Date
Page

Theory of computations:

Unit: 1 Introduction

Unit: 2 Finite Automates and Regular language

unit: 3 context free language and Push down Automation.

unit: 4: Turing Machines

unit 5: undecidability computability theory

unit 6: computational complexity.

complexity theory.

Automata theo

Unit-1

Introduction:

Date
Page

Alphabets:

↳ An alphabet is a finite, non-empty set whose elements called symbols.

↳ It is denoted by Σ.

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

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

String:

↳ a string over an alphabet Σ is a finite sequence of symbols where each symbol is an element of Σ

↳ It is denoted by ω.

↳ The length of string ω, denoted by $|\omega|$, is the no. of symbols contained in ω.

↳ The empty string denoted by e or ϵ is string having length zero.

eg. let $\Sigma = \{0, 1\}$

Then, $\epsilon, 0, 1, 01, 10, 101, 0010, \ldots$ are strings over Σ having length $0, 1, 1, 2, 3, 4$ respectively.

power of Alphabet: (Same length of all elements)

↳ The set of all strings of certain length k from an alphabet is the k^{th} power of alphabet.

Date
Page

i.e. $\Sigma^k = \{ w : |w| = k \}$

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

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

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

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

⋮

Kleene closure:

↳ The set of all strings over an alphabet Σ is called Kleene closure of Σ.

↳ It is denoted by Σ^*.

↳ $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$

↑ includes empty string

Positive closure:

↳ The set of all strings over an alphabet Σ exlept empty string is called positive closure of Σ.

↳ It is denoted by Σ^+.

↳ $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$

↑ doesnot includes empty string

Date
Page

concatenation of string:

↳ Two strings over the same alphabet can be combined to form third by operation of concatenation.

↳ concatenation of two string x and y denoted by $x \cdot y$ or xy is the string x followed by string y.

↳ Formally,

$w = xy$ if and only if

$|w| = |x| + |y|$

$w[i] = x[i]$ for $i = 1 \ldots |x|$

$w[|x| + j] = y[j]$ for $j = 1 \ldots |y|$

eg:

if $x = 0011$ and $y = 0101$

then,

$w = x \cdot y =$ ~~0011~~ ~~0101~~ 00110101

language:

↳ any set of strings over an alphabet Σ given some conditions is called language.

↳ It is denoted by L.

↳ $L \subseteq \Sigma^*$

eg:

$L = \{ w \in \{a, b\}^* : w$ has odd no. of a $\}$

certain condition (assumed)

Date
Page

Regular Expression:

↳ Is a formula to represent a language with help of symbols, three different operations union (∪ or +), concatenation (·) and kleene closure (*) and with parenthesis if needed.

↳ eg. find Regular expression of following.

1) $L = \{ w \in \{0,1\}^* : w \text{ ends with } 00 \}$

Here,

$L = \{00, 000, 0000, \ldots, 100, 1100, \ldots$
$0100, 1000, 10100, 001000, \ldots\}$

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

2) $L = \{ w \in \{0,1\}^* : w \text{ starts with } 0 \text{ and } 1 \text{ ends with } 1 \}$

Here,

$L = \{01, 001, 011, \ldots, 01101, 00001, \ldots\}$

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

3) $L = \{ w \in \{a,b\}^* : w \text{ have even no. of } a \text{ followed by odd no. of } b \}$

Here,

$L = \{aab, aabbb, aaaab, aaaabbb, \ldots\}$

$\therefore R.E = (aa)^* . b . (bb)^*$

Day-2

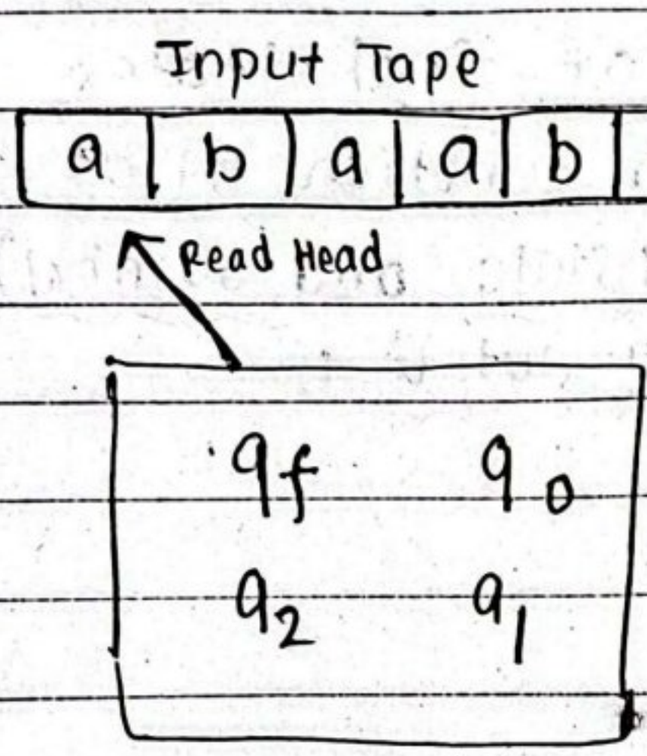
Date
Page

Chapter-2 Finite Automata and Regular language.

↳ Finite Automata is the model of computation that accepts/ rejects languages.

↳ It captures all possible states and transition that a machine can take while responding to a sequence of input symbols.

↳ It has finite sets of states, edge that lead from one state to another and edge labelled with a symbol.



[Fig: Block diagram of finite Automata]

↳ Here, Input tape is a device where string are kept.

↳ finite control is main part with finite no. of states.

↳ finite control can sense what symbol is written at any position on the input tape by means of movable reading head.

↳ Two types: (a) Deterministic finite Automata (DFA) (DF

(b) Non-deterministic finite Automata (

Date
Page

a) Deterministic finite Automata (DFA):

↳ It is a finite Automata that can exist in only one state at a given time.

↳ It is denoted by 5 tuples $(Q, \Sigma, \delta, q_0, f)$.

$Q \longrightarrow$ a finite set of states

$\Sigma \longrightarrow$ a finite set of symbols

~~$\delta \longrightarrow$ a start state a transition function~~

$q_0 \longrightarrow$ a start state

$f \longrightarrow$ a set of final states

$\delta \longrightarrow$ a transition function that takes input argument (a state and symbol) and returns a state as ~~inp~~ output.

Here,

$q_0 \in Q$

$F \subseteq Q$

$\delta \Rightarrow Q \times \Sigma \longrightarrow Q$

Where, $Q \times \Sigma$ is a set of 2 tuples

(q, a) with $q \in Q$

$a \in \Sigma$

↳ DFA is often represented by states or transition diagram.

↳ Transition function can also be represented by a table called transition table.

Date
Page

Example:

let the DFA be $D = (Q, \Sigma, \delta, q_0, F)$

where,

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

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

$q_0 = \{q_0\}$

$F = \{q_2\}$

Now,

$\delta(q_0, a) = q_1$

$\delta(q_0, b) = q_0$

$\delta(q_1, a) = q_1$

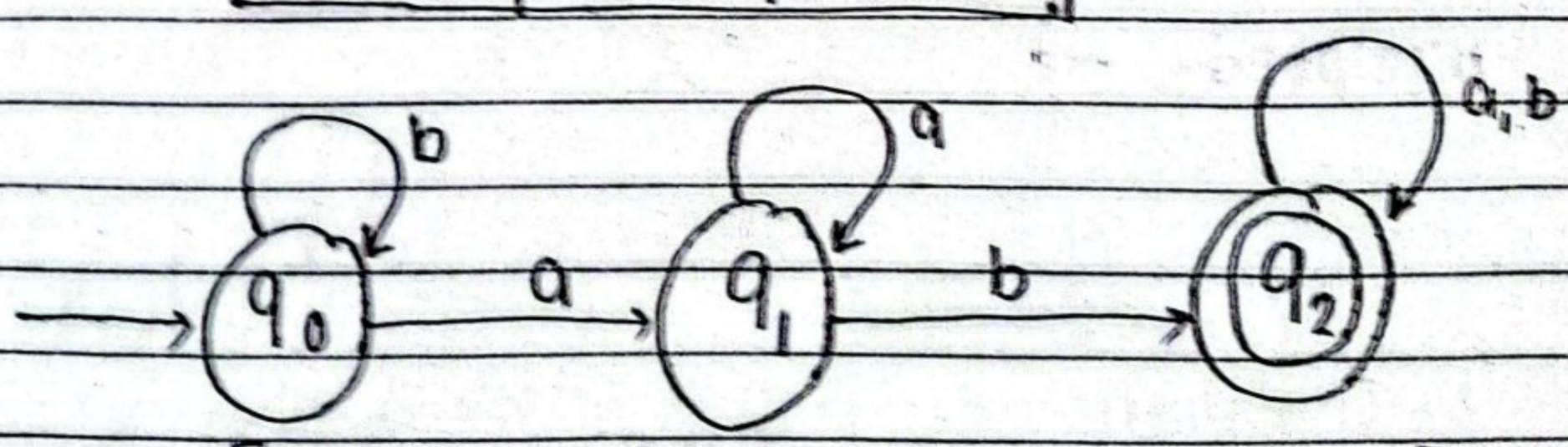
$\delta(q_1, b) = q_2$

$\delta(q_2, a) = q_2$

$\delta(q_2, b) = q_2$

Transition table:

Q/Σ	a	b
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_2	q_2



[Fig: State diagram or transition diagram]

final state: Double circle

Date
Page

Check the strings:

i) bbaa ii) abab ii) baab.

i) bbaa

$\delta(q_0, bbaa)$
$\vdash \delta(q_0, baa)$
$\vdash \delta(q_0, aa)$
$\vdash \delta(q_1, a)$
$\vdash \delta(q_1, \epsilon)$

As q_1 is not a final state, this string is rejected.

ii) abab

$\delta(q_0, abab)$

$\vdash \delta(q_1, bab)$
$\vdash \delta(q_2, ab)$
$\vdash \delta(q_2, b)$
$\vdash \delta(q_2, \epsilon)$

Accepted.

iii) baab

$$\delta(q_0, baab)$$

$$\vdash \delta(q_0, aab)$$
$$\vdash \delta(q_1, ab)$$
$$\vdash \delta(q_1, b)$$
$$\vdash \delta(q_2, \epsilon)$$

Accepted.

Hence it accepts those strings with ab strings.

$$L = \{ w \in \{a,b\}^* : w \text{ has } ab \text{ as substring} \}.$$

Day-3

Q) Design a DFA for a language:

$L = \{ w, w \in \{0,1\}^* : w \text{ starts with } 0 \text{ and ends with } 1 \}$

Solution:

$$L = \{01, 001, 011, 0001, 0111, 0101, 0011, \ldots\}$$

let the DFA be $D = (Q, \Sigma, \delta, q_0, F)$

Where,

$$Q = \{q_0, q_1, q_2, q_3\}$$
$$\Sigma = \{0,1\}$$
$$q_0 = \{q_0\}$$
$$F = \{q_2\}$$

Date
Page

Transition table: δ:

Q/Σ	0	1
q_0	q_1	q_3
q_1	q_1	q_2
q_2	q_1	q_2
q_3	q_3	q_3

Transition states:

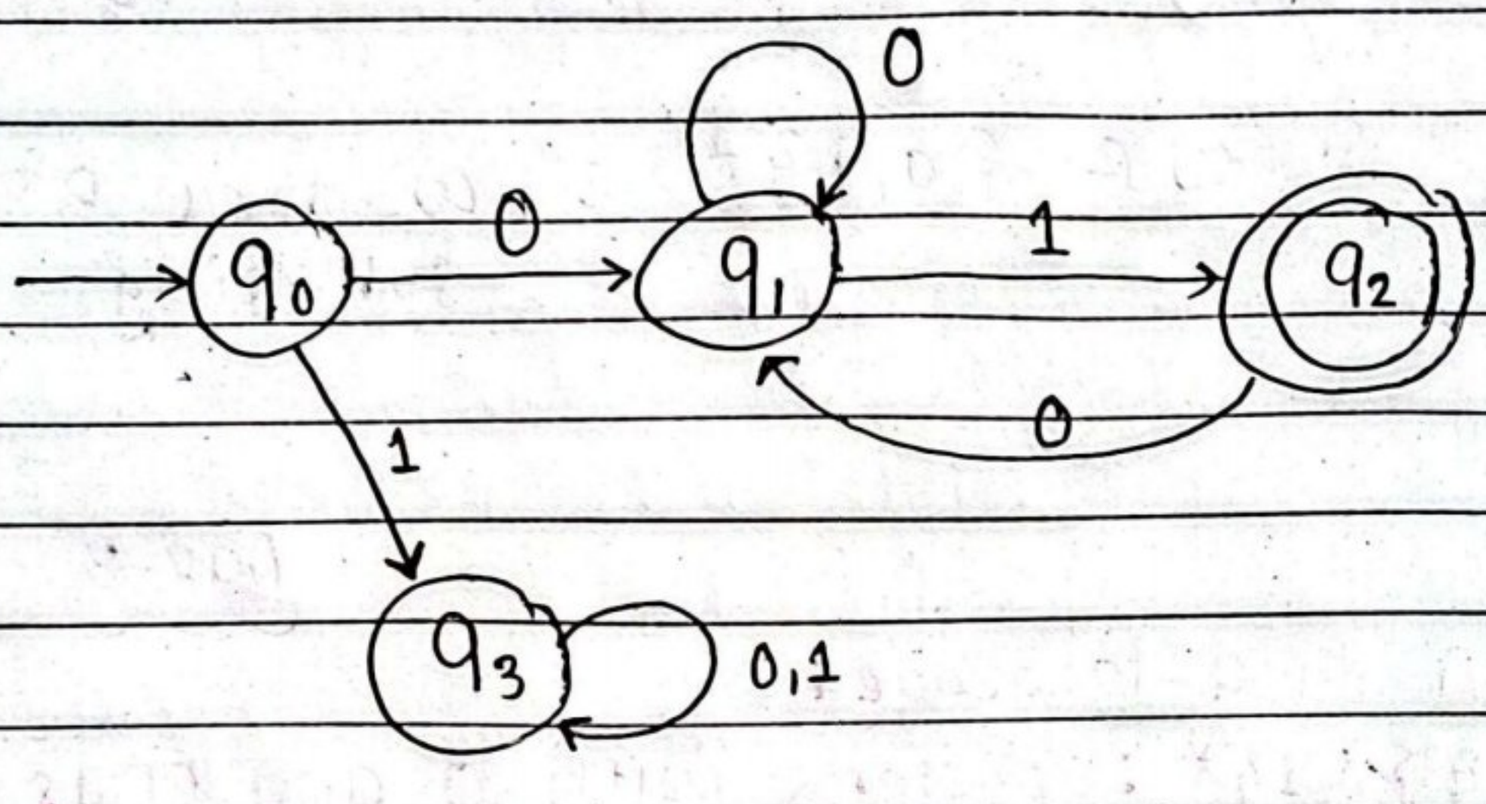


Fig: state diagram.

Testing strings:

(i) 01001

$\delta(q_0, 01001)$
$\vdash \delta(q_1, 1001)$
$\vdash \delta(q_2, 001)$
$\vdash \delta(q_1, 01)$
$\vdash \delta(q_1, 1)$
$\vdash \delta(q_2, \epsilon)$ (accepted)

(ii) 0100

$\delta(q_0, 0100)$
$\vdash \delta(q_1, 100)$
$\vdash \delta(q_2, 00)$
$\vdash \delta(q_1, 0)$
$\vdash \delta(q_1, \epsilon)$
(rejected)

Date
Page

(1) Design a DFA for language:

$L = \{ w \in \{0,1\}^* : w \text{ starts with } 01 \}$

Solution:

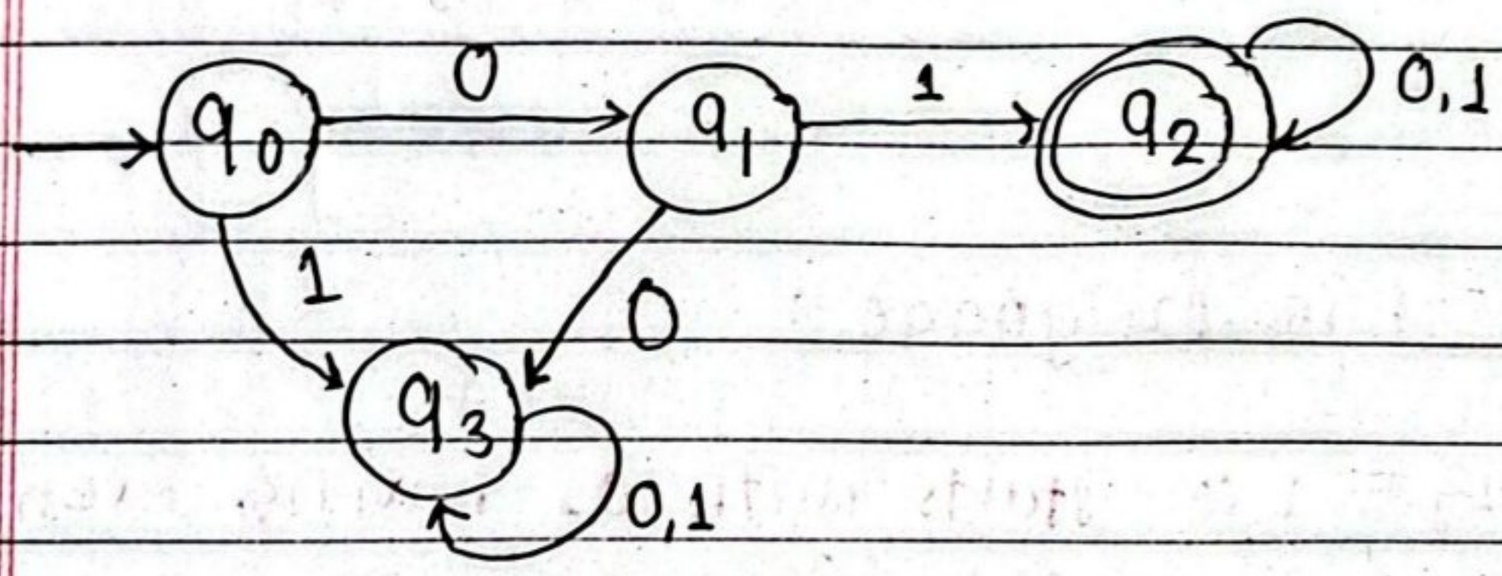
$L = \{01, 010, 011, 0100, 0101, 0111, 0110, ---\}$

Now,

let the DFA be $D = (Q, \Sigma, \delta, q_0, F)$

So,

let's roughly sketch the state diagram at first



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

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

$q_0 = \{q_0\}$

$F = \{q_2\}$

Q/Σ	0	1
q_0	q_1	q_3
q_1	q_3	q_2
q_2	q_2	q_3
q_3	q_3	q_3

Testing strings:

① 01001

$\delta(q_0, 01001)$
$\vdash \delta(q_1, 1001)$
$\vdash \delta(q_2, 001)$
$\vdash \delta(q_2, 01)$
$\vdash \delta(q_2, 1)$
$\vdash \delta(q_2, \epsilon)$

(accepted)

② 001

$\delta(q_0, 001)$
$\vdash \delta(q_1, 01)$
$\vdash \delta(q_3, 1)$
$\vdash \delta(q_3, \epsilon)$

(rejected)

Q) Design a DFA in language:

$L = \{ w \in \{0,1\}^* : w$ starts with 01 having even lengths $\}$.

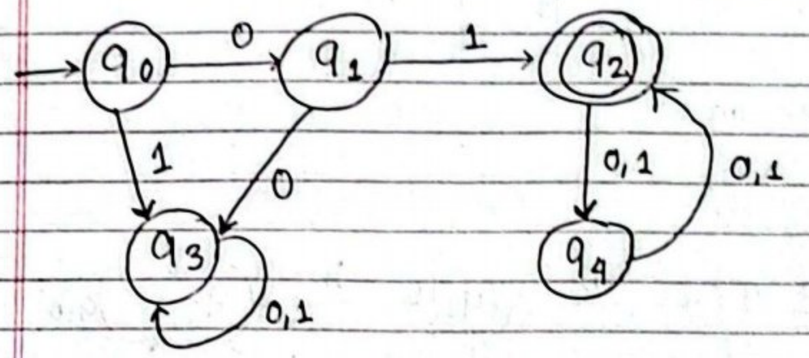
Solution:

let the DFA be $D = (Q, \Sigma, \delta, q_0, F)$

where,

$Q = \{q_0, q_1, q_2, q_3, q_4\}$
$\Sigma = \{0, 1\}$
$q_0 = \{q_0\}$
$F = \{q_2\}$
δ:

Q / Σ	0	1
$\rightarrow q_0$	q_1	q_3
q_1	q_3	q_2
q_2*	q_4	q_4
q_3	q_3	q_3
q_4	q_2	q_2



[Fig: state diagram]

Q) Design a DFA in language:

$L = \{ w \in \{a,b\}^* : w$ doesn't have three consecutive b's $\}$

Solution:

$L = \{ \epsilon, b, bb, a, aa, aaa, ---, ababb, babba ---\}$

let the DFA be $D = \{Q, \Sigma, \delta, q_0, F\}$

so,

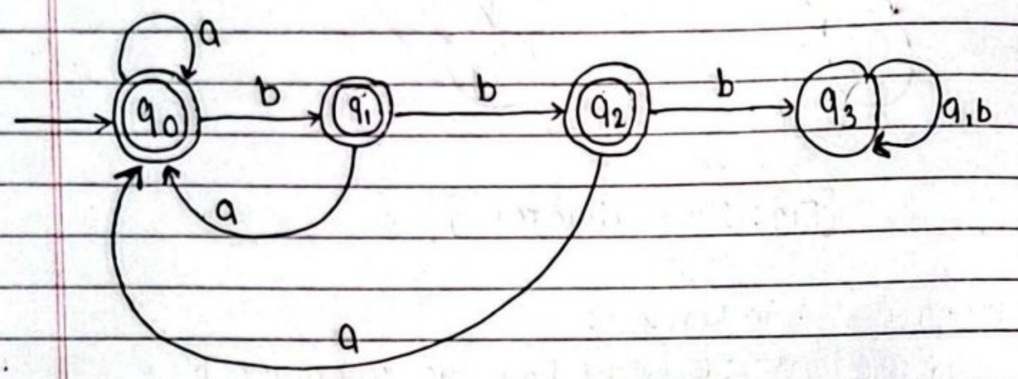
$Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{a, b\}$
$q_0 = \{q_0\}$
$f = \{a, b\}$

δ:

Transition table:

Q / Σ	a	b
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_0	q_3
q_3	q_3	q_3

Transition diagram:



[Fig: states of transition]

Non Deterministic finite Automata (NFA):

↳ It is a finite automata where several possible next states for a given combination of current state and input symbol could exist.

