine with time complexity  $T(n)$ .

Quantum Computer

→ The qualities of  $\alpha$  and  $\beta$  are called basis states  $\alpha < \alpha | 0 < \alpha | \beta < \beta | 0 < \beta | \beta$  as called superposition

$$1 = |\alpha|^2 + |\beta|^2$$
$$\alpha | 0 < \alpha | \beta < \beta | 0 < \beta |$$

$|\alpha\rangle$  and  $|\beta\rangle$  can be combined as linear superposition of states other than like classical bit as  $\alpha$ , a quantum can

→ Two possible states are expressed as Beamer

The output can be obtained as shown

$$\langle \alpha | \beta + \langle \beta | \alpha = \langle \alpha | \beta$$

as superposed quantum mechanically as  $|\psi\rangle = |\alpha\rangle + |\beta\rangle$ . The data in digital computer is represented using binary digits or and transistors. The data in digital computer is designed electronic computers are based on quantum computers are different from binary computers as quantum computers are multilevel computers based on quantum mechanics.

The computers that are built based on called quantum computers the principles of quantum mechanics are the principles of quantum mechanics are called quantum computers

Quantum Computer?

Quantum computation

$\langle B \rangle = \sum |B_i|^2$

Thus, a quantum computer can be defined as

A quantum system built from numbers of quantum gates such as CNOT, ANIS, NEAR etc, do calculations involving wires and elements of quantum circuitry quantum computation.

Carry out manipulation of quantum information.

Whereas in case of a classical computer,  $\alpha_{00} + \alpha_{01} + \alpha_{10} + \alpha_{11}$  is changed to  $\alpha_{00} + \alpha_{01} + \alpha_{10} + \alpha_{11}$

$\leftarrow$  Normal NOT gate is implemented on  $\alpha_{00}$  and  $\alpha_{11}$

where  $|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$

$|\alpha_{00}|^2 + |\alpha_{11}|^2 = |\alpha_{00}|^2 + |\alpha_{10}|^2$

$\leftarrow$  Multiple qubits can be defined in a quantum state as represented as  $|00\rangle, |01\rangle, |10\rangle, |11\rangle$  and states,  $|00\rangle, |01\rangle, |10\rangle, |11\rangle$  and  $|000\rangle, |001\rangle, |010\rangle, |011\rangle, |100\rangle, |101\rangle, |110\rangle, |111\rangle$  are called basis states.

Quantum system has to be in superposition basis.

Binary number. For example, a two-bit binary number can be defined in a

$\leftarrow$  In digital computer, data can be represented as 0 or 1. But, it is not possible to determine the quantum state in observation. In quantum computation we get a state  $|0\rangle$  with probability  $|\alpha_0|^2$  and a state  $|1\rangle$  with probability  $|\alpha_1|^2$

Various forms made by computers such as  
such as recursive functions and post systems  
such as permutations of three persons  
such as stabilised by three persons  
such as A. church, S.C. Kleene and E. Post  
such as it can be censured system  
it and easily if it can be censured system  
the basic function by recursive system  
and primitive recursive system. A post system  
such as to predict next generation  
such as an application and some prediction  
such as by which successive stages can be derived  
in addition to recursive functions and  
post systems, many forms of computers and  
models have been proposed. In extension  
it was found that through the unpredictable  
models looked quite different, they expected  
the same thing. The association was  
followed in such a way that each model is built  
according to its own principles, can be used  
by a human being as a team of engineers.  
In addition there is a classification of computers  
by same tuning machine.

Church-Turing Hypothesis / (Church - Turing Thesis)

This thereby maintains that all the models of computations that are proposed and yet to be proposed are equivalent in their power to recognize languages of complex functions. This is because the above statement is known as "Church thesis". Some named after the logician A. Church. Since the church's thesis is closely related to Turing's thesis which states that we cannot go beyond the limits made by computers that are called church-Turing thesis.

The existing ones

These models of computation more powerful than models of computation named after the logician A. Church. Since these are called church-Turing thesis.