↳ It reads the input symbols and many choose at each step to go to any of legal status.

↳ here, choice is not determined by anything in model, so non-deterministic.

↳ Formally, NFA is defined by 5 tuples $(Q, \Sigma, \delta, q_0, F)$.

where,

$Q \rightarrow$ a finite set of states.
$\Sigma \rightarrow$ a finite set of symbols
$q_0 \rightarrow$ a start state
$F \rightarrow$ a set of final states.
$\delta \rightarrow$ transition function
where,
$\delta \Rightarrow Q \times \Sigma$ = subset of Q.

eg.

let, NFA be $N = (Q, \Sigma, \delta, q_0, F)$

where,

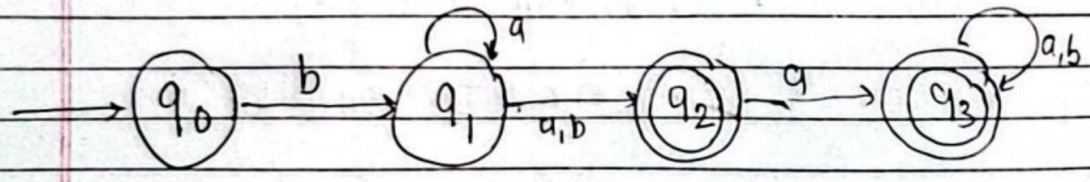
$Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{a, b\}$
$q_0 = q_0$

$F = \{q_2, q_3\}$

δ:

Q/Σ	a	b
q_0	—	q_1
q_1	$\{q_1, q_2\}$	q_2
q_2	q_3	—
q_3	q_3	q_3

1) $\delta(q_0, b) = q_1$
2) $\delta(q_1, a) = \{q_1, q_2\}$
3) $\delta(q_1, b) = q_2$
4) $\delta(q_2, a) = q_3$
5) $\delta(q_2, b) = —$
6) $\delta(q_3, a) = q_3$
7) $\delta(q_3, b) = q_3$



[Fig: state - diagram]

Note: following different paths, if any of the path is accepted then the string is accepted.

Test for strings

1) bbb

$\delta(q_0, bbb)$
$\vdash \delta(q_1, bb)$
$\vdash (q_2, b)$
~~$\vdash \delta$~~.

{ reject as b is left. }

2) baa

$\delta(q_0, baa)$ | $\delta(q_0, baa)$
$\vdash (q_1, aa)$ | $\vdash \delta(q_1, aa)$
$\vdash (q_1, a)$ | $\vdash \delta(q_2, a)$
$\vdash (q_1, \epsilon)$ | $\vdash \delta(q_3, \epsilon)$
rejected | accepted.

{ so accepted }

Q) Design a NFA for language:

$L = \{w \in \{a,b\}^* : w$ has pattern bb or bab

Solution:

$L = \{bb, abb, bbb, abba, \cdots, bab, abab, \cdots\}$

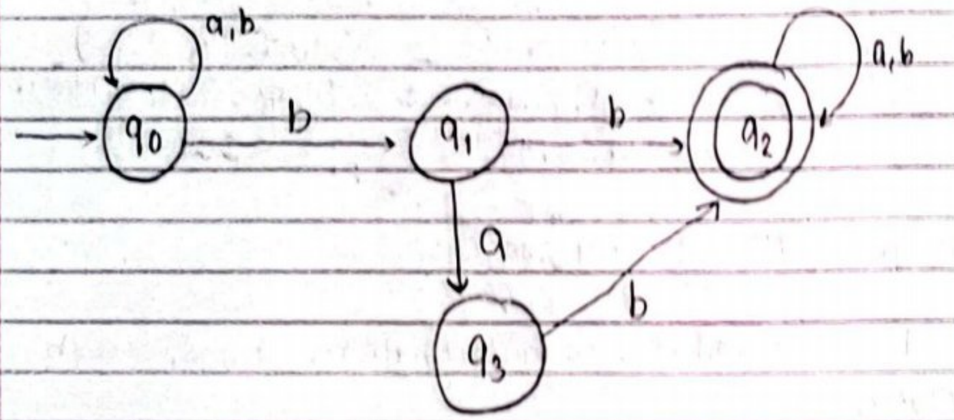
let the NFA be $N = \{Q, \Sigma, \delta, q_0, F\}$

so,

$Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{a, b\}$
$q_0 = \{q_0\}$
$F = \{q_2\}$

δ:

Q/Σ	a	b
q_0	q_0	$\{q_0, q_1\}$
q_1	q_3	q_2
q_2	q_2	q_2
q_3	—	q_3



[Fig: state diagram]

(i) ababbq

$\delta(q_0, ababba)$
$\vdash \delta(q_0, babba)$
$\vdash \delta(q_1, abba)$
$\vdash \delta(q_3, bba)$
$\vdash \delta(q_2, ba)$
$\vdash \delta(q_2, a)$
$\vdash \delta(q_2, \epsilon)$ accepted

(ii) aabaab

$\vdash \delta(q_0, aabaab)$
$\vdash \delta(q_0, abaab)$
$\vdash \delta(q_0, baab)$
$\vdash \delta(q_1, aab)$
$\vdash \delta((q_3, ab)$
rejected

$\vdash \delta(q_0, aabaab)$
$\vdash \delta(q_0, abaab)$
$\vdash \delta(q_0, baab)$
$\vdash \delta(q_0, aab)$
$\vdash \delta(q_0, ab)$
$\vdash \delta(q_0, b)$
$\vdash \delta(q_0, \epsilon)$ reject

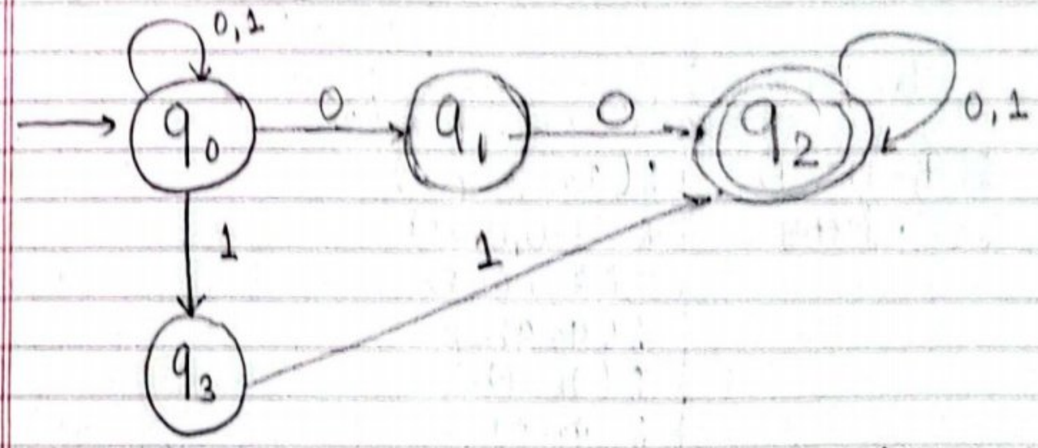
(Q) Design a NFA for language.

$L = \{w \in \{0,1\}^* : w$ has 00 or 11 as substring$\}$

And also check for: (i) 0101
(ii) 011001

solution:

$L = \{00, 100, 001, 1001, 000, \ldots$
$11, 011, 110, 0110, 111, \ldots\}$



let the NFA be $N = \{Q, \Sigma, \delta, q_0, F\}$

so,

$Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{0, 1\}$
$q_0 = q_0$
$F = q_2$

δ:

Q/Σ	0	1
q_0	q_0, q_1	q_3
q_1	q_2	—
q_2	q_2	q_3
q_3	—	q_2

i) $\delta(q_0, 0101)$

$\vdash \delta(q_1, 101)$	$\vdash \delta(q_0, 101)$	$\vdash \delta(q_0, 101)$
~~$\vdash$~~ rejected	$\vdash \delta(q_3, 01)$	$\vdash \delta(q_0, 01)$
	rejected	$\vdash \delta(q_0, 1)$
		$\vdash \delta(q_3, \epsilon)$
		rejected

So it is rejected.

ii) $\delta(q_0, 011001)$

$\delta(q_1, 11001)$	$\delta(q_0, 11001)$
~~δ~~ rejected	$\delta(q_3, 1001)$
	$\delta(q_2, 001)$
	$\delta(q_2, 01)$
	$\delta(q_2, 1)$
	$\delta(q_2, \epsilon)$
	accepted.

So accepted.

Q) Design a NFA for $L = \{w \in \{a,b\}^* : w$ has aa as substring$\}$

Solution:

$L = \{aa, baa, aab, aaa, baab, ---\}$

let the NFA be, $N = \{Q, \Sigma, \delta, q_0, F\}$

So,

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

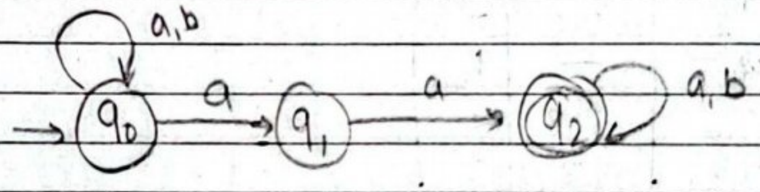
$\Sigma = \{a, b\}$

$q_0 = q_0$

$F = \{q_2\}$

δ:

Q/Σ	a	b
q_0	$\{q_0, q_1\}$	q_0
q_1	q_2	—
q_2	q_2	q_2

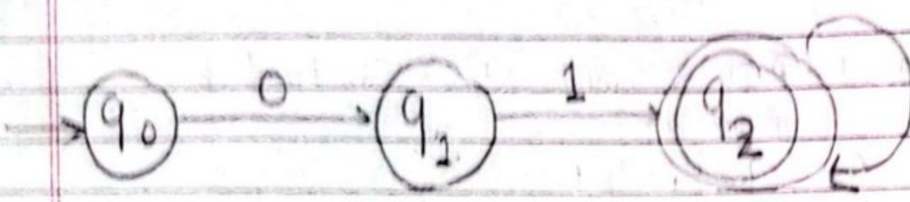


strings equally and DFA & NFA.

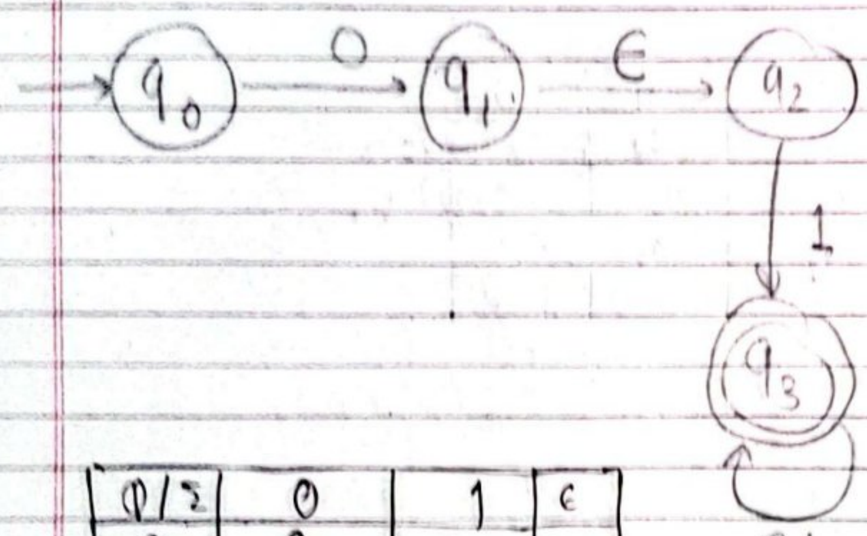
② Design a NFA for

$L = \{ w \in \{0,1\}^* : w \text{ starts with } 01 \}$

$L = \{01, 010, 011, 0100, 0101, 0110, 0111, \ldots\}$



Or



Q/Σ	0	1	ϵ
q_0	q_1	—	-
q_1	q_1	—	q_2
q_2	-	q_3	-
q_3	q_3	q_3	-

Assignment 1:

Equivalence of NFA and DFA:

↳ For each NFA, there is an equivalent DFA.

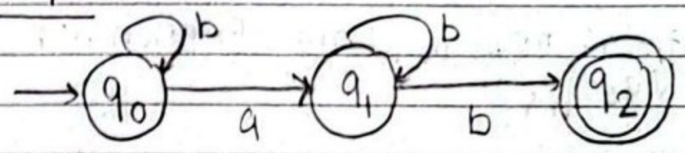
↳ conversion steps:

i) Observe all the transition from starting state q_0, for every symbol in Σ.

ii) For new states from (i), check ~~out~~ all transitions for every Σ.

iii) Repeat step ii) till new states are appeared.

Example:



Solution:

Given NFA be $N = (Q, \Sigma, \delta, q_0, F)$.

$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b\}$
$q_0 = \{q_0\}$
$F = \{q_2\}$

δ:

1) $\delta(q_0, a) = q_1$
2) $\delta(q_0, b) = q_0$
3) $\delta(q_1, b) = \{q_2, q_1\}$

final state chai jaha jaha q_2 za tyo sabai

let, equivalent DFA be $D = \{Q', \Sigma', \delta', q_0', f'\}$

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

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

$q_0' = \{q_0\}$ Assume

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

δ':

Q'/Σ'	a	b
$\{q_0\}$	$\{q_1\}$	$\{q_0\}$
$\{q_1\}$	ϕ	$\{q_2\}$
$\{q_1, q_2\}$	ϕ	$\{q_1, q_2\}$

considered as single state

let start state of DFA be start of given NFA.
i.e $\{q_0\}$ N.S

For $\{q_0\}$

$\delta'(\{q_0\}, a) = \delta(q_0, a) = \{q_1\}$ N.S

$\delta'(\{q_0\}, b) = \delta(q_0, b) = \{q_0\}$

For $\{q_1\}$

$\delta'(\{q_1\}, a) = \phi$

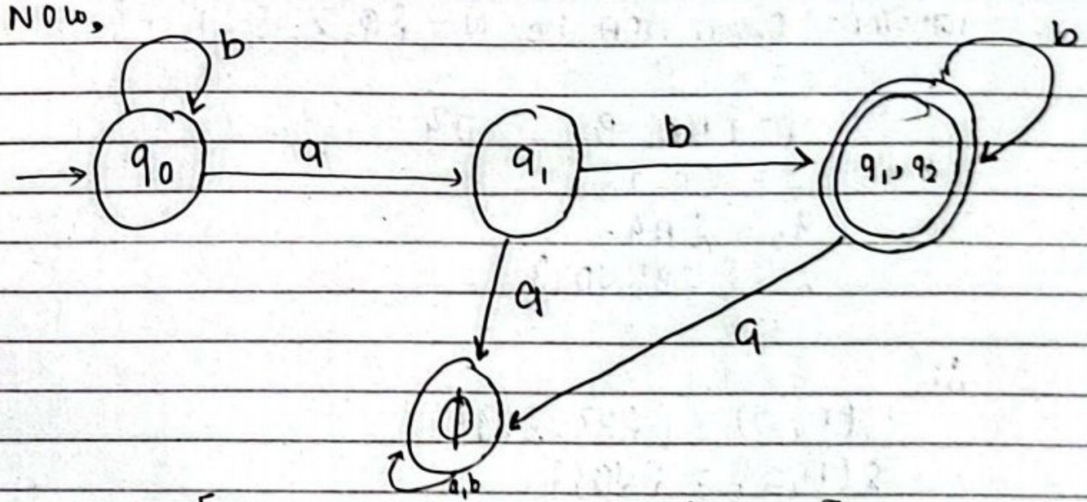
$\delta'(\{q_1\}, b) = \{q_1, q_2\}$ N.S

For $\{q_1, q_2\}$

$\delta'(\{q_1, q_2\}, a) = \delta(q_1, a) \cup \delta(q_2, a) = \phi \cup \phi = \phi$

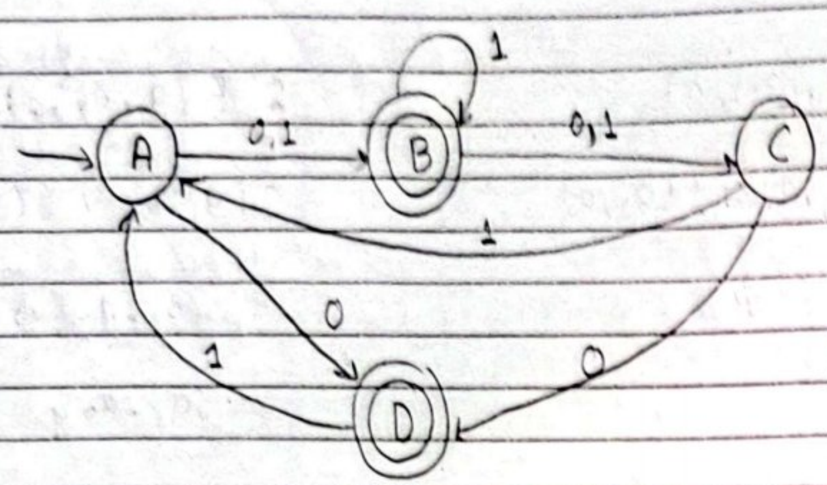
$\delta'\{\{q_1, q_2\}, b) = \delta(q_1, b) \cup \delta(q_2, b) = \{q_1, q_2\} \cup \phi = \{q_1, q_2\}$

Now,



[Fig: State diagram of DFA].

Q) Convert following NFA into DFA:



Solution: Given NFA be $N = \{Q, \Sigma, \delta, q_0, F\}$

$Q = \{A, B, C, D\}$
$\Sigma = \{0, 1\}$
$q_0 = \{A\}$
$F = \{\{B\}, \{D\}\}$

δ:

$\delta(A, 0) = \{\{B\}, \{D\}\}$
$\delta(A, 1) = \{\{B\}\}$
$\delta(B, 0) = \{C\}$
$\delta(B, 1) = \{\{B\}, \{C\}\}$
$\delta(C, 0) = \{D\}$
$\delta\{C, 1\} = \{A\}$
~~$\delta\{D, 0\} = \{$~~
$\delta(D, 1) = \{A\}$

let equivalent DFA be $D = \{Q', \Sigma', \delta', q_0', F'\}$

$Q' = \{\{A\}, \{B\}, \{C\}, \{A, B, C\}, \{B, C\}, \{C, D\}, \{D\}\}$
$\Sigma' = \{0, 1\}$ $\{B, D\}, \{B, C, D\}\}$
$q_0' = \{A\}$ N.S
$F' = \{\{B\}, \{A, B, C\}, \{B, C\}, \{C, D\}, \{D\}, \{B, C, D\}\}$

δ:

- $\delta'(\{A\}, 0) = \{B, D\}$ N.S. $\quad \delta'(\{A\}, 1) = \{B\}$ N.S

- $\delta'\{\{B, D\}, 0\}$
  $= \delta(B, 0) \cup \delta(D, 0)$
  $= C \cup \phi$
  $= \{C\}$ N.S
  $\quad \delta'(\{B, D\}, 1)$
  $= \delta(B, 1) \cup \delta(D, 1)$
  $= \{B, C\} \cup \{A\}$
  $= \{A, B, C\}$ N.S.

- $\delta'(B, 0\}$
  $= \{C\}$
  $\quad \delta'(B, 1)$
  $= \{B, C\}$ N.S.

- $\delta'(\{B, C\}, 0)$
  $= C \cup D$
  $= \{C, D\}$ N.S.
  $\quad \delta'(\{B, C\}, 1)$
  $= \{B, C\} \cup \{A\}$
  $= \{A, B, C\}$

- $\delta'\{\{C, D\}, 0\}$
  $= D \cup \phi$
  $= \{D\}$ N.S
  $\quad \delta'(\{C, D\}, 1)$
  $= A \cup A$
  $= \{A\}$

- $\delta'(C,0)$ = $\{D\}$

- $\delta'\{D,0\}$ = ϕ

- $\delta'\{D,1\}$ = $\{A\}$

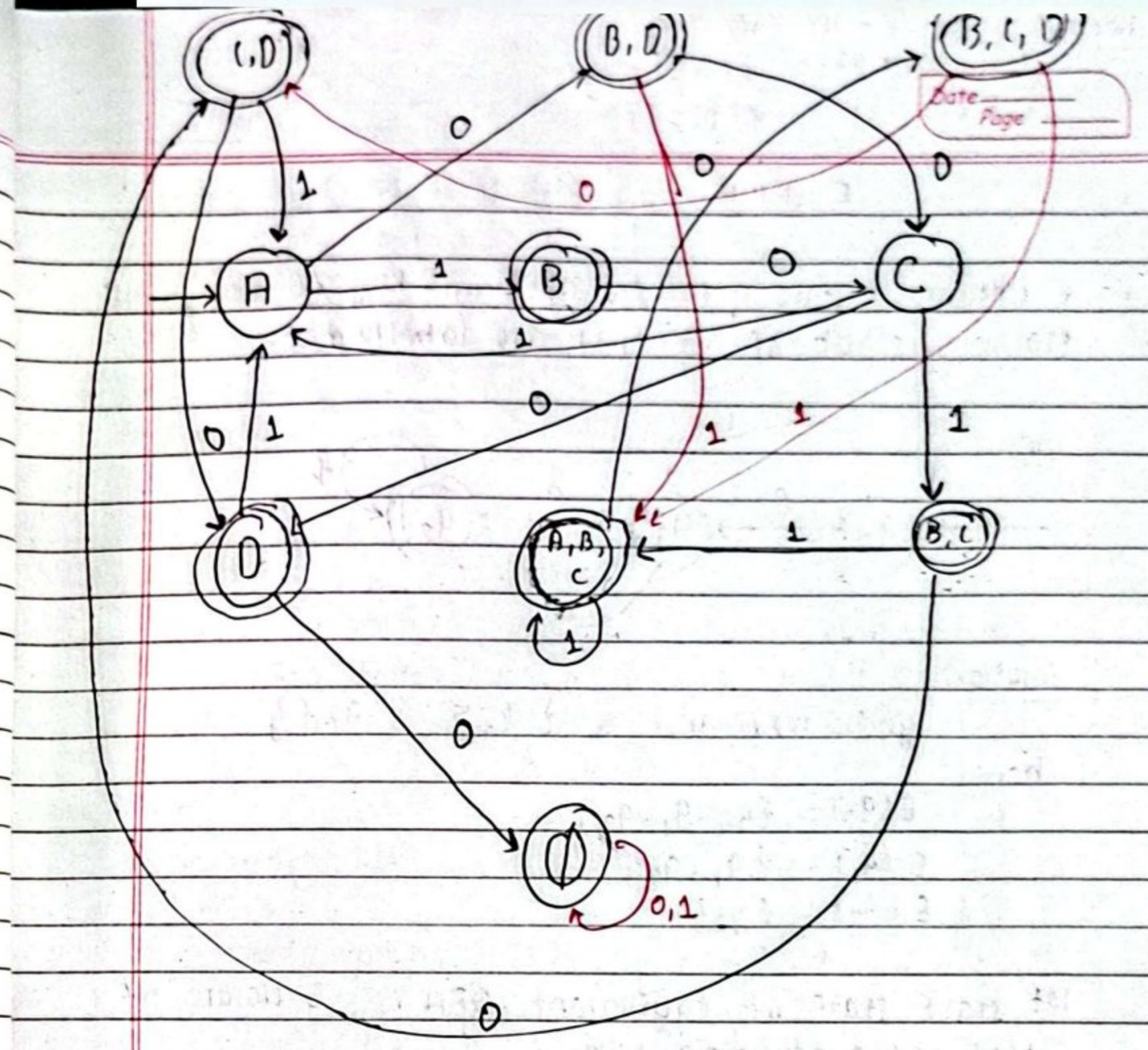
- $\delta'(\{A,B,C\},0)$ = $\{B,D\} \cup \{C\} \cup \{D\}$ = $\{B,C,D\}$ N.S.

- $\delta'(\{A,B,C\},1)$ = $\{B\} \cup \{B,C\} \cup \{A\}$ = $\{A,B,C\}$

- $\delta'(\{B,C,D\},0)$ = $\{C\} \cup \{D\}$, = $\{C,D\}$

- $\delta'(\{B,C,D\},1)$ = $\{B,C\} \cup \{A\} \cup \{A\}$ = $\{A,B,C\}$

δ': Q'/Σ'	0	1
$\{A\}$	$\{B,D\}$	$\{B\}$
$\{B\}$	$\{C\}$	$\{B,C\}$
$\{C\}$	$\{D\}$	$\{A\}$
$\{D\}$	ϕ	$\{A\}$
$\{A,B,C\}$	$\{B,C,D\}$	$\{A,B,C\}$
$\{B,C\}$	$\{C,D\}$	$\{A,B,C\}$
$\{C,D\}$	$\{D\}$	$\{A\}$
$\{B,D\}$	$\{C\}$	$\{A,B,C\}$
$\{B,C,D\}$	$\{C,D\}$	$\{A,B,C\}$



Converting ϵ-NFA to DFA:

Steps:

(i) Find out the ϵ-closure of starting state and calculate union of ϵ-closure of each transition for every symbol of Σ.

(ii) Repeat same process till new states are appeared.

Note: → (x) → (y) → (z) → (p)

E(x) = {x, y} E(z) = {z, p}

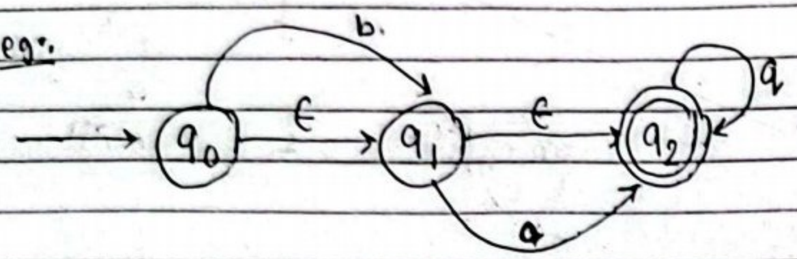
E(y) = {y} E(p) = {p}

Date Page

E-closure:

↳ ϵ-closure of state q can be obtained by following all transitions out of q that are labelled ϵ.

eg:



Solution:

given NFA be $N = \{Q, \Sigma, \delta, q_0, F\}$

here,

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

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

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

let, start state of equivalent DFA is ϵ-closure of start state of NFA. i.e.

$\{q_0, q_1, q_2\}$ N.S.

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

$\delta'(\{q_0,q_1,q_2\}, a)$

$= \delta(q_0,a) \cup \delta(q_1,a) \cup \delta(q_2,a)$

$\phi \cup \epsilon(q_2) \cup \epsilon(q_2)$

$\{q_2\}$ N.S.

$\delta'(\{q_0,q_1,q_2\}, b)$

$\epsilon\{q_1\} \cup \phi \cup \phi$

$\{q_1, q_2\}$ N.S.

For q_2

$\delta'(q_2, a)$

$\cdot \epsilon\{q_2\}$

$\{q_2\}$

$\delta'(q_2, b)$

$\cdot \phi$

For $\{q_1, q_2\}$

$\delta'(\{q_1,q_2\}, a)$

$= \epsilon\{q_2\} \cup \epsilon\{q_2\}$

$= \{q_2\}$

$\delta'(\{q_1,q_2\}, b)$

$= \phi \cup \phi$

$= \phi$

No more new states, so process terminates.

Now,

Equivalent DFA $D = \{Q', \Sigma', \delta', q_0', F'\}$

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

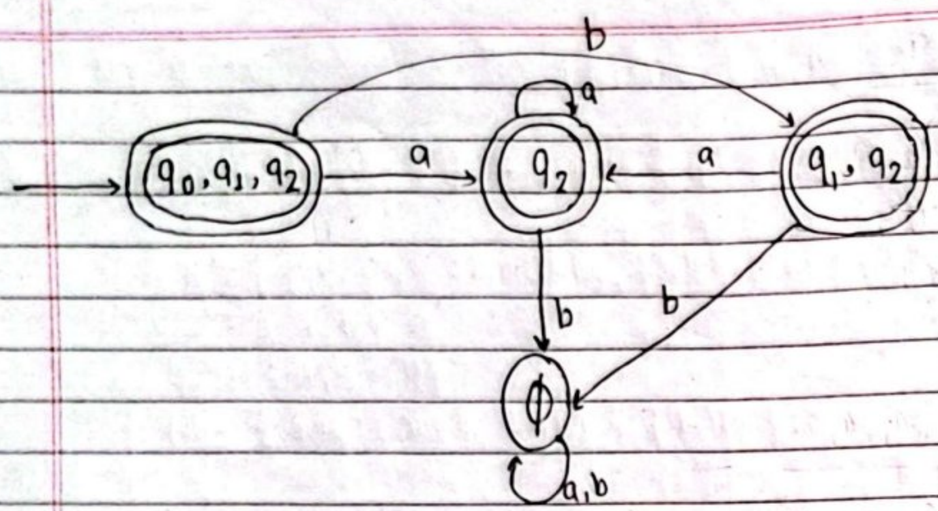
$\Sigma' = \{a, b\}$

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

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

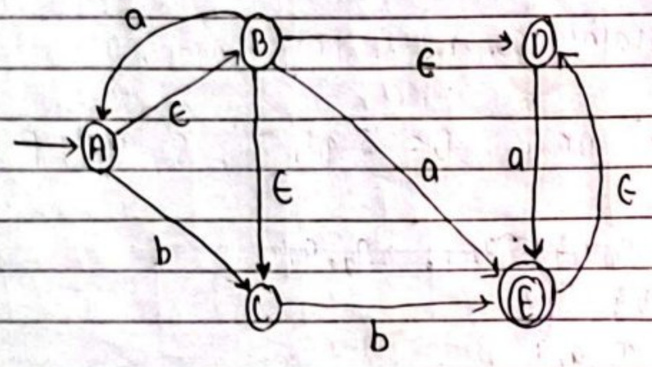
δ':

Q'/Σ'	a	b
$\{q_0,q_1,q_2\}$	$\{q_2\}$	$\{q_1,q_2\}$
$\{q_2\}$	$\{q_2\}$	ϕ
$\{q_1,q_2\}$	$\{q_2\}$	ϕ



[Fig: states diagram]

Q) convert following NFA to DFA.



Solution:

given NFA $N = \{Q, \Sigma, \delta, q_0, F\}$

Here,

$\epsilon(A) = \{\{A, B, C, D\}$

$\epsilon(B) = \{B, C, D\}$

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

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

$\epsilon(E) = \{E, D\}$

let the start of equivalent DFA is e-closure of start state of NFA. i.e.

$\{A, B, C, D\}_{N.S.}$

For $\{A, B, C, D\}$

$\delta'(\{A, B, C, D\}, a)$
$= \phi \cup \epsilon(E) \cup \epsilon(A) \cup \phi \cup \epsilon(E)$
$= \{E, D\} \cup \{A, B, C, D\}$
$= \{A, B, C, D, E\}_{N.S.}$

$\delta'(\{A, B, C, D\}, b)$
$= \epsilon(C) \cup \phi \cup \epsilon(E) \cup \phi$
$= \{C\} \cup \{E, D\}$
$= \{C, D, E\}_{N.S.}$

For $\{A, B, C, D, E\}$

$\delta'(\{A, B, C, D, E\}, a)$
$\phi \cup [\epsilon(A) \cup \epsilon(E)] \cup \phi \cup \epsilon(E) \cup \phi$
$= \{A, B, C, D, E\}$

$\delta'(\{A, B, C, D, E), b)$
$= \epsilon(C) \cup \phi \cup \epsilon(E) \cup \phi \cup \phi$
$= \{C, D, E)$

For $\{C, D, E\}$

$\delta'(\{C, D, E\}, a)$
$\phi \cup \epsilon(E) \cup \phi$
$\{E, D\}_{N.S.}$

$\delta'(\{C, D, E\}, b)$
$= \epsilon(E) \cup \phi \cup \phi$
$= \{C, D)$

For $\{E, D\}$

$\delta'\{(E, D), a)$
$= \phi \cup \epsilon(E)$
$= \{E, D\}$

$\delta'(\{E, D\}, b)$
$= \phi \cup \phi$
$= \phi$

No more new states, so process terminates

Now,

Equivalent DFA $D = \{Q', \Sigma', \delta', q_0', F'\}$

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

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

$q_0' = \{A,B,C,D\}$

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

δ:

Q'/Σ'	a	b
$\{A,B,C,D\}$	$\{A,B,C,D,E\}$	$\{C,D,E\}$
$\{A,B,C,D,E\}$	$\{A,B,C,D,E\}$	$\{C,D,E\}$
$\{C,D,E\}$	$\{E,D\}$	$\{E,D\}$
$\{E,D\}$	$\{E,D\}$	$\emptyset$

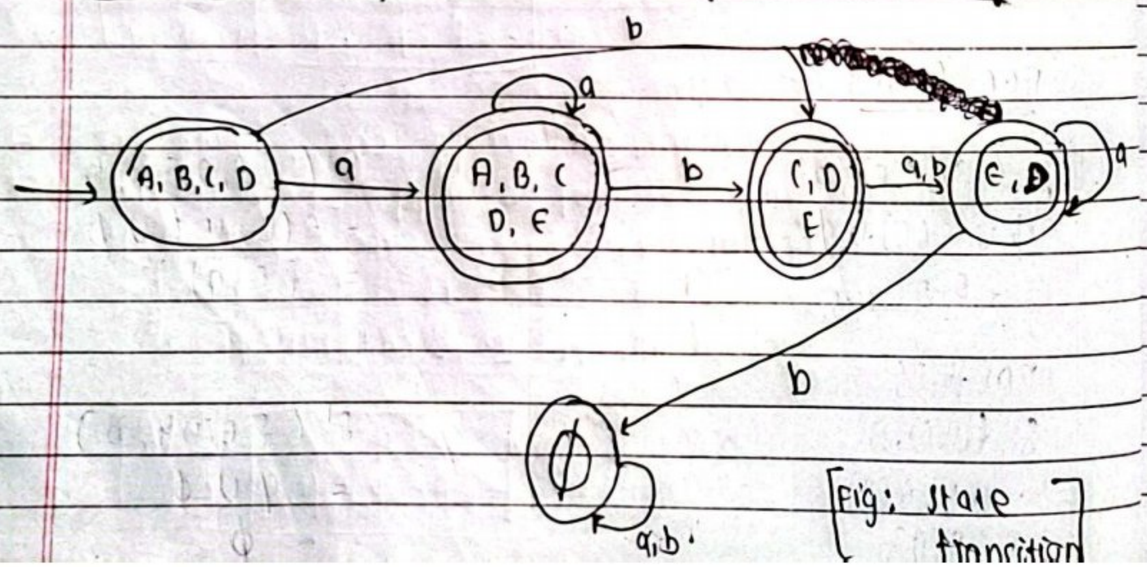


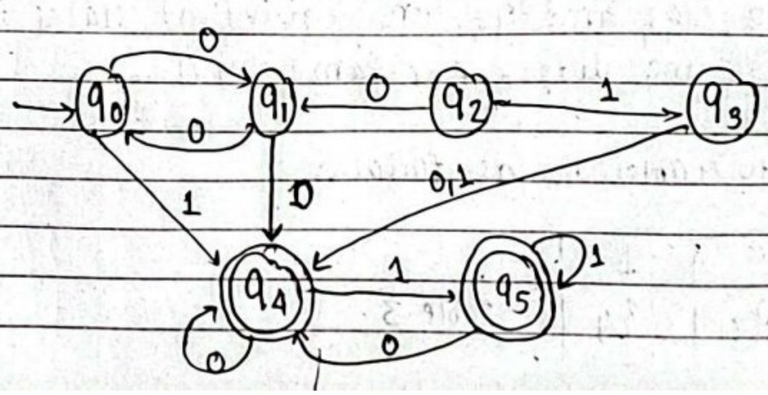
Fig: state transition

State minimization:

- It is the process of minimizing the given DFA by removing unreachable states and by merging similar equivalent states.

Steps:

1. Remove unreachable states from given DFA.
2. construct a transition table for rest of states.
3. Divide transition table of step 2 into two subtables as:
   - (a) rows that start with final states
   - (b) rows that start with non final states.
4. Eliminates equivalent state from tables 3(a) & 3(b) seperately.
5. Repeat step 4 until equivalent states are appeared.
6. Combine / merge updated table.
7. Draw minimized diagram.



↳ Here q_2 and q_3 are unreachable states so remove them.

↳ construct transition table of remaining states.

Q/Σ	0	1
q_0	q_1	q_4
q_1	q_4	q_0
q_4	q_4	q_5
q_5	q_4	q_5

↳ Divide above table as following tables.

Q/Σ	0	1
q_0	q_1	q_4
q_1	q_4	q_0

Table 1: Non-final state

Q/Σ	0	1
q_4	q_4	q_5
q_5	q_4	q_5

Table 2: final state.

↳ In table 1, no equivalent state.

↳ In table 2, q_4 and q_5 are equivalent states because they have same output for same input.

So, update table 2 as follow:

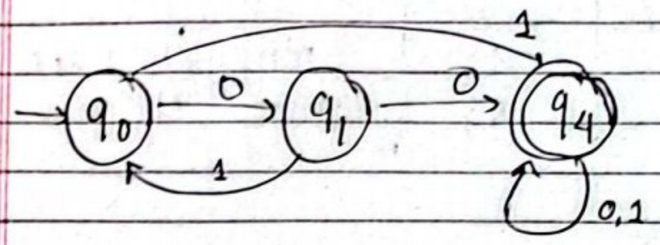
Q/Σ	0	1
q_4	q_4	q_4

table 3.

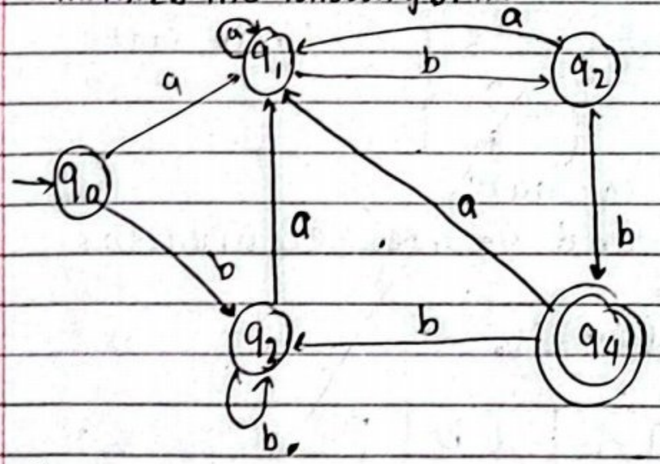
→ Now, combine updated table Table 3 and Table 1 as follow.

Q/Σ	0	1
q_0	q_1	q_4
q_1	q_4	q_0
q_4	q_4	q_4

Finally, minimized diagram is as:



Q) Minimize the following DFA.



Solution:

Here

Q/Σ	a	b
q_0	q_1	q_3
q_1	q_1	q_2
q_2	q_1	q_4
q_3	q_1	q_3
q_4	q_1	q_3

Here,

Divide above tabe as.

Q/Σ	a	b
q_1	q_1	q_2
q_2	q_1	q_4

non ~~finite~~ state (fina)

Q/Σ	a	b
q_0	q_1	q_3
q_3	q_1	q_3
q_4	q_1	q_3

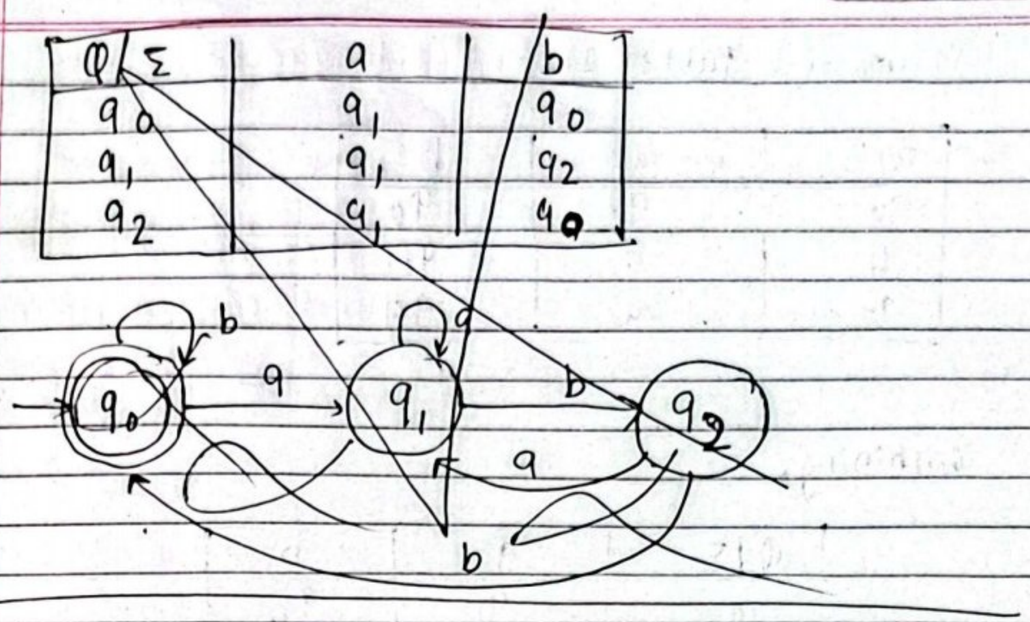
finale state

- In table 1, no eq. state.
- In table 2, q_0 and q_3 are equivalent.

So,

update table 2 is,

Q/Σ	a	b
q_0	q_1	q_0



Q/Σ	a	b
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_4
q_4	q_1	q_3

Divide above table as,

Q/Σ	a	b
q_0	q_1	q_3
q_1	q_1	q_2
q_2	q_1	q_4
q_3	q_1	q_3

Table 1: non-final

Q/Σ	a	b
q_4	q_1	q_3

Table 2: final

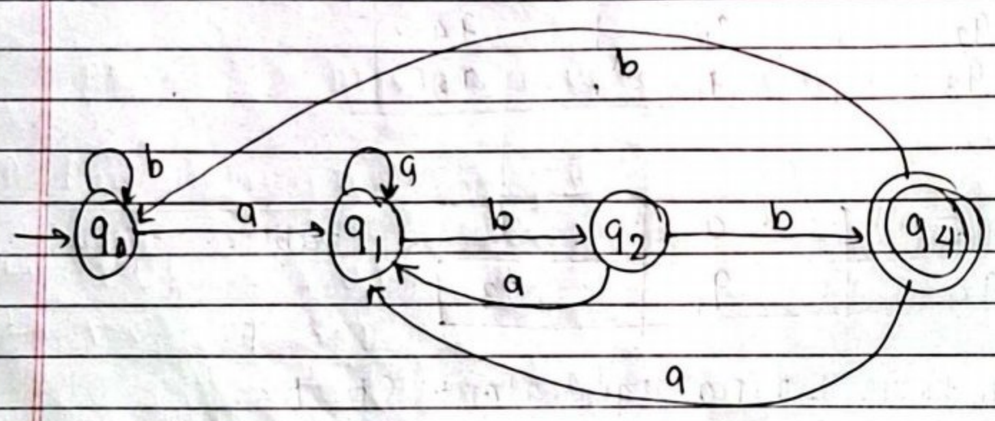
- In table 2: no equivalent state.
- In table 1, q_0 and q_3 are equivalent.

So updated table 1 is.

Q/Σ	a	b
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_4

combining.

Q/Σ	a	b
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_4
q_4	q_1	q_0



Regular Expression and languages.

↳ The language accepted by finite automata is called regular language

↳ The algebraic expression that represents regular ~~expression~~ language is called regular expression.

↳ let Σ be an alphabet, the class of regular language over Σ is defined as follows.

(i) ϕ is regular language representing empty language

(ii) $\{\in\}$ is regular language representing language of empty string.

(iii) For each $a \in \Sigma, \{a\}$ is a regular language.

(iv) If $L_1, L_2, \dots L_n$ are regular language then so is $L_1 \cup L_2 \cup \dots L_n$.

(v) If $L_1, L_2, \dots L_n$ are regular language then so is $L_1 \cdot L_2 \cdot \dots L_n$.

(vi) If L is a regular language then so is L^*.

Application of Regular language:

① **Validation:**

↳ Determining that a string complies with a set of formatting constraints like email validation, password validation, etc.

② **Search and selection:**

↳ Identifying a subset of items from ~~language~~ larger sets on a basis of platform match.

③ **Tokenization:**

↳ Converting a sequence of characters into words, tokens for later interpretation.

Alg. Algebric laws for Regular Expressions.

① **Commutativity:**

↳ $r \cup s = s \cup r$

↳ $r \cdot s \neq s \cdot r$

② **Associativity:**

↳ $r \cup (s \cup t) = (r \cup s) \cup t$

↳ $r \cdot (s \cdot t) = (r \cdot s) \cdot t$

③ **Distributive:**

↳ $r(s \cup t) = rs \cup rt$

↳ $(s \cup t)r = sr \cup tr$.

④ **Identity:**

↳ $r \cup \phi = \phi \cup r = r$

↳ $r \cdot \epsilon = \epsilon \cdot r = r$.