Since there is no precise definition for "effective computation" as there is no precise definition for "algorithm". So, this statement is not proved as the same time it has been not been disproved even though it is simply stated and proved, nor

magically by scientists have accumulated enough evidence to be believed that the power of standard machines like computer can be matched by any other machine.

Even though it is generally accepted that the theory of computation has come to a standstill, now

These models of computation more powerful than models of computation named after the logician A. Church. Since these are called church-Turing thesis.

These predictions that it is suitable to consider the proposed and yet to be proposed models that all the models of computation that are proposed and yet to be proposed are equivalent in their power to recognize languages of complex functions. This

This thereby maintains that all the models of computations that are proposed and yet to be proposed are equivalent in their power to recognize languages of complex functions. This

the boundary of L and R, and focusing the end/infinite word to be written

$$S(a, b) = (a, b) S$$

$$(a, b) S = S(a, b)$$

the more transistions of the form

$$S \leftarrow \text{cured } g \times L + g \times T \times (L, P_3 \text{ each})$$

L  $\leftarrow$  set of type symbols

the special symbols E and R  
E  $\leftarrow$  set of input alphabet which also has  
G  $\leftarrow$  set of finite states  
L  $\leftarrow$  set of type symbols

$$M = (Q, Z, T, S, \delta, B, F)$$

The LBA is a TM

### Recursive Linear bounded Automaton (LBA)

This is superset of all TMs  
clearly from all these types of machines  
constructed PDA and finite automaton. If it is  
the usage of tape, we can observe LBA, we  
say. Thus, using TM and by extending  
it is bounded based on the length of the input  
Automaton (LBA). So, LBA is a TM with  
class of machine called Linear Bounded

longer than the wordspace. This leads to another

these two features. So, longer the string,

the given string has to be enclosed between

the tape using too determinates L and R.

of the tape, let us extract the wordspace in

Automaton, with slight alteration in usage

for some of E and x, e.g.

$$[x \cdot b \times I] + [I \cdot f \cdot a]$$

It is clear from the definition that the Blö head cannot go out of the boundary as specified as I. and L. now, the string can be converted by LBA only if there is a boundary in place.

Now both end-markers are both head should not point any other symbol  
any other cell within the input shape  
both the end-markers should not appear in head between getting to the right-end of the tape  
cell of which is ended in the leftmost marker which is ended in the rightmost  
The symbol I, is called the right end

head from getting off the left-end of the tape  
cell of the input shape passing to the Blö  
marker which is ended in the leftmost  
The symbol I, is called the left end

In contains two special symbols, I, and L

Observe the following pattern:

F  $\leftarrow$  Final states

B  $\leftarrow$  Blank characters

AO  $\leftarrow$  Start state

$$f(n) = O(n^k) \text{ and } f(n) \in O(c n^k)$$

$$f(n) = O(g(n)) \text{ or } f(n) \in O(g(n))$$

So, by definition, we can write:

$$f(n) \leq c * g(n) \text{ for } n > n_0 \text{ where } g(n) = n^k \text{ and } n_0 = 1.$$

$$\text{for } n > 1$$

$$\text{i.e., } f(n) \leq c * n^k \text{ where } c = (|a_k| + |a_{k-1}| + \dots + |a_1| + |a_0|)$$

$$\leq c * n^k \text{ where } c = (|a_k| + |a_{k-1}| + \dots + |a_1| + |a_0|)$$

$$\leq n^k (|a_k| + |a_{k-1}| + \dots + |a_1| + |a_0|)$$

$$n^k \left[ |a_k| + \frac{|a_{k-1}|}{n} + \dots + \frac{|a_1|}{n^{k-1}} + \frac{|a_0|}{n^k} \right]$$

$$\leq |a_k| n^k + |a_{k-1}| n^{k-1} + \dots + |a_1| n + |a_0|$$

$$|f(n)| = |a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0|$$

Each term in the summation is of the form  $a_i n^i$ . Since  $n$  is non-negative, a positive term will be negative only if  $a_i < 0$ . So,

it is given that  $f(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$

### SOLUTION:

Then  $f(n) = O(n^k)$ .

**THEOREM:** If  $f(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$  with  $a_k > 0$