⑤ **Annihilator**

↳ ~~$r \cup \phi$~~ $= \phi \cdot r = r \cdot \phi = \phi$

⑥ **Idempotent:**

↳ $r \cup r = r$.

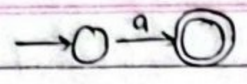
Equivalence of finite Automata and Regular Expression.

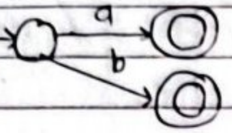
↳ we can convert each of regular expression into finite automata.

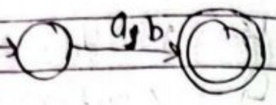
↳ Also, from given finite automata we can have regular expression.

↳ From R.E to F.A. → NFA or ENFA

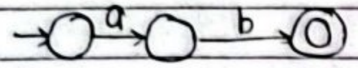
① $\phi \Rightarrow$

ii) a ⇒ 

iii) a+b ⇒ 

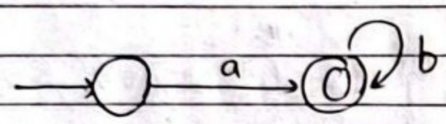
OR 

iv) a.b

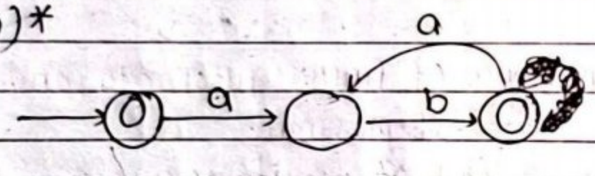


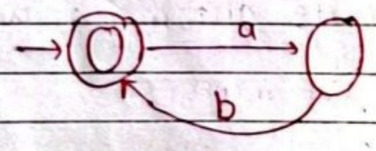
v) a*

vi) ab*

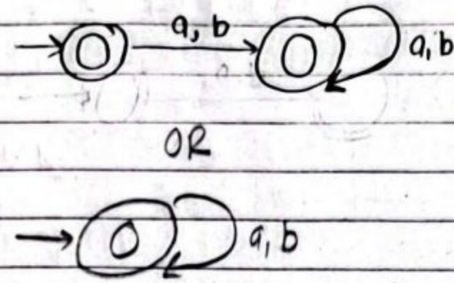


vii) (ab)*



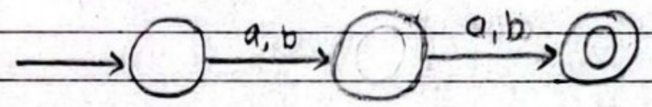
OR


viii) (a+b)*

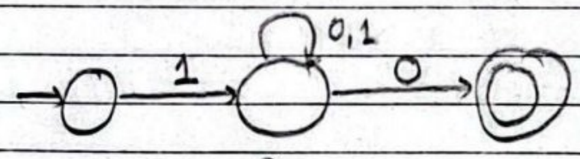


ix) (a+b) (a+b)

{ aa, ab, ba, bb }

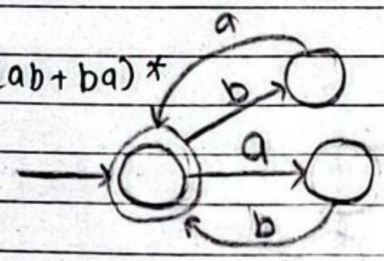


x) 1 (0+1)* 0 { 100, 110, 1000, 1010, 1100, 1110, ... }



abbaab

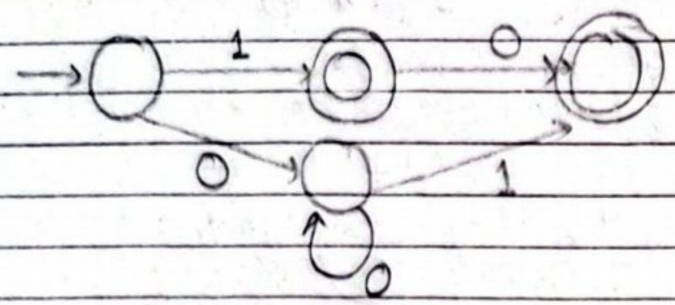
xi) (ab+ba)*



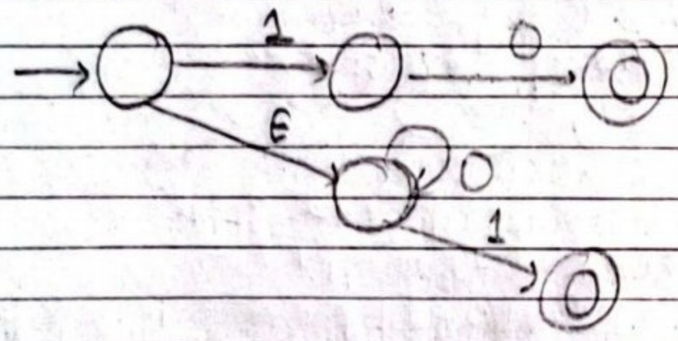
{ ab, ba, abab, abba,

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

01, 001, 0001 ...
10
1



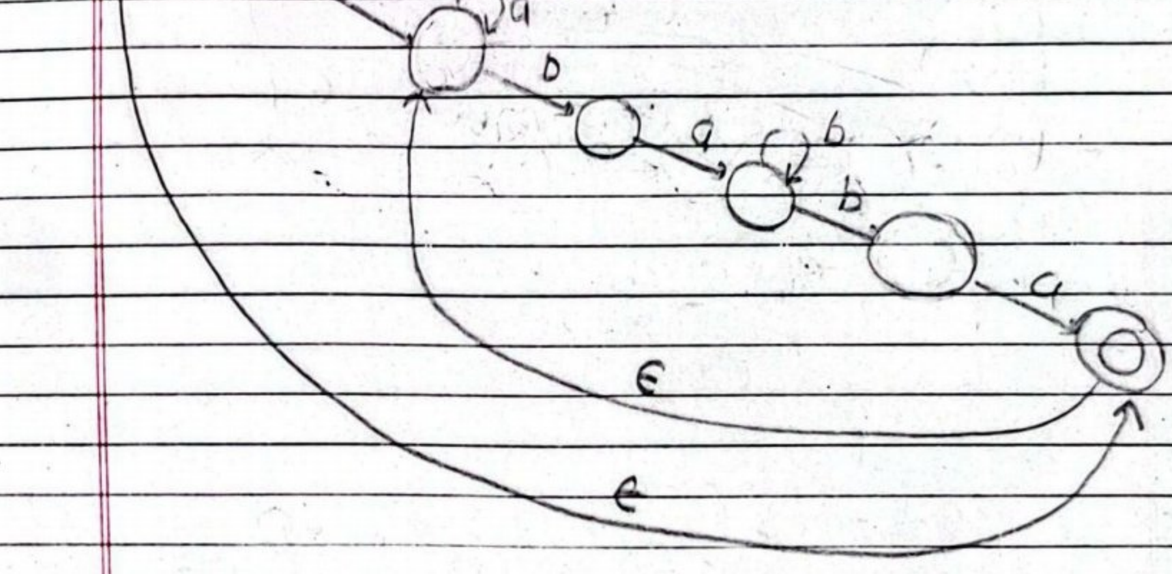
OR,



xiii) $(a^*ba\ b^*ba)^* \cup (b^*ab)$

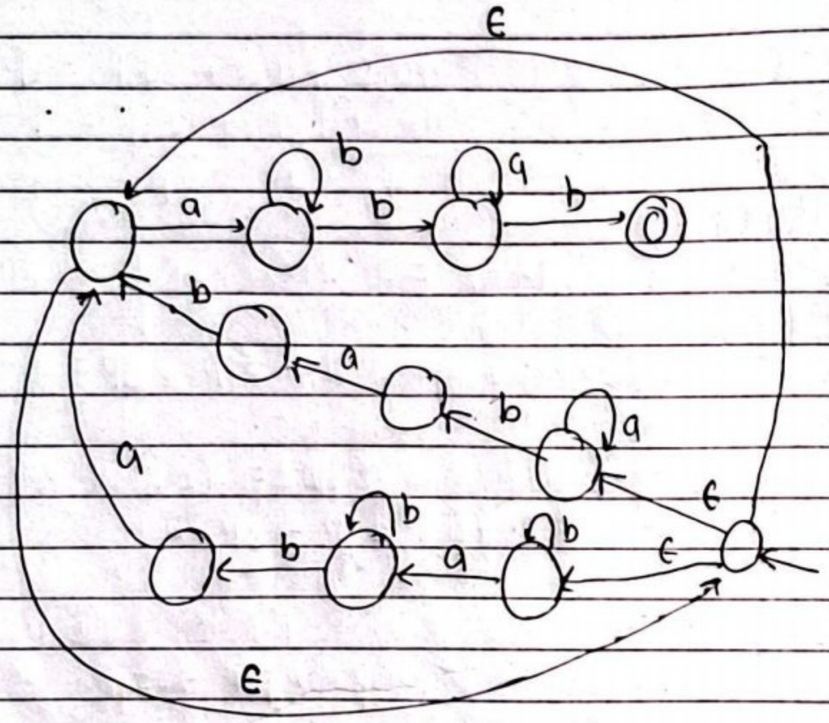
{ ab, bab, bbab, bbbab, ----? }

{ ϵ, baba, ababa, babba, ababba, aababbba? ..? }



xiv) $(a^*bab \cup b^*ab^*ba)^* ab^* ba^* b$

⇒ { babaabb, abaqbb, ababq bbab, ---}
{ abb }.



Date
Page

finite Automata

From FA to R.E

regular expression.

Arden's Theorem:

Statement:

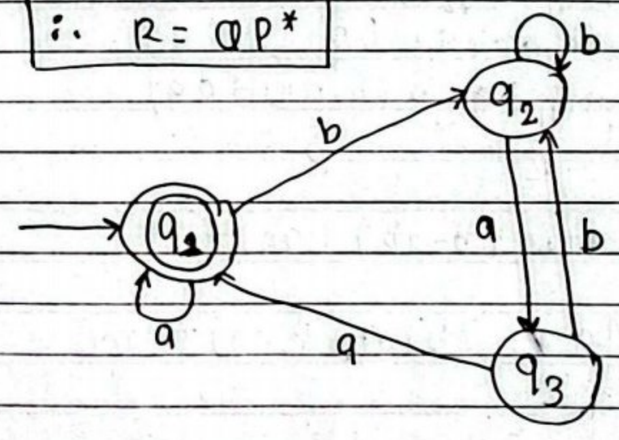
Let P and Q be two regular expressions over alphabet Σ, if P doesnot contain empty or null string then $R = Q + RP$ has a unique solution i.e $R = QP^*$

Proof:

$$R = Q + RP$$
$$\text{or, } R = Q + (Q+RP)P$$
$$\text{or, } R = Q + QP + RP^2$$
$$\text{or, } R = Q + QP + (Q+RP)P^2$$
$$\text{or } R = Q + QP + QP^2 + RP^3$$
$$\text{or, } R = Q + QP + QP^2 + QP^3 + \cdots$$
$$\text{or, } R = Q(\epsilon + P + P^2 + P^3 + \cdots)$$

$\therefore R = QP^*$

eg.



Soln

$q_1 = \epsilon + q_1 a + q_3 a$ ------- (i)
$q_2 = q_1 b + q_2 b + q_3 b$ ------- (ii)
$q_3 = q_2 a$ ------- (iii)

From eqn (ii) & (iii),

$q_2 = q_1 b + q_2 b + q_2 ab$
or, $\underset{R}{q_2} = \underset{Q}{q_1 b} + \underset{R}{q_2}(\underset{P}{b + ab})$

Compare thrs with $R = Q + RP$,

$R = q_2$, $Q = q_1 b$, $P = (b + ab)$

$\therefore$ we get, $R = QP^*$
i.e $q_2 = q_1 b (b + ab)^*$ ------- (iv)

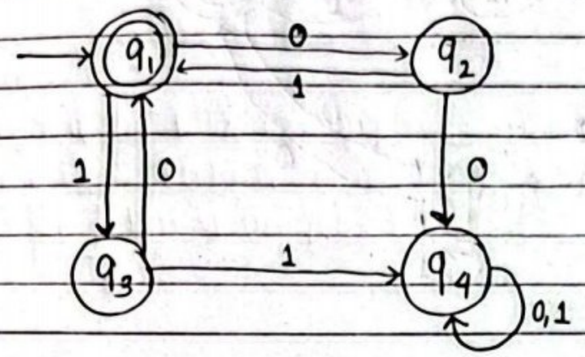
From (i),

$q_1 = \epsilon + q_1 a + q_2 aa$
$= \epsilon + q_1 a + q_1 b (b + ab)^* aa$
$\underset{R}{q_1} = \underset{Q}{\epsilon} + \underset{R}{q_1}(\underset{P}{a + b (b + ab)^* aa})$

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

$\therefore RE = (a + b (b + ab)^* aa)^*$ any.

Q1 Find regular expression of following F.A using Arden's theorem.



Solution:

$q_1 = \epsilon + q_3 0 + q_2 1$
$q_2 = q_1 0$

$q_3 = q_1 1$
$q_4 = q_2 0 + q_3 1 + q_4 0 + q_4 1$

From eqn (i), (ii), (iii)

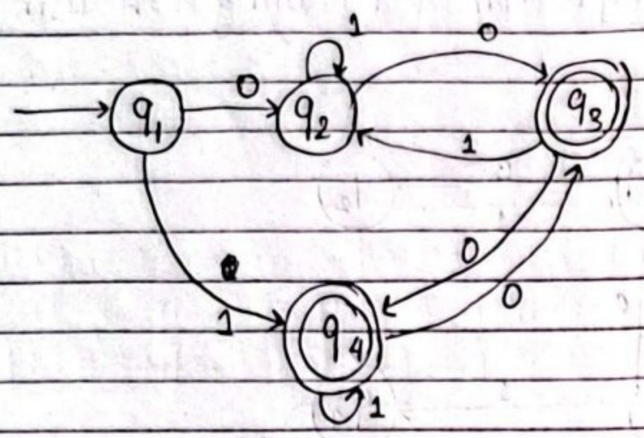
$q_1 = \epsilon + q_1 10 + q_1 01$

$\underset{R}{q_1} = \underset{Q}{\epsilon} + \underset{R}{q_1}(\underset{P}{10 + 01})$

Compare with $R = Q + RP$,
$R = q_1$, $Q = \epsilon$, $P = (10 + 01)$

we get, $R = QP^*$
$\therefore q_1 = \epsilon (10 + 01)^*$
$\therefore RE = (01 + 10)^*$ any.

Q)



Soln

$q_1 = \epsilon$ ——— (i)

$q_2 = q_1 0 + q_2 1 + q_3 1$ ——— (ii)

$q_3 = q_2 0 + q_4 0$ ——— (iii)

$q_4 = q_1 1 + q_3 0 + q_4 1$ ——— (iv)

From (iii) and (iv)

$q_4 = 1 + q_2 00 + q_4 00 + q_4 1$

or, $\underbrace{q_4}_{R} = \underbrace{1 + q_2 00}_{Q} + \underbrace{q_4}_{R}\underbrace{(00+1)}_{P}$

or, $q_4 = (1 + q_2 00)(00+1)^*$ ——— (v)

From (ii),

$q_2 = q_1 0 + q_2 1 + q_3 1$

or, $q_2 = 0 + q_2 1 + q_2 01 + q_4 01$

or, $q_2 = 0 + q_2(1+01) + (1+q_2 00)(00+1)^* 01$

or, $q_2 = 0 + q_2(1+01) + (00+1)^* 01 + q_2 00(00+1)^* 01$

or, $\underbrace{q_2}_{R} = \underbrace{0 + (00+1)^* 01}_{Q} + \underbrace{q_2}_{R}\underbrace{\left((1+01) + 00(00+1)^* 01\right)}_{P}$

$\therefore q_2 = (0 + (00+1)^* 01)((1+01) + 00(00+1)^* 01)^*$ ——— (vi)

$\therefore q_4 = (1 + (0+(00+1)^* 01)((1+01) + 00(00+1)^* 01)^* 00)(00+1)^*$

$q_3 = (0 + (00+1)^* 01)((1+01) + 00(00+1)^* 01)^* 0 + (1 + (0+(00+1)^* 01)((1+01) + 00(00+1)^* 01)^* 00)(00+1)^* 0$

Now,

$R.E = R.E(q_3) \cup R.E.(q_4)$

Closure properties of Regular language.

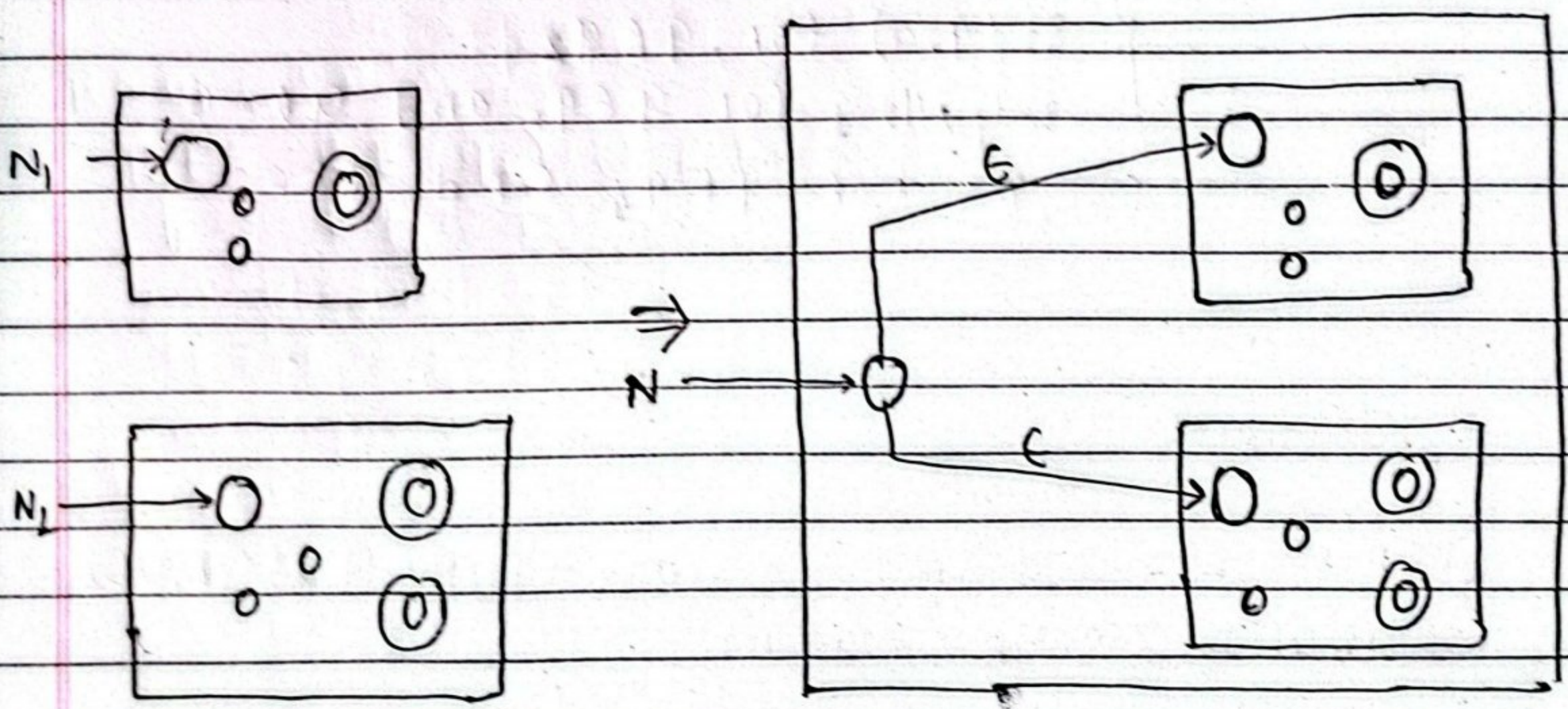
↳ The class of language accepted by finite auto mata is closed under,

(i) Union (ii) Concatenation (iii) Kleene closure.

(i) Union:

↳ Let L_1 and L_2 are two regular languages. Now prove $L_1 \cup L_2$ is regular.

↳ Here, idea is take two NFA's N_1 and N_2 for L_1 and L_2 respectively and combine them to one new NFA N such that it accepts input i.e either accepted by N_1 or N_2.



[Fig: Construction of NFA to recognize $L_1 \cup L_2$]

let,

$N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize L_1

$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize L_2

Now,

$N = (Q, \Sigma, \delta, q_0, F)$ is such that,

$L(N) = L_1 \cup L_2$

Here,

$Q = Q_1 \cup Q_2 \cup \{q_0\}$

q_0 = start state of N

$F = F_1 \cup F_2$

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{for } q \in Q_1 \\ \delta_2(q,a) & \text{for } q \in Q_2 \\ \{q_1, q_2\} & \text{for } q \in q_0 \text{ and } a = \epsilon \\ \phi & \text{for } q \in q_0 \text{ and } a \neq \epsilon \end{cases}$$

Example:

Q) let, $L_1 = aba$ and $L_2 = bag$. Show that $L_1 \cup L_2$ is also regular language.

Solution,

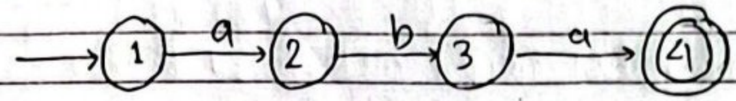
NFA of

let, N_1 is language of L_1.

N_2 is NFA of L_2.

So,

$N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ represents L_1.



$Q_1 = \{1,2,3,4\}$

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

$q_1 = \{1\}$

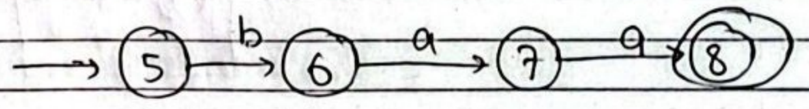
$F_1 = \{4\}$

$\delta_1(1,a) = 2$

$\delta_1(2,b) = 3$

$\delta_1(3,a) = 4$

$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ represent L_2.



$Q_2 = \{5,6,7,8\}$

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

$q_2 = \{5\}$

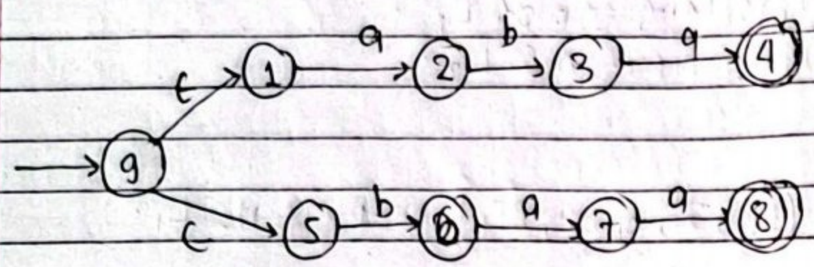
$F_2 = \{8\}$

$\delta_2(5,b) = 6$

$\delta_2(6,a) = 7$

$\delta_2(7,a) = 8$

Combine N_1 and N_2 to make new $N = (Q, \Sigma, \delta, q_0, F)$ as follows:



Here,

$Q = \{\underbrace{1,2,3,4}_{q_1}, \underbrace{5,6,7,8}_{q_2}, 9\}$

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

$q_0 = \{9\}$

$F = \{4, 8\}$
$\quad\quad\downarrow \quad\downarrow$
$\quad\quad f_1 \quad f_2$

δ :-

$\left.\begin{array}{l} \delta(9, \epsilon) = \{1, 5\} \\ \delta(1, a) = 2 \\ \delta(2, b) = 3 \\ \delta(3, a) = 4 \end{array}\right\} \delta_1$

$\left.\begin{array}{l} \delta(5, b) = 6 \\ \delta(6, a) = 7 \\ \delta(7, a) = 8 \end{array}\right\} \delta_2$

N accepts $L_1 \cup L_2$.

$\therefore L_1 \cup L_2$ is regular language.

(ii) Concatenation :

↳ Let L_1 and L_2 are two regular languages now show that $L_1 \cdot L_2$ is also regular.

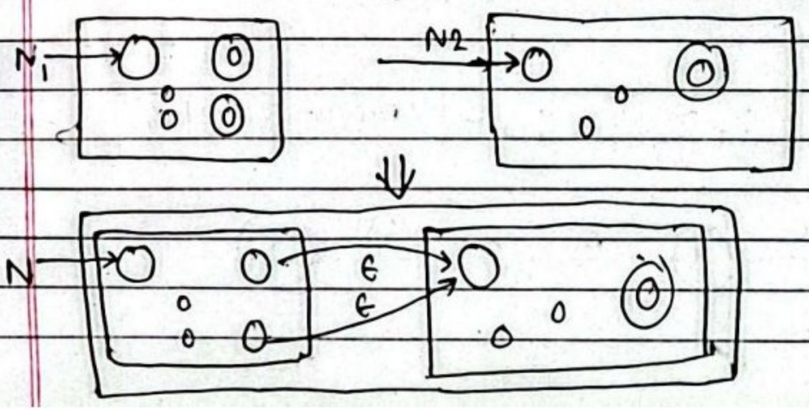
↳ Here, idea is take two NFA's N_1 and N_2 for L_1 and L_2 respectively. combine them to make new NFA N.

↳ Assign N's starting state to be start state of N_1.

↳ The accept state of N_1 have additional ϵ-arrows that non deterministically allow branching to N_2 whenever N_1 is in accepting state.

↳ The accept state of N are that of N_2 only.

↳ So, N accept when input is split into two parts where first is accepted by N_1 and second by N_2.



Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize L_1
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize L_2.

NOW,
$N = (Q, \Sigma, \delta, q_0, F)$ is such that.

$L(N) = L_1 \cdot L_2$

Here,

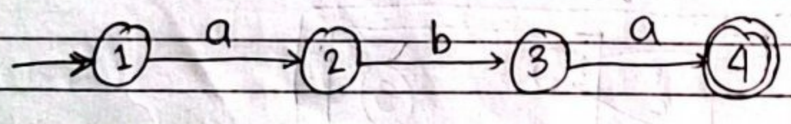
$Q = Q_1 \cup Q_2$
q_0 = start state of $N = q_1$
F = accept state of $N = F_2$

$$\delta(q,a) = \begin{cases} \delta_1(q,a) \text{ for } q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a) \text{ for } q \in F_1 \text{ and } a \neq \epsilon \\ \delta_1(q,a) \cup \{q_2\} \text{ for } q \in F_1 \text{ and } a = \epsilon \\ \delta_2(q,a) \text{ for } q \in Q_2 \end{cases}$$

Q) Example: Let $L_1 = aba$ and $L_2 = baq$, show that $L_1 \cdot L_2$ is also regular language.

Soln:

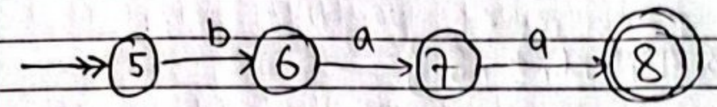
Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize L_1:



$Q_1 = \{1,2,3,4\}$
$\Sigma = \{a,b\}$
$q_1 = 1$
$F_1 = \{4\}$
δ_1:-

$\delta_1(1,a) = 2$
$\delta_1(2,b) = 3$
$\delta_1(3,a) = 4$

Let $N_2 = (Q_2, \Sigma, \delta_2, q_1, F_2)$ recognize L_2:

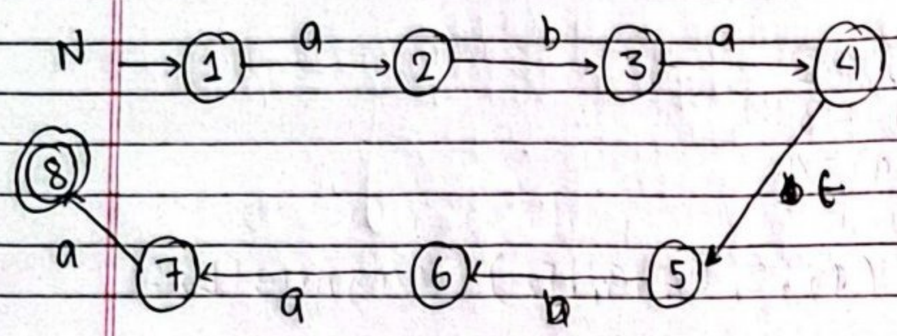


$Q_2 = \{5,6,7,8\}$
$\Sigma = \{a,b\}$
$q_2 = 5$
$f_2 = \{8\}$

$\delta_2(5,b) = 6$
$\delta_2(6,a) = 7$
$\delta_3(7,a) = 8$

NOW,

Combine N_1 and N_2 to make new $N = (Q, \Sigma, \delta, q_0, F)$ as follows:



$Q = \{1, 2, 3, 4, 5, 6, 7, 8\}$
$\Sigma = \{a, b\}$
$q_0 = \{1\}$
$f_1 = \{8\}$

δ:

$\delta(1, a) = 2$
$\delta(2, b) = 3$
$\delta(3, a) = 4$
$\delta(4, \epsilon) = 5$
$\delta(5, b) = 6$
$\delta(6, a) = 7$
$\delta(7, a) = 8$

(iii) Kleene closure:

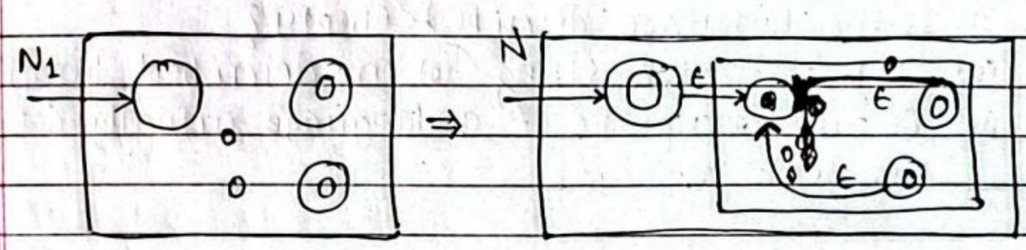
↳ For a regular language A, prove A^* is also regular.

↳ take a NFA N_1 for A and make a new NFA N for A^*.

↳ N will accept its input whenever it can be broken into several pieces and N_1 occupies each piece.

↳ Here, construct N like N_1 with additional ϵ arrow returning to start state from accept state.

↳ modify N so that it accepts ϵ, which always is member of A^* so here add new start state which is also accept state and has ϵ arrow to old start state.



let $N_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ recognize A.

now,

$N = (Q, \Sigma, \delta, q_0, F)$ to recognize A^*.

~~Here,~~

Here,

$Q = Q_1 \cup \{q_0\}$

q_0 = start state of N

$F = F_1 \cup \{q_0\}$

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{for } q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a) & \text{for } q \in F_1 \text{ and } a = \epsilon \\ \delta_1(q,a) \cup \{q_1\} & \text{for } q \in F_1 \text{ and } a = \epsilon \\ \{q_1\} & \text{for } q = q_0 \text{ and } a = \epsilon \\ \phi & \text{for } q = q_0 \text{ and } a \neq \epsilon \end{cases}$$

DECESION properties of Regular language:

↳ Here, we consider some of the fundamental question about language:

① Is the language described empty?
② Is a particular string ω in described language?
③ Do two descriptions of a language actually describe the same language?

↳ We can answer the above questions as follows:

① Testing emptiness of regular language:

↳ If our representation is any kind of finite automata, the emptiness question is whether there is any path whatever from start state to some accepting state.

↳ If yes, then language is non empty

↳ Else language is empty.

② Testing membership in a regular language:

↳ Let L be any regular language and ω is any string. we have to test whether $\omega \in L$ or not.

↳ we can represent L by some finite automata then simulate the automata processing the string of input symbols ω begining in the start state.

↳ If automata ends in accepting state, the answer is yes otherwise no.

③ Conversion among representations:

↳ some language can be represented by more than one definations.

- convert NFA to DFA and vice-versa
- convert automata to R.E and vice versa.

Pumping lemma for Regular language:

↳ Informally, it says that all sufficient long words, in a regular language may be pumped i.e have a middle section of word repeated an arbitary no. of times to produce new word also lies with in same language.

↳ pumping lemma is a powerful technique to show that language is not regular.

Statement :

- let L be a regular language
- w be any string, $w \in L$ with $|w| \geq n$, where n be integer constant.
- there are strings x, y, z such that $w = xyz$ where
  $|xy| < n$
  $|y| > 0$

Then,

$xy^iz \in L$ for all $i \geq 0$

Here, substring y is pumped.

proof:

↳ Suppose L is a regular language, then L is accepted by same DFA M.

↳ let M has n states.

↳ let M accepts string w of length n or greater

↳ let length of w, $|w| = m$ such that $m \geq n$.

↳ Here, $w = a_1, a_2, \dots, a_m$ where $m \geq n$, each $a_i \in \Sigma$ be input symbol to M.

↳ For $j = 1 \dots n$, q_j be state of M.

↳ Then,

$$\begin{aligned}\delta(q_0, w) &= \delta(q_0, a_1, a_2, \dots a_m) \\ &= \delta(q_1, a_2, \dots a_m) \\ &= - - - \\ &= - - - \\ &= \delta(q_n, \epsilon)\end{aligned}$$

↳ Since,

$m \geq n$ and DFA M has only n states by pregonhole principle, there exists some i an j° with $0 \leq i \leq j \leq n$, such that $q_i = q_j$

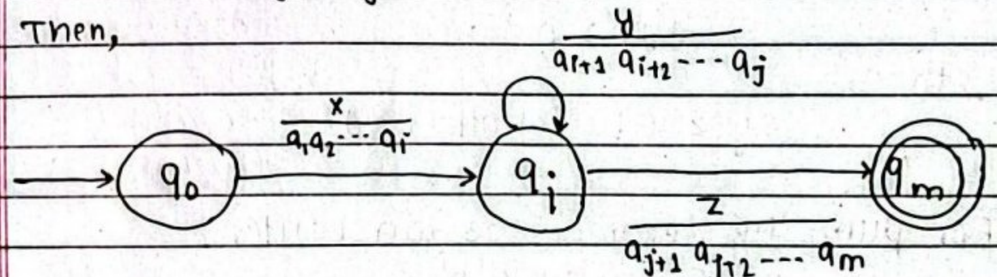
we can break, $w = xyz$ as follows.

$x = a_1 a_2 \dots a_i$

$y = a_{i+1} a_{i+2} \dots a_j$

$z = a_{j+1} a_{j+2} \dots a_m$

Then,



i.e. starting a_{i+1} a_{i+2} ... a_j takes M from q_i back to

itself since $q_i = q_j$.

So,

M accepts $(q_1, q_2, \cdots q_i)(q_{i+1} q_{i+2} \cdots q_j)^*(q_{j+1} q_{j+2} \cdots q_m)$

for all $k \geq 0$

:

$\therefore xy^iz \notin L$ for all $i \geq 0$.

e.g. Show that $L = \{a^i b^i ; i \geq 1\}$ is not regular language using pumping lemma for regular language.

Solution:

↳ Let L be any regular language

↳ n be pumping constant

↳ w be any ~~simple~~ string, $w \in L$ with $|w| \geq n$.

↳ w can be rewritten as

$w = xyz$ such that

$|xy| \leq n$

$|y| > 0$

then,

$xy^iz \in L$ for all $i \geq 0$.

For pumping lemma we can write,

$w = xyz = a^i b^i$

$a^5 b^5 =$ aaaaa bbbbb

x y

$|ab| \leq n$

$5 \leq 5$ ∝ (milyo) length.

→ It can be written as,

$w = xyz =$ aa..... aa..... ab..... bb..... b

x y z

here,

$y = a^k ; k \geq 1$

Now,

from pumping lemma.

for $i = 0$,

$w = xz$

$= a^{n-k} b^n \notin L$

i.e contradiction with theorem.

for $i = 2$,

$w = xy^2z$

$= xyyz$

$= a^{n+k} b^n \notin L$

i.e. contradiction with theorem.

So given language,

$L = \{a^i b^i ; i \geq 1\}$ is not regular language.

Chapter-3

context free language and pushdown automata

Date
Page

context free Grammar (CFG)

↳ A more powerful method of describing language.
↳ It can describe certain features that have a recursive structure.
↳ An important application of CFG occurs in specifications and compilation of programming languages.
↳ the collection of languages associated with CFG are called context free languages (CFL).
↳ CFG is a language generator on some set of rules.

eg.

$A \longrightarrow 0A1$
$A \longrightarrow B$
$B \longrightarrow \#$

↳ In grammar, collection of substitution rules are called productions.
↳ each rules comprise a symbol and string seperated by arrow.
↳ symbol is called variables.
↳ strings that consist of variable and other symbols are called terminal string.
↳ one variable is a start variable that usually occurs on the left hand side of the topmost rule.

In above eg. grammar contains 3 rules. A and B are variables. 0, 1, # are terminals and A is a start symbol

↳ above grammar can generate string 000#111 as follows.

$A \Rightarrow 0A1$
$\Rightarrow 00A11$
$\Rightarrow 000A111$
$\Rightarrow 000B111$
$\Rightarrow 000\#111$

Formal defination :

- a CFG is a 4 tuple (V, Σ, R, S) or (V, T, P, S).
  or
  Here,
  V is a finite set of variables.
  Σ or T is a finite set of Terminals.
  R or P is a finite set of rules.
  S is a start variable, $S \in V$.

Some terminologies :

① **Derivation :**

↳ Let $G = (V, \Sigma, R, S)$ be a CFG. If $\omega_1, \omega_2, \ldots \omega_n$ are strings over variable ~~sum~~ V such that, $\omega_1 \Rightarrow \omega_2 \Rightarrow \cdots \Rightarrow \omega_n$ then we say ω_n is derivable.

↳ the sequence of substitution to obtain a string is called derivation. say ω_1 derives ω_n i.e. $\omega_1 \overset{*}{\Rightarrow} \omega_n$. then the sequence of steps to obtain ω_n from ω_1 is called derivation.

② **language of CFG (LCG):**

↳ If $G = (V, \Sigma, R, S)$ is a CFG. then the language of G denoted by L(G) is the set of terminal strings that have derivations from start symbol i.e.

$$L(G) = \{ \omega \in \Sigma^* ; \; S \overset{*}{\Rightarrow} \omega \}$$

③ **Sentential form:**

↳ Derivations from start symbol produce strings that have special rules. this is called sequential form.
i.e. if $G = (V, \Sigma, R, S)$ is a CFG. Then any string α such that $S \overset{*}{\Rightarrow} \alpha$ is a sentential form.

↳ if $S \underset{lm}{\overset{*}{\Rightarrow}} \alpha$ then α is a left sentential form.

↳ if $S \underset{rm}{\overset{*}{\Rightarrow}} \alpha$, then α is a right sentential form.

leftmost and Rightmost derivation:

↳ In leftmost derivation, leftmost variable is replaced first.

↳ In rightmost derivation, rightmost variable is replaced first.

eg.

$E \longrightarrow I$

$E \longrightarrow E+E \mid E*E \mid (E)$

$I \longrightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

now,

$E*(I+E)$ is a sentential form as there is a derivation:

$E \Rightarrow E*E$

$\Rightarrow E*(E)$

$\Rightarrow E*(E+E)$

$\Rightarrow E*(I+E)$

i) consider $a*E$

$E \underset{lm}{\Rightarrow} E*E$

$\underset{lm}{\Rightarrow} I*E$

$\underset{lm}{\Rightarrow} a*E$

$\therefore a*E$ is left sentential form.

ii) consider $E*(E+E)$

$E \underset{rm}{\Rightarrow} E*E$

$\underset{rm}{\Rightarrow} E*(E)$

$\underset{rm}{\Rightarrow} E*(E+E)$

$\because E*(E+E)$ is right ~~sequential~~ sentential form.

Derivative Tree / parse Tree: (syntax tree)

↳ It is a tree representation of strings of terminals using the productions defined by the grammar.

↳ It pictorically shows how the start symbol of a grammar derives a string in language.

↳ is an ordered tree that shows essentials of derivations.

↳ Formally, given CFG, $G = (V, \Sigma, R, S)$, a parse tree has following properties :

i) the root is labelled by start symbol.
ii) each interior nodes of tree are variables.
iii) each left node is labelled by terminal symbol or ϵ.
iv) if an interior node is labelled with non-terminal A and its children are $x_1, x_2, \dots x_n$ from left to right then there is a production p as,

$$A \longrightarrow x_1 x_2 \dots x_n \text{ for each } x_i \in \Sigma.$$

→ consider a grammer where,

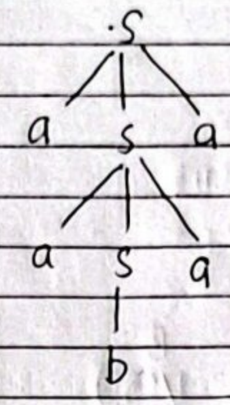
$$S \longrightarrow aSa \mid a \mid b \mid \epsilon$$

Now. for string aabaa

derivation is as follows:

$$S \Rightarrow aSa$$
$$\Rightarrow aaSaa$$
$$\Rightarrow aabaa$$

It's parse tree is:



Q) Consider a grammar

$$S \longrightarrow A1B$$
$$A \longrightarrow 0A \mid \epsilon$$
$$B \longrightarrow 0B \mid 1B \mid \epsilon$$

derive and construct parse tree for following strings.

i) ~~00101~~ 00101

$S \Rightarrow A1B$
$\Rightarrow 0A1B$
$\Rightarrow 00A1B$
$\Rightarrow 00\epsilon 1B$
$\Rightarrow 00\epsilon 10B$
$\Rightarrow 00\epsilon 101B$
$\Rightarrow 00\epsilon 101\epsilon$
$\Rightarrow 00101$

A1B
0A10B
00A101B
00ε101ε
00101

ii) 00011

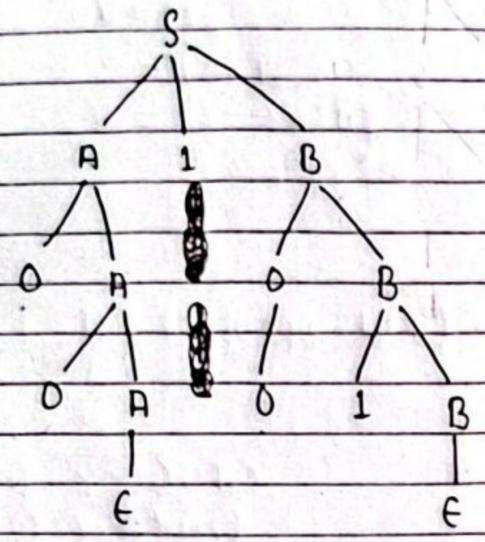
$S \Rightarrow A1B$
$\Rightarrow 0A1B$
$\Rightarrow 00A1B$
$\Rightarrow 000A1B$
$\Rightarrow 000\epsilon 11B$
$\Rightarrow 000\epsilon 11\epsilon$
$\Rightarrow 00011$

0A11B
00A11ε
000A11
000ε11
00011

Date
Page

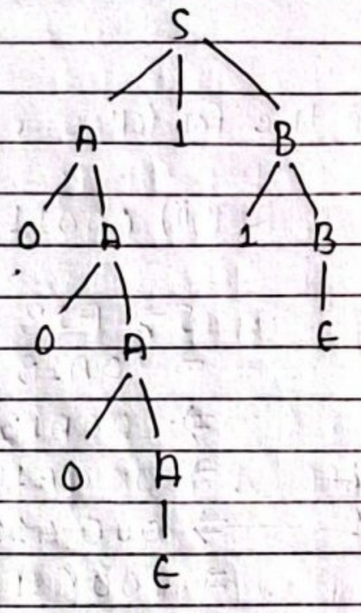
It's paise tree:

i)



001001

ii)



00011

Date
Page

Q) consider a grammar:

$S \longrightarrow S+S$
$S \longrightarrow S*S$
$S \longrightarrow S/S$
$S \longrightarrow S-S$
$S \rightarrow (S)$
$S \rightarrow a$

derive string $a \rightarrow (a*a)/a-a$

(i) left most Derivation and
(ii) right most derivation.

Also draw paise tree.

LMD:

$S \Rightarrow S+S$
$\Rightarrow a+S$
$\Rightarrow a+S/S$
$\Rightarrow a+(S)/S$
$\Rightarrow a+(S*S)/S$
$\Rightarrow a+(a*S)/S$
$\Rightarrow a+(a*a)/S$
$\Rightarrow a+(a*a)/S-S$
$\Rightarrow a+(a*a)/a-S$
$\Rightarrow a+(a*a)/a-a.$

RMD:

$S \Rightarrow S/S$
$\Rightarrow (S)/S-S$
$\Rightarrow (S)/S-S$
$\Rightarrow (S)+S/S-S$
$\Rightarrow S+(S)/S-S$
$\Rightarrow a+(S)/S-S$
$\Rightarrow a+(a*S)/S-S$
$\Rightarrow a+(a*S)/S-S$
$\Rightarrow a+(a*a)/S-S$
$\Rightarrow a+(a*a)/a-S$
$\Rightarrow a+(a*a)/a-a$

Q) Let $G = (V, \Sigma, R, S)$ where,

$V = \{S\}$
$\Sigma = \{a, b\}$
$R = \{S \to aSb$
$S \to ab$
$\}$

Find the language $L(G)$.

Soln:

i) $S \Rightarrow ab$
ii) $S \Rightarrow aSb$
$\Rightarrow aabb$
iii) $S \Rightarrow aSb$
$\Rightarrow aaSbb$
$\Rightarrow aaabbb$

Q)
iv) $S \Rightarrow aSb$
$\Rightarrow aaSbb$
$\Rightarrow$ ~~aaaSbbb~~ $\Rightarrow aaaSbbb$
$\Rightarrow$ ~~aaaaPbbb~~.

$\therefore$ ~~$L(G) = \{a$~~ $\Rightarrow a^{n-1} S b^{n-1}$
$\Rightarrow a^n b^n$

$\therefore L(G) = \{a^n b^n : n \geq 0\}$

Q) Let $G = (V, \Sigma, R, S)$, where,

$V = \{S\}$
$\Sigma = \{a, b\}$
$R = \{$
$S \to aaS$
$S \to b$
$\}$

Find $L(G)$

Solution:

i) $S \Rightarrow b$
ii) $S \Rightarrow aaS$
$\Rightarrow aab$
iii) $S \Rightarrow aaS$
$\Rightarrow aaaaaS \Rightarrow aaaab$

iii) $S \Rightarrow aaS$
$\Rightarrow aaaaS$
$\Rightarrow aaaaaaS$
$\Rightarrow aaaaaab$

iv) $S \Rightarrow aaS$
$\Rightarrow aaaaS$
$\Rightarrow aaaaaaS$
$\Rightarrow a^{2n} b$

$\therefore L(G) = \{a^{2n} b : n \geq 0\}$

Q) Let $G = (V, \Sigma, R, S)$ where,

$V = \{S, A\}$

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

$R = \{$
$S \longrightarrow aA|\epsilon$
$A \longrightarrow bS$
$\}$

Find $L(G)$.

Solution:

i) $S \Longrightarrow \epsilon$

ii) $S \Rightarrow aA$
$\Rightarrow abS$
$\Rightarrow ab\epsilon$

iii) $S \Rightarrow aA$
$\Rightarrow abS$
$\Rightarrow abaA$
$\Rightarrow ababS$
$\Rightarrow abab\epsilon$

iv) $S \Rightarrow aA$
$\Rightarrow abS$
$\Rightarrow abaA$
$\Rightarrow ababS$
$\Rightarrow ababaA$
$\Rightarrow abababS$
$\rightarrow ababab\epsilon$

v) $S \Longrightarrow aA$
$\Rightarrow abS$
$\Rightarrow abaA$
$\Rightarrow ababS$
⋮
$(ab)^n$

$\therefore L(G) = \{(ab)^n : n \geq 0\}$

Q) Write a CFG for language.
$L = \{a^m b^n : m \geq n\}$

Solution:

Let CFG be $G = (V, \Sigma, R, S)$
where,
$V = \{S\}$
$\Sigma = \{a, b\}$
$R = \{ S \longrightarrow \epsilon$
$S \longrightarrow aS$
$S \longrightarrow aSb$
$\}$
$S = S$

Now, Generalizing the grammar for the language, we get,

$L = \{\epsilon, a, aa, aaa, \cdots, ab, aab, aaabb, \cdots aaab, \cdots$

Now,

The grammar is,

$S \longrightarrow G$

$S \longrightarrow a$
$S \longrightarrow aa$
$S \longrightarrow aaa$
$S \longrightarrow aaaa$
} $S \longrightarrow aS$

$S \longrightarrow aSb$.

examples:

$S \Rightarrow aSb$
$\Rightarrow aaSbb$
$\Rightarrow aaaSbb$
$\Rightarrow aaaεbb$
$= aaabb$.

Q) write a CFG for language,

$L = \{\omega\omega^R : \omega \in \{0,1\}^*\}$

$L = \{\epsilon, 00, 11, 0110, 1001, 010010, 110011\}$

So,

$V = \{S\}$
$\Sigma = \{0,1\}$
$R = \{$
$S \longrightarrow \epsilon$
$S \longrightarrow 0S0$
$S \longrightarrow 1S1$

$S := \{S\}$.

Q) write a CFG for language:

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

solution,

$L = \{ab, aab, abb, aabb, aaab, aaabb, abbb, abbbb, \ldots\}$

So, $R = \{$
$S \longrightarrow aSb$
$S \longrightarrow ab$
$S \longrightarrow aS$
$S \longrightarrow Sb$
$\}$

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

aaabbbbb.

$S \longrightarrow aSb$
$\rightarrow aaSbb$
$\rightarrow$ ~~aaabS~~
$\rightarrow aaSbbb$
$\rightarrow aaSbbbb$
$\rightarrow aaabbbbb$.

OR,

$S \longrightarrow AB$
$A \rightarrow a$
$A \longrightarrow aA$
$B \rightarrow b$
$B \longrightarrow bB$

Ambiguous Grammer:

↳ a grammar $G = (V, \Sigma, R, S)$ is said to be ambiguous if there is a string $w \in L(G)$ for which we can derive two or more distinct derivation tree rooted at S and yielding w.

↳ In other words, a grammar is ambiguous if it can produce more than one leftmost derivation or more than one rightmost derivation for some string in language of grammar.

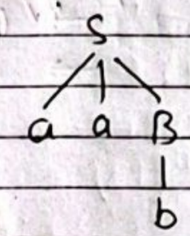
eg.

$S \longrightarrow AB \mid aaB$.
$A \rightarrow a \mid Aa$
$B \rightarrow b$

Now for string aab, we have two leftmost derivation and distinct paise tree as follows,

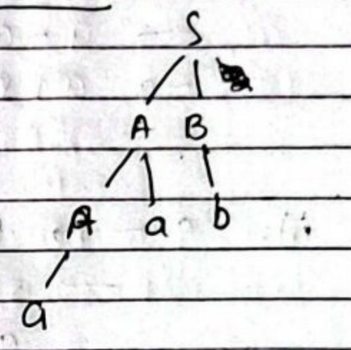
(i) $S \Rightarrow aaB$
$\Rightarrow aab$

paise tree:



(ii) $S \Rightarrow AB$
$\Rightarrow AaB$
$\Rightarrow aaB$
$\Rightarrow aab$

parse tree:



example:

Consider Grammar,

$S \longrightarrow S+S$
$S \longrightarrow S*S$
$S \longrightarrow a \mid b$

for string a+a*b, show that above grammar is ambiguous.

Solution:

a+a*b

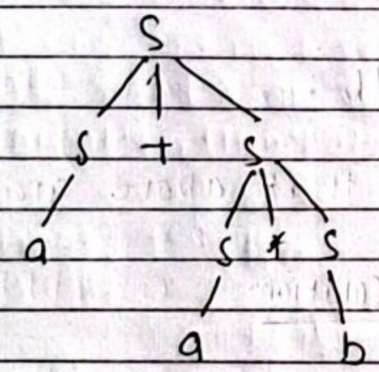
i) a + a*b

ii) ~~a+a*b~~

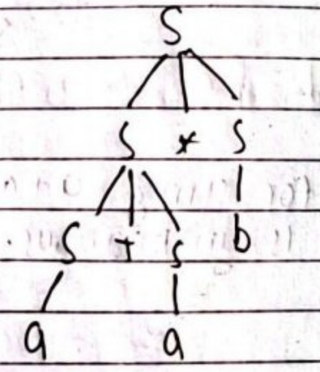
$S \longrightarrow S+S$
$\rightarrow a+S$
$\rightarrow a+S*S$
$\rightarrow a+a*S$
$\rightarrow a+a*b$

ii) $S \longrightarrow S*S$
$\rightarrow S+S*S$
$\rightarrow a+S*S$
$\rightarrow a+a*S$
$\rightarrow a+a*b$

parse tree (i)



parse tree (ii).



As they have unique left most derivation parse tree, it is ambiguous grammar.

(i) we must eliminate useless symbol
ii) we must eliminate empty production.
iii) we must eliminate unit production.

Simplification of context free diagram:

i) Eliminating useless symbol:

↳ Useless symbols are those variables or terminals that do not appear in any derivation of a terminal string from start symbol.

↳ For a grammar $G = (V, \Sigma, P, S)$ a symbol x is useful if there is some derivation of the form $S \overset{*}{\Rightarrow} \alpha x \beta$.
↳ Derives in zero or more step

↳ Elimination of useless symbol include identifying whether or not the symbol is 'generating' and 'reachable'

↳ a symbol x is generating if $x \overset{*}{\Rightarrow} w$ for some terminal string w.

↳ a symbol x is reachable if there is derivation $S \overset{*}{\Rightarrow} \alpha x \beta$ for some α & β.

↳ Thus, we elimirate non-generating and non-rechable symbols.

e.g: consider a grammar:

$S \longrightarrow aB \mid bX$
$A \longrightarrow Bad \mid bSX \mid a$ (non reachable)
$B \longrightarrow aSB \mid bBX$ (non generating)
$X \longrightarrow SBd \mid aBX \mid ad.$

here, A and x are generating symbols as:

$A \longrightarrow a$

and $x \longrightarrow ad$

Similarly, S is also generating symbol as,

$S \Rightarrow bx$

$\Rightarrow bad$

But B is non-generating because we can't get any terminal strings.

∴ eliminate B.

we get,

$S \longrightarrow bx$

$A \longrightarrow bsx \mid a$

$x \longrightarrow ad$

here, S and x are reachable symbols but, A is not reachable, eliminate it.

Finally we get,

$S \longrightarrow bx$

$x \longrightarrow ad.$

ii) Eliminating empty production:

↳ A grammar is said to have empty production if there is a production of the form $A \longrightarrow \epsilon$

↳ here, A is nullable variable.

↳ if $A \longrightarrow \epsilon$ is a production to be eliminated then we look for all productions whose right side contain A and replace each occurance of A in each of those occurance of A in each of those production to obtain non empty production.

↳ Now, those resultant non-empty productions must be added to grammar to keep the language generated to be same.

e.g.

$S \longrightarrow aA$

$A \longrightarrow b \mid \epsilon$

Soln:

here, $A \longrightarrow \epsilon$ is empty production where A is nullable variable.

To eliminate $A \longrightarrow \epsilon$, first list all the productions having 'A' in R.H.S's then replace each A with ϵ and get non empty productions.

$S \longrightarrow aA \Rightarrow S \longrightarrow a$

Now,

Updated grammar is,

$S \longrightarrow aA \mid a$

$A \rightarrow b.$

eg.

$S \longrightarrow ABAC$

$A \longrightarrow aA \mid \epsilon$

$B \longrightarrow bB \mid \epsilon$

$C \longrightarrow c$

Solution:

Here, $A \longrightarrow \epsilon$ is an empty production so, A is nullable variable.

To eliminate $A \rightarrow \epsilon$, first listing all the production having 'A' in R.H.S.

Then,

$S \longrightarrow ABAC$

$A \longrightarrow aA$

So,

$S \longrightarrow ABAC \Rightarrow S \longrightarrow BAC \mid ABC \mid BC$

$A \longrightarrow aA \Rightarrow A \longrightarrow a$

∴ Updated grammar after eliminating $A \rightarrow \epsilon$:

$S \longrightarrow ABAC \mid BAC \mid ABC \mid BC$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid \epsilon$

$C \longrightarrow C$

Now, Again we got $B \longrightarrow \epsilon$ so, B is nullable variable.

To eliminate $B \rightarrow \epsilon$, first listing all the productions having 'B' in R.H.S.

then,

$S \longrightarrow ABAC \Rightarrow S \longrightarrow AAC$

$S \longrightarrow BAC \Rightarrow S \longrightarrow AC$

$S \longrightarrow ABC \Rightarrow S \longrightarrow AC$

$S \longrightarrow BC \Rightarrow S \longrightarrow C$

$B \longrightarrow bB \Rightarrow B \longrightarrow b$

Finally,

$S \longrightarrow ABAC \mid BAC \mid ABC \mid BC \mid AAC \mid AC \mid aC \mid C$

$A \longrightarrow aA \mid a$

$B \longrightarrow bB \mid b$

$C \longrightarrow c$

iii) Eliminating unit production:

↳ a unit production is a production of the form $A \to B$ where both A and B are variables.

⇒ Algorithm:

while (there exists unit productions $A \to B$)
{

i) select a unit production $A \to B$, such that there exists a production $B \to \alpha$ where α is terminal.
ii) for (every non unit production, $B \to \alpha$)
- add production $A \to \alpha$ to grammer.

iii) eliminate $A \to B$ from grammar

}

eg:
$A \to B$
$B \to a$
$B \to c$

soln:
Here, $A \to B$ is a unit production. we have $B \to a$ & $B \to c$, so we use can eliminate $A \to B$ and get final grammar as follow

$A \to a$
$A \to c$
$B \to a$
$B \to c$.

eg:

$S \to AB$
$A \to a$
$B \to \epsilon | b$
$C \to D$
$D \to E$
$E \to a$

Here, $B \to C$, $C \to D$, $D \to E$ are unit production we have,
$A \to a$, $B \to b$ so we can eliminate.
$S \to AB$, $B \to C$, $C \to D$, $D \to E$ and get final production,

Initially we cannot remove
$B \to C$
$C \to D$
so removing $D \to E$ at first as E has terminal.

So, $D \to E \Rightarrow D \to a$

Now,
$C \to D \Rightarrow C \to a$

And,
$B \to C \Rightarrow B \to a$

So final production is,

$S \longrightarrow AB$
$A \longrightarrow a$
$B \longrightarrow a|b$
$C \longrightarrow a$
$D \longrightarrow a$
$E \longrightarrow a$

Chomsky Normal Form (CNF):

↳ a CFG $G = (V, \Sigma, R, S)$ is said to be in Chomsky Normal form (CNF) if every productions in G are in one of two forms.

$A \longrightarrow BC$
$A \longrightarrow a$

where,
A, B, C are variables and a is terminal.

↳ So, a grammar in CNF is one which should not have,
i) useless symbols
ii) Empty productions
iii) unit productions.

Greibach Normal Form (GNF)

↳ a grammar $G = (V, \Sigma, R, S)$ is said to be in Greibach Normal form (GNF) if all the productions are of the form,

$$A \longrightarrow a\alpha$$

where a is terminal and α is string of zero or more variable.

Q) Convert following CFG into CNF.

$G = (V, \Sigma, R, S)$ where
$V = \{S, A, B\}$
$\Sigma = \{a, b\}$
$R = \{$
$S \longrightarrow aAB | AaB | B$
$A \longrightarrow aA | \epsilon$
$B \longrightarrow ab | bA$

Solution:

↳ NO useless symbols.
↳ Eliminate empty production
$A \longrightarrow \epsilon$.
for this:

$S \to aAB \Rightarrow S \to aB$
$S \to AaB \Rightarrow S \to aB$
$A \to aA \Rightarrow S \to a$
$B \to bA \Rightarrow B \to b.$

So updated grammar is,

$S \to aAB \mid AaB \mid B \mid aB$
$A \to aA \mid a$
$B \to ab \mid bA \mid b$

↳ Eliminate unit production $S \to B$.

∴ we get,

$S \to aAB \mid AaB \mid ab \mid bA \mid b \mid aB$
$A \to aA \mid a$
$B \to ab \mid bA \mid b$

Now,

$S \to V_1AB \mid AV_1B \mid V_1V_2 \mid V_2A \mid b \mid V_1B$
$V_1 \to a$
$V_2 \to b$
$A \to V_1A \mid a$
$B \to V_1V_2 \mid V_2A \mid b.$

finally,

$S \to Z_1B \mid Z_2B \mid V_1V_2 \mid V_2A \mid b \mid V_1B$
$Z_1 \to V_1A$
$Z_2 \to AV_1$
$V_1 \to a$
$V_2 \to b$
$A \to V_1A \mid a$
$B \to V_1V_2 \mid V_2A \mid b.$

This is in CNF.

Q) Convert following CFG into CNF.

$S \to AAC$
$A \to aAb \mid \epsilon$
$C \to aC \mid a$

↳ no useless symbols.
↳ no unit production
↳ Eliminating empty production as,

$S \to AAC \Rightarrow S \to C \mid AC$
$A \to aAb \Rightarrow A \to ab$

So, updated grammar is,

$S \to \epsilon \mid AC \mid AAC$
$A \to ab \mid aAb$
$C \to aC \mid a$

↳ Eliminate unit production $S \to C$.
we get,

$S \longrightarrow ac \mid a \mid AC \mid AAC$
$A \longrightarrow ab \mid aAb$
$C \longrightarrow ac \mid a$

Now,

$S \longrightarrow V_1C \mid a$
$V_1 \longrightarrow a$
$A \longrightarrow V_1AV_2$
$V_2 \longrightarrow b$
$C \longrightarrow V_1C \mid a$ ans

$S \longrightarrow V_1C \mid a \mid V_1C \mid V_1V_1C$
$V_1 \longrightarrow a$
$A \longrightarrow V_1b \mid V_1AV_2$
$V_2 \longrightarrow b$
$C \longrightarrow V_1C \mid a$

$S \longrightarrow V_1C \mid a \mid V_1C \mid V_1Z_1$
$Z_1 \longrightarrow AV_2 \mid V_1C$
$V_1 \longrightarrow a$
$A \longrightarrow V_1V_2 \mid V_1Z_2V_2$
$Z_2 \longrightarrow V_1A$
$V_2 \longrightarrow b$
$C \longrightarrow V_1C \mid a$

is the required CNF.

a) Convert following CFG into CNF.

$S \longrightarrow ABA \mid abA \mid BC$
$A \longrightarrow aA \mid \epsilon$
$B \longrightarrow baB \mid C$
$C \longrightarrow aC$

↳ No useless symbols.
↳ Eliminating Empty production $A \longrightarrow \epsilon$.

So, $S \longrightarrow ABA \Rightarrow S \longrightarrow AB \mid BA \mid B$
$S \longrightarrow abA \Rightarrow S \longrightarrow ab$
$A \longrightarrow aA \Rightarrow A \longrightarrow a$

So, updated grammar is,

$S \longrightarrow AB \mid BA \mid B \mid ab \mid BC$
$A \longrightarrow a$
$B \longrightarrow baB \mid \epsilon$
$C \longrightarrow aC$

↳ Eliminating unit production, $B \longrightarrow C$

$S \longrightarrow AB \mid BA \mid B \mid ab \mid BC$
$A \longrightarrow a$
$B \longrightarrow baB \mid aC$
$C \longrightarrow aC$

Now,

$S \longrightarrow ABA \mid abA \mid BC$
$A \rightarrow aA \mid G$
$B \rightarrow baB \mid C$
$C \rightarrow aC.$

↳ So, C, B are useless symbols as we can't generate terminals.

$S \longrightarrow abA$
$A \rightarrow aA \mid \epsilon$

↳ Eliminating Empty production. $A \rightarrow G$

$S \longrightarrow abA \Rightarrow S \rightarrow ab$
$A \longrightarrow aA \Rightarrow A \rightarrow a$

So updated grammar is,

$S \longrightarrow ab \mid abA$
$A \rightarrow aA \mid a$

↳ no unit production. So,

$S \longrightarrow V_1V_2 \mid V_1V_2A$
$V_1 \rightarrow a$
$V_2 \rightarrow b$
$A \rightarrow V_1A \mid a$

Again,

$S \longrightarrow V_1V_2 \mid Z_1A$
$Z_1 \longrightarrow V_1V_2$
$V_1 \rightarrow a$
$V_2 \rightarrow b$
$A \rightarrow V_1A \mid a$

ans#.

Q) Convert following CFG into GNF.

$S \longrightarrow AB \mid BC$
$A \rightarrow aB \mid bA \mid a$
$B \rightarrow bc \mid cC \mid a$

⇓

$S \longrightarrow aBB \mid bAB \mid aB \mid bCC \mid cCC \mid ac$
$A \rightarrow aB \mid bA \mid a$
$B \rightarrow bc \mid cC \mid a$
$C \rightarrow c$

Left Recursion:

↳ A production of grammar is said to have left recursion if the left most variable of its RHS is same as variable of L.H.S.

eg. $S \longrightarrow Sa \mid \epsilon$

α

Date
Page

Elimination of left recursion.

↳ left recursion is eliminated by converting the grammar into right recursive grammar.

↳ If we have,

$$A \longrightarrow A\alpha \mid \beta$$

then,

$$A \longrightarrow \beta A'$$
$$A' \longrightarrow \alpha A' \mid \epsilon$$

✓ eg: consider following and eliminate left recursion.

$$A \longrightarrow ABd \mid Aa \mid g$$
$$B \longrightarrow Be \mid b$$

⇓

$$A \longrightarrow gA'$$
$$A' \longrightarrow BdA' \mid aA' \mid \epsilon$$
$$B \longrightarrow bB'$$
$$B' \longrightarrow eB' \mid \epsilon$$

Q) Convert following CFG into GNF.

$$S \longrightarrow XA \mid BB$$
$$B \longrightarrow b \mid SB$$
$$X \longrightarrow b$$
$$A \longrightarrow a$$

Date
Page

Here,

$$S \longrightarrow bA \mid SBB \mid BSB \mid BB \mid bBt$$
$$B \longrightarrow b \mid XA \mid BBB$$
$$X \longrightarrow b$$
$$A \longrightarrow a$$

Now,

$$S \longrightarrow bA \mid BB$$
$$B \longrightarrow b \mid SB$$
$$X \longrightarrow b$$
$$A \longrightarrow a$$

then,

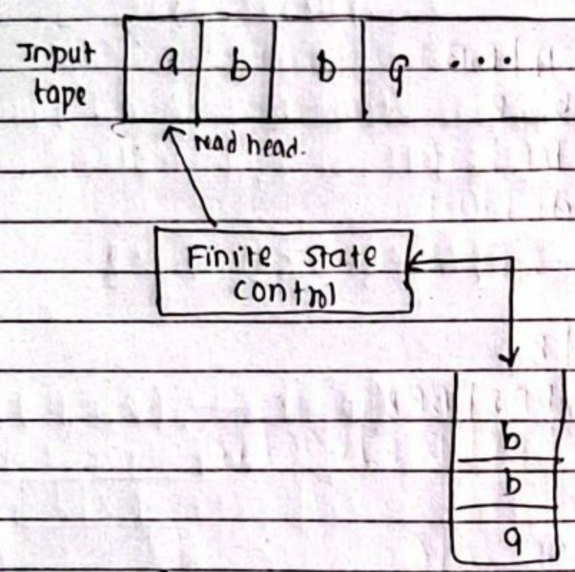
$$S \longrightarrow bA \mid BB$$
$$B \longrightarrow b \mid bAB \mid BBB$$
$$X \longrightarrow b$$
$$A \longrightarrow a$$

Now,

$$S \longrightarrow bA \mid BB$$
$$B \longrightarrow bB' \mid bABB'$$
$$B' \longrightarrow BBB' \mid \epsilon$$
$$X \longrightarrow b$$
$$A \longrightarrow a$$

Pushdown Automata (PDA):

↳ is automata that defines CFL.
↳ it can be thought as a ∈-NFA with addition of stack.
↳ so, it remembers infinite amount of information.
↳ but, it can only access information in stack as last in first out.



[Fig: block diagram of PDA]

↳ PDA is abstract machine determined by following three things.
  i) input
  ii) finite state control
  iii) stack.

↳ Each move of machine is determined by three things:
  i) current state
  ii) Input symbol
  iii) symbol on top of stack.

↳ The move consists of,
  i) changing state / stay on same state.
  ii) replace stack top by string of zero or more symbols.

↳ move of machine contains along one stack operation either push or pop.

↳ popping the top symbol of the stack means replacing it by ∈.

↳ pushing γ on ~~both~~ the stack means replacing stack top say X by γ.

↳ A PDA can write symbols on the stack and read them back later.

Formal defination:

↳ A PDA is defined by seven tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where,
  $Q \rightarrow$ finite set of states
  $\Sigma \rightarrow$ finite set of symbols
  $\Gamma \rightarrow$ finite set of stack symbols.

$q_0 \rightarrow$ start state ; $q_0 \in Q$

$Z_0 \rightarrow$ initial stack symbol ; $Z_0 \in T$

$F \rightarrow$ set of final states ; $F \subseteq Q$

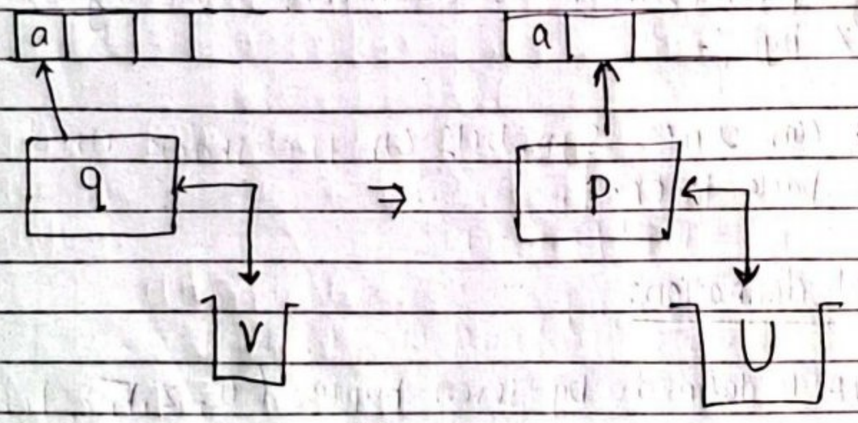
$\delta \rightarrow$ transition function that takes an argument a triplet (q, a, x) where q is a state, a is either input symbol in Σ or ϵ.

x is stack symbol.

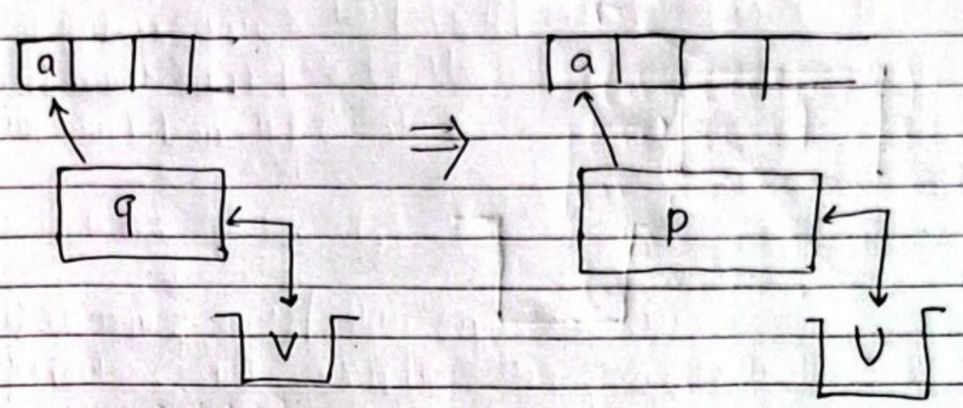
4 output of δ is a finite set of pair (p, γ) where p is new state and γ is string on state after transition.

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

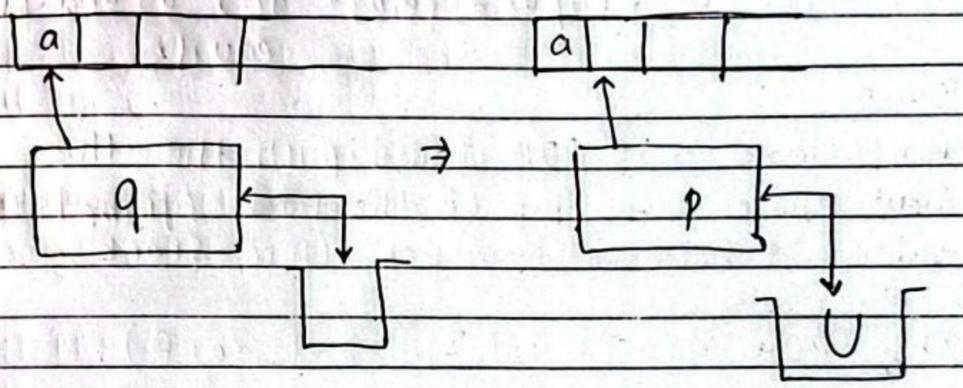
i) $\delta(q, a, v) \rightarrow (p, u)$



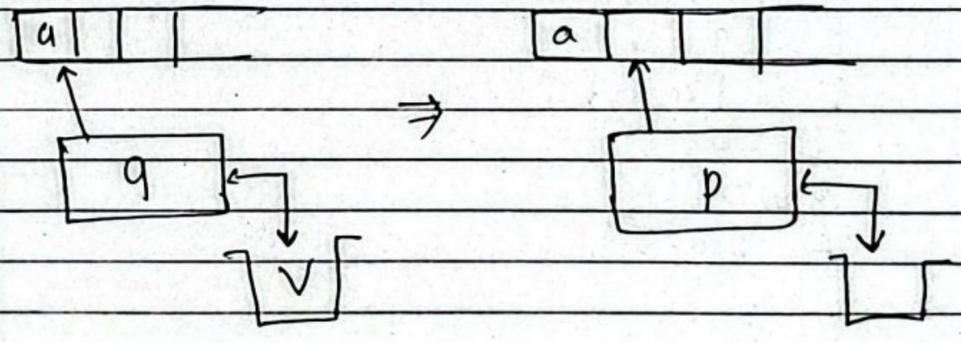
ii) $\delta(q, \epsilon, v) \rightarrow (p, u)$

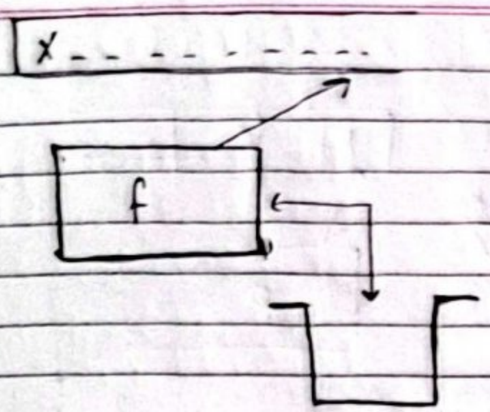


iii) $\delta(q, a, \epsilon) \rightarrow (p, u)$



iv) $\delta(q, a, v) \rightarrow (p, \epsilon)$



v) 

Q) Design a PDA accepting string over {a,b} such that number of a's and b's are equal.

i.e. $L = \{w \mid w \in \{a,b\}^*$ and a's and b's are equal.

Solution:

↳ Here, idea is that PDA should push the the input symbol if the top of the stack symbol is same as it or z_0 else pop from stack.

let, PDA be $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$.

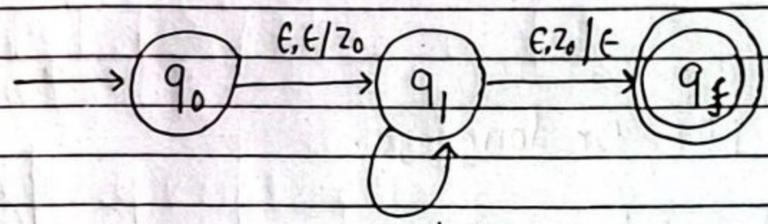
where,

$Q = \{q_0, q_1, q_f\}$
$\Sigma = \{a, b\}$
$\Gamma = \{z_0, a, b\}$
$q_0 = q_0$
$z_0 = z_0$
$F = \{q_f\}$

δ:

i) $\delta(q_0, \epsilon, \epsilon) = (q_1, z_0)$
ii) $\delta(q_1, a, z_0) = (q_1, az_0)$
iii) $\delta(q_1, b, z_0) = (q_1, bz_0)$
iv) $\delta(q_1, a, a) = (q_1, aa)$
v) $\delta(q_1, b, a) = (q_1, \epsilon)$
vi) $\delta(q_1, a, b) = (q_1, \epsilon)$
vii) $\delta(q_1, b, b) = (q_1, bb)$.
viii) $\delta(q_1, \epsilon, z_0) = (q_f, \epsilon)$

{ // suru ma a pani aauna sakxa b pani aauna sakxa.



check strings

bbaa

$\delta(q_0, bbaa, \epsilon)$
$\vdash \delta(q_1, bbaa, z_0)$
$\vdash \delta(q_1, baa, bz_0)$
$\vdash \delta(q_1, aa, bbz_0)$
$\vdash \delta(q_1, a, bz_0)$
$\vdash \delta(q_1, \epsilon, z_0)$
reject.

$\delta(q_0, abaabb, \epsilon)$

$\vdash \delta(q_1, abaabb, z_0)$
$\vdash \delta(q_1, baabb, az_0)$
$\vdash \delta(q_1, aabb, z_0)$
$\vdash \delta(q_1, abb, az_0)$
$\vdash \delta(q_1, bb, aaz_0)$

$\vdash \delta(q_1, b, az_0)$
$\vdash \delta(q_1, \epsilon, z_0)$

$\vdash \delta(q_f, \epsilon, \epsilon)$.

Accept.

Q) Design a PDA for language.

$L = \{0^n 1^n \mid n \geq 0\}$
$L = \{\epsilon, 01, 0011, 000111, \ldots\}$

Let the PDA be $P = \{Q, \Sigma, \Gamma, \delta, q_0, z_0, f\}$
where,

$Q = \{$
$\Sigma = \{0, 1\}$
$\Gamma = \{$
$q_0 =$
$z_0 =$
$f = \{$

δ:

i) $\delta(q_0, \epsilon, \epsilon) = (q_1, z_0)$
ii) $\delta(q_1, 0, z_0) = (q_1, 0z_0)$
iii) $\delta(q_1, 0, 0) = (q_1, 00)$
iv) $\delta(q_1, 1, 0) = (q_2, \epsilon)$
v) $\delta(q_2, 1, 0) = (q_2, \epsilon)$
vi) $\delta(q_2, \epsilon, z_0) = (q_f, \epsilon)$.
vii) $\delta(q_1, \epsilon, z_0) = (q_f, \epsilon)$

Soln,

Here,

001

$\delta(q_0, 001, \epsilon)$
$\vdash \delta(q_1, 001, z_0)$
$\vdash \delta(q_1, 01, 0z_0)$
$\vdash \delta(q_1, 1, 00z_0)$
$\vdash \delta(q_1, \epsilon, 0z_0)$.

0101

$\delta(q_0, 0101, \epsilon)$
$\vdash \delta(q_1, 0101, z_0)$
$\vdash \delta(q_1, 101, 0z_0)$
$\vdash \delta(q_1, 01, 00z_0)$
$\vdash \delta(q_1, 1, 0z_0)$
$\vdash \delta(q_1, 1, 0z_0)$.
$\vdash \delta(q_1, \epsilon, z_0)$
$\vdash \delta(q_f, \epsilon, \epsilon)$.

Accept

Problem

Q) Design a PDA for language,

$$L = \{a^n b^{2n} \quad n \geq 0\}$$

Here,

$L = \{\epsilon, abb, aabbbb, aaabbbbbb, \ldots\}$

i) $\delta(q_0, \epsilon, \epsilon) = \delta(q_1, Z_0)$

ii) $\delta(q_1, a, Z_0) = \delta(q_1, aZ_0)$

iii)

iv) $\delta(q_1, a, a) = \delta(q_1, aa)$

v) $\delta(q_1, b, a) = (q_2, \epsilon)$

vi) $\delta(q_2, b, a) = (q_1, \epsilon)$

vii) $\delta(q_1, \epsilon, Z_0) = (q_f, \epsilon)$.

Check for strings:

1) aabbbb

$\delta(q_0, aabbbb, \epsilon)$
$\delta(q_0, aabbbb, Z_0)$
$\delta(q_0, abbbb, aZ_0)$
$\delta(q_2, bbbb, aaZ_0)$
$\delta(q_1, bbb, aZ_0)$
$\delta(q_2, bb, aZ_0)$
$\delta(q_1, b, Z_0)$

$\rightarrow \delta(q_f, \epsilon, Z_0)$
$\vdash \delta(q_f, \epsilon, \epsilon)$.

i) aaabbbb

$\delta(q_0, aaabbbb, \epsilon)$
$\vdash \delta(q_0, aaabbbb, Z_0)$
$\vdash \delta(q_1, aabbbb, aZ_0)$
$\vdash \delta(q_1, abbbb, aaZ_0)$
$\vdash \delta(q_2, bbbb, aaaZ_0)$

$\vdash \delta(q_1, bbb, aaZ_0)$

$\vdash \delta(q_2, bb, aZ_0)$
$\vdash \delta(q_1, b, Z_0)$

$\vdash \delta(q_f, \epsilon, Z_0)$.

accepted.

ii) aaabbb.

$\delta(q_0, aaabbb, \epsilon)$
$\vdash \delta(q_0, aaabbb, Z_0)$
$\vdash \delta(q_1, aabbb, aZ_0)$
$\vdash \delta(q_1, abbb, aaZ_0)$
$\vdash \delta(q_2, bbb, aaaZ_0)$
$\vdash \delta(q_2, bb, aaZ_0)$
$\vdash \delta(q_1, b, aZ_0)$
$\vdash \delta(q_2, \epsilon, aZ_0)$

rejected.

Date
Page

a) Design a PDA for language:

$L = \{ a^i b^j c^k : k = i+j \}$

$L = \{ \epsilon, ac, bc, abcc, aacc, bbcc, abbbccccc, \cdots \}$

δ:

$\delta(q_0, \epsilon, \epsilon) = \delta(q_1, z_0)$
$\delta(q_1, a, z_0) = \delta(q_1, az_0)$
$\delta(q_1, b, z_0) = \delta(q_1, bz_0)$
$\delta(q_1, a, a) = \delta(q_1, aa)$
$\delta(q_1, b, a) = \delta(q_1, ba)$
$\delta(q_1, c, a) = \delta(q_1, \epsilon)$

} a ani huda input e, b, a aauna sakn second string.

$\delta(q_1, b, b) = (q_1, bb)$
$\delta(q_1, c, b) = (q_1, \epsilon)$

$\delta(q_1, a, c) = (q_2,$
$\delta(q_1, b, c) = (q_2,$

$\delta(q_1, \epsilon, z_0) = (q_f, \epsilon)$.

acbc
accept vaza.

Q) Design a PDA for language,

$L = \{wcw^R : w \in \{a,b\}^*\}$

$L = \{c, aca, bcb, babcbab, bbacabb, abbcbba, aabcbaa, \ldots\}$

δ:

$\delta(q_0, \epsilon, \epsilon) = (q_1, Z_0)$
$\delta(q_0, a, Z_0) = (q_1, aZ_0)$
$\delta(q_0, b, Z_0) = (q_1, bZ_0)$
$\delta(q_0, a, a) = (q_1, aa)$
$\delta(q_0, b, a) = (q_1, ba)$
$\delta(q_0, a, b) = (q_1, ab)$
$\delta(q_0, b, b) = (q_1, bb)$
$\delta(q_0, c, a) = (q_2, a)$
$\delta(q_0, c, b) = (q_2, b)$
$\delta(q_2, a, a) = (q_2, \epsilon)$
~~$\delta(q_2, b, a) = (q_2, a)$~~
~~$\delta(q_2, a, b) = (q_2, b)$~~
$\delta(q_2, b, b) = (q_2, \epsilon)$
$\delta(q_1, c, Z_0) = (q_2, Z_0)$
$\delta(q_1, \epsilon, Z_0) = (q_f, \epsilon)$

① abbcbba

$\delta(q_0, abbcbba, \epsilon) = \delta(q_1, Z_0)$
$\vdash \delta(q_1, abbcbba, Z_0) = \delta(q_1, aZ_0)$
$\vdash \delta(q_1, bbcbba, a) = \delta(q_1, baZ_0)$
$\vdash \delta(q_1, bcbba, ba) = \delta(q_1, bbaZ_0)$
$\vdash \delta(q_1, bba, bba) = \delta(q_2, bbaZ_0)$
$\vdash \delta(q_1, ba, bba) = \delta(q_1, bbaZ_0)$
$\vdash \delta(q_1, a, bba) = \delta(q_1, bbaZ_0)$
$\vdash \delta(q_1, \epsilon, bba) = \delta($

① abcab

Non Deterministic PDA is more powerful than Deterministic PDA.

Date
Page

Q) Design a PDA for,

$L = \{ ww^R : w \in \{a,b\}^* \}$

Here,

$L = \{ \epsilon, aa, bb, abba, baab, babbab, \ldots \}$

δ:

(i) $\delta(q_0, \epsilon, \epsilon) = (q_1, Z_0)$

(ii) $\delta(q_1, a, Z_0) = (q_1, aZ_0)$

(iii) $\delta(q_1, b, Z_0) = (q_1, bZ_0)$

(iv) $\delta(q_1, a, a) = (q_1, aa)$

(v) $\delta(q_1, b, a) = (q_1, ba)$

(vi) $\delta(q_1, b, b) = (q_1, bb)$

(vii) $\delta(q_1, a, a) = (q_{12}, \epsilon)$

(viii) $\delta(q_1, b, b) = (q_2, \epsilon)$

(ix) $\delta(q_1, a, b) = (q_1, ab)$

(x) $\delta(q_2, a, a) = (q_2, \epsilon)$

(xi) $\delta(q_2, b, b) = (q_2, \epsilon)$

(xii) $\delta(q_2, \epsilon, Z_0) = (q_f, \epsilon)$.

① babbab

$\Rightarrow \delta(q_0, babbab, \epsilon)$

$\Rightarrow \vdash \delta(q_1, babbab, Z_0)$

$\Rightarrow \vdash \delta(q_1, abbab, bZ_0)$

$\Rightarrow \vdash \delta(q_1, bbab, abZ_0)$

$\Rightarrow \vdash \delta(q_1, bab, babZ_0)$

$\vdash \delta(q_2, ab, abZ_0)$

$\vdash \delta(q_2, b, bZ_0)$

$\vdash \delta(q_2, \epsilon, Z_0)$

$\vdash \delta(q_f, \epsilon, \epsilon)$

accepted.

Instantaneous Description of PDA (ID of PDA)

↳ Description of any PDA by a triplet (q, w, γ) where q is the state, w is the remaining input and γ is the stack contents is called Instantaneous description of PDA.

↳ Notation of ID for PDA is used as to describe changes in state, input and stack.

↳ Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA.

↳ Define a relation $\vdash$.

↳ Suppose $\delta(q, a, x)$ contains (p, α). Then for all string w in Σ^* and β in Γ^*.

$$(q, aw, x\beta) \vdash (p, w, \alpha\beta)$$

↳ What remains on the input w and what is below top of stack β do not influence the action of PDA.

↳ We can also use symbol $\vdash^*$ when PDA is understand to represent zero or more moves of PDA

eg: $(q_0, aabbbaab, z_0) \vdash^* (q_2, \epsilon, \epsilon)$

↳ We can define acceptance of any string by PDA in two ways:

(i) acceptance by final state:

↳ Given PDA P, the language accepted by final state $L(P)$ is

$$\{w \mid (q, w, z_0) \vdash^* (p, \epsilon, \gamma)\} \text{ where } p \in F, \gamma \in T^*$$

(ii) acceptance by empty stack:

↳ Given PDA P, the language accepted by empty stack $L(P)$ is,

$$\{w \mid (q, w, z_0) \vdash^* (p, \epsilon, \epsilon)\} \text{ where } p \in Q.$$

Equivalence of PDA and CFG

↳ To show the languages defined by PDA are exactly CFL.

↳ Main goal is to prove that CFG and PDA accept same class of language.

(i) From CFG to PDA:

↳ Given CFG, $G = (V, T, P, S)$ we construct a PDA M that accepts language generated by G.

i.e. $L(M) = L(G)$

↳ PDA can be defined as,

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

where
$Q = \{q\}$
$\Sigma = T$
$\Gamma = V \cup T$
$Z_0 = S$
$F = \phi$
$q_0 = q$

δ can be:
(i) $\delta(q, \epsilon, A) = (q, \alpha)$ for $A \to \alpha$ in G
(ii) $\delta(q, a, a) = (q, \epsilon)$
Here transition occurs in only one state q.

Alternatively:
↳ Defines two states p and q.
↳ p is initial state
↳ let initially state symbol is ϵ.
↳ PDA starts with the p, read ϵ and inserts 'S' into stack and then changes to state q.
↳ Then all transition occurs in q.
↳ Now, PDA can be defined as $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$
where
$Q = \{p, q\}$
$\Sigma = T$
$\Gamma = V \cup T$
$q_0 = p$
$Z_0 = \epsilon$
$F = \phi$

δ:
(i) $\delta(p, \epsilon, \epsilon) = (q, S)$
(ii) $\delta(q, \epsilon, A) = (q, \alpha)$ for all $A \to \alpha$ in G
(iii) $\delta(q, a, a) = (q, \epsilon)$

Example: Let $G = (V, T, P, S)$
where P is:
$E \to T \mid E + T$
$T \to F \mid T * F$
$F \to a \mid (E)$

Ans: Here,
Equivalent PDA is $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$
where
$Q = \{q\}$
$\Sigma = \{q, +, *, (,)\}$
$\Gamma = \{E, F, T, a, +, *, (,)\}$
$q_0 = q$
$Z_0 = E$
$F = \phi$

δ:-
(i) $\delta(q, \epsilon, E) = \{(q, T), (q, E + T)\}$
(ii) $\delta(q, \epsilon, T) = \{(q, F), (q, T * F)\}$
(iii) $\delta(q, \epsilon, F) = \{(q, (E)), (q, a)\}$
(iv) $\delta(q, a, a) = (q, \epsilon)$
(v) $\delta(q, +, +) = (q, \epsilon)$
(vi) $\delta(q, *, *) = (q, \epsilon)$
(vii) $\delta(q, (, () = (q, \epsilon)$
(viii) $\delta(q,),)) = (q, \epsilon)$

a+(a*a)

(1) a+(a*a)	δ(q0, a+(a*a), E)
E ⟹ E+T	⊢δ(q0, a+(a*a), E+T)
⇒ E+F	⊢δ(q0, a+(a*a), T+T)
⇒ F+(E)	⊢δ(q0, a+(a*a), F+T)
⇒ T+(E)	⊢δ(q0, a+(a*a), a+T)
⇒ T+(T)	⊢δ(q0, +(a*a), +T)
⇒ a+(T*F)	⊢δ(q0, (a*a), T)
⇒ a+(T*a)	⊢δ(q0, (a*a), E)
⇒ a+(F*a)	⊢δ(q0, (a*a), (E))
⇒ a+(a*a)	⊢δ(q0, a*a), E))
	⊢δ(q0, a*a), T))
	⊢δ(q0, a*a), T*F))

⊢δ(q0, a*a), F*F)
⊢δ(q0, a*a, a*F)
⊢δ(q0, *a), *F)
⊢δ(q0, a), F)
⊢δ(q0, a), a)
⊢δ(q0,),))
⊢δ(q0, ε, ε).

From PDA to CFG:

↳ Let $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ be a PDA that accepts a language L by empty stack. Then a CFG, $G_1 = (V, \Sigma, R, S)$ that generates L can be constructed using following rules:

i) assume that special symbol z_0 is in stack.

ii) Let S be start symbol of grammar.

iii) For every $q \in Q$, add a production

$$S \rightarrow [q_0\ z_0\ q]$$ in P.

iv) For every $q, r \in Q$, $a \in \{\Sigma \cup \epsilon\}$, $x = \Gamma$, if $\delta(q, 0, x) = (r, \epsilon)$ then add a production

$$[q\ x\ r] \rightarrow a.$$

v) For every $q, r \in Q$, $a \in \{\Sigma \cup \epsilon\}$, $x \in T$ and $x \geq 1$, if $\delta(q, a, x) = (r, x_1 x_2 \cdots x_k)$ where $x_1 x_2 \cdots x_k \in T$, then for every choice of $q_1, q_2, \ldots q_k \in Q$.

add the production $[q\ x\ q_k] \rightarrow a\,[r\ x_1\ q_1]\,[q_1\ x_2\ q_2] \cdots [q_{k-1}\ x_k\ q_k]$.

eg. Consider a PDA with following transition functions,

i) $\delta(q_1, a, z_0) = (q_1, a z_0)$
ii) $\delta(q_1, a, a) = (q_2, a)$.
iii) $\delta(q_2, a, a) = (q_3, a)$
iv) $\delta(q_3, a, a) = (q_1, \epsilon)$
v) $\delta(q_1, \epsilon, z_0) = (q_2, \epsilon)$

Solution:

$Q = \{q_1, q_2, q_3\}$
$\Sigma = \{a\}$
$\Gamma = \{a, z_0\}$
$q_0 = q_1$
$z_0 = z_0$
$F = \phi$

SO, i) $S \rightarrow [q_1\ z_0\ q_1]$
ii) $S \rightarrow [q_1\ z_0\ q_2]$
iii) $S \rightarrow [q_1\ z_0\ q_3]$

iv) $[q_3 \cdot a\ q_3] \longrightarrow a$

v) $[q_1\ z_0\ q_1] \rightarrow \epsilon$

vi) $[q_1\ z_0\ q_1] \rightarrow a[q_1\ a\ q_1][q_1\ z_0\ q_1]$
$[q_1\ z_0\ q_1] \rightarrow a[q_1\ a\ q_2][q_2\ z_0\ q_1]$
$[q_1\ z_0\ q_1] \rightarrow a[q_1\ a\ q_3][q_3\ z_0\ q_1]$

$[q_1\ z_0\ q_2] \rightarrow a[q_1\ a\ q_1][q_1\ z_0\ q_2]$
$[q_1\ z_0\ q_2] \rightarrow a[q_1\ a\ q_2][q_2\ z_0\ q_2]$
$[q_1\ z_0\ q_2] \rightarrow a[q_1\ a\ q_3][q_3\ z_0\ q_2]$

$[q_1\ z_0\ q_3] \rightarrow a[q_1\ a\ q_1][q_1\ z_0\ q_3]$
$[q_1\ z_0\ q_3] \rightarrow a[q_1\ a\ q_2][q_2\ z_0\ q_3]$
$[q_1\ z_0\ q_3] \rightarrow a[q_1\ a\ q_3][q_3\ z_0\ q_3]$

→ yo last vayera

vii) $[q_1\ a\ q_1] \longrightarrow a[q_2\ a\ q_1]$
$[q_1\ a\ q_2] \longrightarrow a[q_2\ a\ q_2]$
$[q_1\ a\ q_3] \longrightarrow a[q_2\ a\ q_3]$

viii)
$$[q_2, a, q_1] \longrightarrow a\begin{bmatrix} q_3 & q & q_1 \end{bmatrix}$$
$$[q_2, a, q_2] \longrightarrow a\begin{bmatrix} q_3 & q & q_2 \end{bmatrix}$$
$$[q_2, a, q_3] \longrightarrow a\begin{bmatrix} q_3 & a & q_3 \end{bmatrix}$$

Let's assume:

PDA: aaaa

$\delta(q_1, aaaa, z_0)$
$\vdash \delta(q_1, aaa, az_0)$
$\vdash \delta(q_2, aa, az_0)$
$\vdash \delta(q_3, a, az_0)$
$\vdash \delta(q_1, \epsilon, z_0)$
$\vdash \delta(q_1, \epsilon, \epsilon)$

Accepted by empty stack. ~~Pushdown~~.

CFG:

$S \Rightarrow [q_1\ z_0\ q_1]$
$\Rightarrow a[q_1\ a\ q_1][q_1\ z_0\ q_1]$
$\Rightarrow aa[q_2\ a\ q_1]\epsilon$
$\Rightarrow aaa[q_3\ a\ q_1]$
$\Rightarrow aaaa.$

generated as well.

Pumping lemma for CFL:

↳ It says that in any sufficiently long string in a CFL, it is possible to find at most two short near by substrings that we can pump in tandem (one behind another) i.e we may repeat both of strings for i times and the resulting string will be still in the language.

↳ Here string is divided into five parts so that its second and fourth parts may be repeated together any no. of time and resulting string still remains in language.

↳ It is used to show that language is not a CFL.

Statement:

↳ Let L be a CFL and w be any string in L.
↳ Let n be a positive integer or pumping constant with $|w| \geq n$.
↳ Then we can write $w = uvxyz$ such that

i) $|vxy| \leq n$
ii) $|vy| > 0$
iii) for all $r \geq 0$, $uv^ixy^iz \in L$.

i.e. two strings v and y can be pumped any no. of times and the resulting string will still be in L.

Example:

Show that, $L = \{a^n b^n c^n : n \geq 1\}$ is not CFL using pumping lemma.

Soln: let L be a CFL and w be any string in L; $w \in L$. m be any pumping constant with $|w| \geq m$. then, w can be written as ~~$w = uvxyz = a^n b^n c^n$~~

$w = uvxyz = a^n b^n c^n$.

Such that,

$|vxy| \leq n$ & $|uy| > 0$

and must satisfy $uv^ixy^iz \in L$ for all $i \geq 0$.

Case-I: vxy contains both a's and b's

Now, $w = uvxyz$

$= \underbrace{aa\cdots}_{u}\underbrace{aa\cdots ab\cdots bb}_{vxy}\underbrace{\cdots bcc\cdots c}_{z}$

let, $v = a^p$, $x = a^q$, $y = b^r$ $p+r \geq 1$.

Then for, $i = 2$,

$w = uv^2xy^2z$

$= a^{n+p} b^{n+r} c^n \notin L$.

i.e. it contradicts our assumption.

Case II : v_{xy} contain both b's and c's

Here, $w = uvxyz$

$= \underbrace{aa\cdots aq\cdots}_{u}\underbrace{bb\cdots bC\cdots CC\cdots C}_{vxy}\underbrace{\quad}_{z}$

let, $v = b^p$, $x = b^q$, $y = c^r$, $p+r \geq 1$.

then, for $i=2$,

$$uv^2xy^2z = a^n b^{n+p} c^{n+p} \notin L$$

i.e. it contradicts our assumption.

Case III : v_{xy} is in a's

Here,

$w = uvxyz = \underbrace{a\cdots aq}_{u}\underbrace{q\cdots a}_{vxy}\underbrace{abb\ bc\cdots c}_{z}$

let, $v = a^p$, $y = a^r$; $p+r \geq 1$

for $i=2$,

$$uv^2xy^2z = a^{n+p+r} b^n c^n$$

i.e. it contradicts our assumption.

Hence, given language $L = \{a^n b^n c^n : n \geq 1\}$ is not a CFL.

Closure properties of CFL:

↳ CFL are closed under,

(i) Union (ii) Concatenation (iii) Kleene closure.

↳ let $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$ be context free grammar.

↳ let us assume, that they have disjoint set of non-terminals.

(i) Union:

↳ Let S be new symbol not in G_1 and G_2.

↳ Construct a new grammar $G = (V, \Sigma, R, S)$ where

$V = V_1 \cup V_2 \cup \{S\}$

$\Sigma = \Sigma_1 \cup \Sigma_2$

$R = R_1 \cup R_2 \cup \{S \to S1, S \to S2\}$

↳ Here G is clearly a CFG because two rules added are also of correct form.

↳ Now, we claim.

$$L(G) = L(G_1) \cup L(G_2)$$

∴ CFL's are closed under union.

(ii) Concatenation:

↳ construct a new grammar $G = (V, \Sigma, R, S)$ where,

$V = V_1 \cup V_2 \cup \{S\}$

$\Sigma = \Sigma_1 \cup \Sigma_2$

$R = R_1 \cup R_2 \cup \{ S \rightarrow S_1 S_2 \}$

Here,

if $S_1 \overset{*}{\underset{G_1}{\Rightarrow}} w_1$ and $S_2 \overset{*}{\underset{G_2}{\Rightarrow}} w_2$

we can claim,

$S \overset{*}{\underset{G}{\Rightarrow}} w_1 w_2$

$\therefore S \longrightarrow S_1 S_2$

~~Then G,~~ Hence,

$L(G) = L(G_1) \cdot L(G_2)$

$\therefore$ CFL's are closed under concatenation.

(iii) Kleene closure:

↳ Construct a new grammar $G = (V, \Sigma, R, S)$ where,

$V = V_1 \cup \{S\}$

$\Sigma = \Sigma_1$

$R = R_1 \cup \{ S \rightarrow SS_1 , S \rightarrow \epsilon \}$

↳ Here,

if $S_1 \overset{*}{\underset{G_1}{\Rightarrow}} w_1$ then $S \overset{*}{\underset{G}{\Rightarrow}} w_1 *$

$\therefore S \rightarrow \epsilon$ & $S \rightarrow SS_1$.

↳ Hence, $L(G) = L(G_1)^*$

$\therefore$ CFL is closed under Kleen closure.

CFL are not closed under Intersection:

↳ Define two CFL L_1 and L_2 as;

$L_1 = \{ a^n b^n c^m : n \geq 1, m \geq 1 \}$

$L_2 = \{ a^m b^n c^n : n \geq 1, m \geq 1 \}$

Here,

grammar for L_1 is	grammar for L_2 is,
$S \rightarrow AB$	$S \rightarrow AB$
$A \rightarrow aAb \mid ab$	$A \rightarrow aA \mid a$
$B \rightarrow cb \mid c$	$B \rightarrow bBc \mid bc$

Now, the language $L = L_1 \cap L_2$

$= \{ a^n b^n c^n : n \geq 1 \}$

is not a CFL.

because,

- L_1 requires equal no. of a and b.
- L_2 requires equal no. of b and c.
- a string in both must have equal no. of all three symbols and thus to be in L.
- But as L is not a CFL, we say that $L_1 \cap L_2$ are

Date
Page

not closed under intersection.

CFL are not closed closed under complementation:

↳ let CFL's are closed under complementation.
↳ let L_1 and L_2 are two CFL's so $\overline{L_1}$ and $\overline{L_2}$ are also CFL.

↳ As we know, CFL's are closed under union.

$\therefore \overline{L_1} \cup \overline{L_2}$ is a CFL.

↳ From De-Morgan's law,

$\overline{L_1} \cup \overline{L_2} = L_1 \cap L_2$

↳ This ~~contra~~ contradics the theorem that CFL's are closed under intersection.

↳ Hence, CFL's are not closed under complementation.

Chapter- 4

Turing Machine

Date
Page

↳ Turing Machine (TM) is an abstract machine developed by english mathematician Alan Turing in 1936.

↳ is the theoritical foundation of modern computer.

↳ TM will have:
  i) finite sets of alphabets
  ii) finite sets of state
  iii) linear tape which is potentially infinite in both end.

↳ Each square or cell of tape can hold one symbol from alphabet

↳ If no symbol than it contain blank .

↳ Reading and writing is done by tape head.

↳ Tape serve as:
  i) input device
  ii) Memory for computation
  iii) Output device.

↳ TM is a much more accurate model of general purpose computer.

↳ It can do everything that a real computer can do

↳ It consists of following:

(i) There are k tapes, for some fixed $k \geq 1$. Each tape is divided into cells and is infinite both to left and to the right.

Each cell stores a symbol belonging to a finite set of tape alphabet. The tape alphabet contains blank symbol, □. If cell contains □, it means, cell is actually empty.

(ii) Each tape has a tape head which can move along the tape, one cell per move. It can also read the cell it currently scans and replace the symbol in the cell by another symbol.

(iii) There is a state control, which can be in any one of finite no. of states. ~~Q contain three sp~~ The finite set of states Q contain three special states: start state, accept state, reject state.

↳ TM performs a sequence of computation states:

In one step it does following:

i) (a) :- immediately before the competution step, TM is in a state r of Q and each of k-tape heads is on certain cell.

(b) :- depending on the current state r and k symbol that are read by the tape head.

↳ TM switches to ~~save~~ state 'r' of Q (which may be equal to r)

↳ each tape head writes a symbol of tape alphabet in the cell it is currently scanning.

↳ Each tape head either move on one cell to left or to right or stays of current cell.

Formal definatron:

↳ TM is a 7 tuple $M = \{Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}\}$

'or'

$= \{Q, \Sigma, \Gamma, \delta, q_0, B, F\}$

Where,

Q → is a finite set of states

Σ → is a finite set of symbols : $\square \notin \Gamma$.

Γ → is a finite set of tape symbols: $\square \in \Gamma$, $\Sigma \subseteq \Gamma$.

q_0 → is a start state

q_{accept} → accept state

q_{reject} → reject state

B → blank symbol

F → set of final states.

δ is a trancition function where,

$Q \times \Gamma^k \rightarrow Q_k \times \Gamma^k \times \{L, R, N\}^k$.

↳ Let $r \in Q$, $q_1, q_2, \dots q_k \in \Gamma$
$r' \in Q$; $q_1', q_2', \dots q_k' \in \Gamma$
and $\delta_1, \delta_2, \delta_3, \dots \delta_k \in \{L, R, N\}$

Then,

$$\delta(r, q_1, q_2, \dots q_k) = (r', q_1', q_2', \dots q_k', \delta_1, \delta_2, \dots \delta_k)$$

(direction)

This means TM is in state r, the head of ith tape reads the symbol q_i; $1 \le i \le k$. then, the head of ith tape replaces the scanned symbol q_i by q_i'.

↳ if $\delta_i = L$, then tape head moves one cell to left

$\delta_i = R$, then tape head ~~doesn't~~ moves one cell to right

$\delta_i = N$, then tape head doesn't move.

↳ The computation terminates at the moment when TM enters the q_{accept} or q_{reject} state.

↳ TM accepts input string if the computation terminates in q_{accept}.

↳ TM ~~reject~~ rejects input string if the computation terminates in q_{reject}.

↳ L(M) is the language accepted by TM.

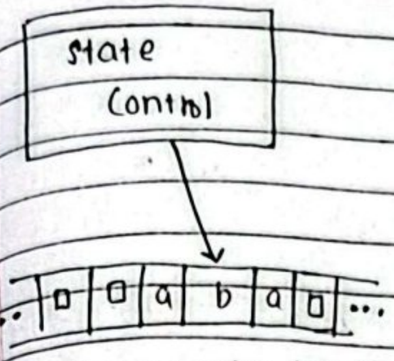


Fig: TM with k=1 tape

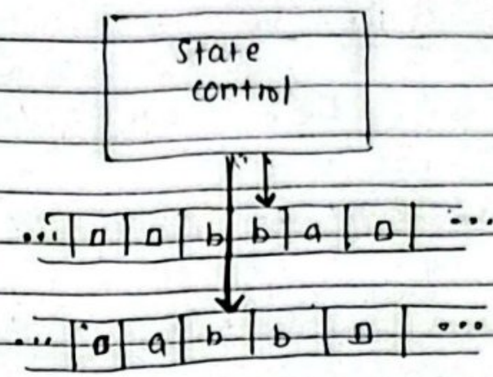


Fig: TM with k=2 tapes.

Instantaneous Description of TM:

↳ A string $x_1, x_2, \dots x_{i-1} q x_i x_{i+1} \dots x_n$ represents instantaneous description of TM where q is a state of TM and Tape head is scanning the ith symbol from left.

↳ $x_1 \dots x_n$ is the portion of tape between leftmost and rightmost ~~tape~~ blanks.

Moves of TM:

① for $\delta(q, x_i) = (p, Y, L)$ i.e next move is leftward.

Then, $x_1 x_2 \dots x_{i-1} q x_i x_{i+1} \dots x_n \vdash_M$

ID: $x_1 x_2 \dots x_{i-2} p x_{i-1} Y x_{i+1} \dots x_n$

for i=1,

$q x_1 x_2 \cdots x_n \vdash_M PBY x_2 \cdots x_n$ (blank)

for i=n, Y=B, (x1 to xn matching)

$x_1 x_2 \cdots x_{n-1} q x_n \vdash_M x_1 x_2 \cdots x_{n-2} P x_{n-1} B$

(ii) For $\delta(q, x_i) = (P, Y, R)$ i.e next move is rightward.

$x_1 x_2 \cdots x_{i-1} q x_i \cdots x_n \vdash$

$x_1 x_2 \cdots x_{i-1} Y p x_{i+1} \cdots x_n$

for i=n,

$x_1 x_2 \cdots x_{n-1} q x_n \vdash x_1 x_2 \cdots x_{n-1} Y P B$

for i=1, Y=B.

$q x_1 x_2 \cdots x_n \vdash p x_2 \cdots x_n$.

Q) Design a TM accepting $L = \{0^n 1^n : n \geq 1\}$

Solution:

- given Finite sequence of 0's and 1's on its tape preceded and followed by blanks.
- TM will change 0 to x and then 1 to Y until all 0's and 1's are matched. ~~starting~~
- Starting at left end of input, it repeatedly changes 0's to X and moves over whether 0's and Y's, it sees until comes to a 1.
- it changes 1 to Y and moves left over Y's and 0's until it finds an X.
- At that point it looks for 0 immediate to the right.
- If it finds one 0 then it changes to X and repeats process changing a matching 1 to Y.

Now,

let TM be $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

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

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

$\Gamma = \{0, 1, X, Y, B\}$

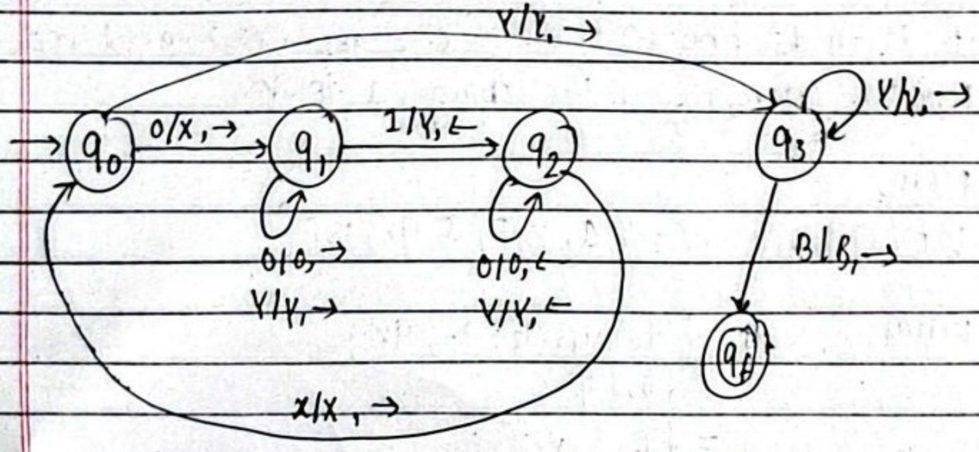
$q_0 = q_0$

B = B

F = { q_f }

δ:

Q/Σ	0	1	X	Y	B
q_0	(q_1, X, R)			(q_3, Y, R)	
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	
q_3				(q_3, Y, R)	(q_f, B, N)



0011

$\vdash q_0 0011$
$\vdash X q_1 011$
$\vdash X 0 q_1 11$
$\vdash X q_2 0 Y 1$
$\vdash q_2 X 0 Y 1$
$\vdash X q_0 0 Y 1$
$\vdash X X q_1 Y 1$
~~$\vdash X X Y q_2$~~

$\vdash X X Y q_1 1$
$\vdash X X q_2 Y Y$
$\vdash X q_2 X Y Y$
$\vdash X X q_0 Y Y$

$\vdash X X Y q_3 Y$

$\vdash X X Y Y q_3 B$

$\vdash X X Y Y B B q_f B$.

Accept.

001

$q_0 001$
$\vdash X q_1 01$
$\vdash X 0 q_1 1$
$\vdash X q_2 0 Y$
$\vdash q_2 X 0 Y$
$\vdash X q_0 0 Y$
$\vdash X X q_1 Y$
$\vdash X X Y q_3$
$\vdash X X Y q_3 B$
$\vdash X X Y B q_{reject} B$

Reject.

$L = \{012, 001122, 000111222, 000011112222, \cdots\}$

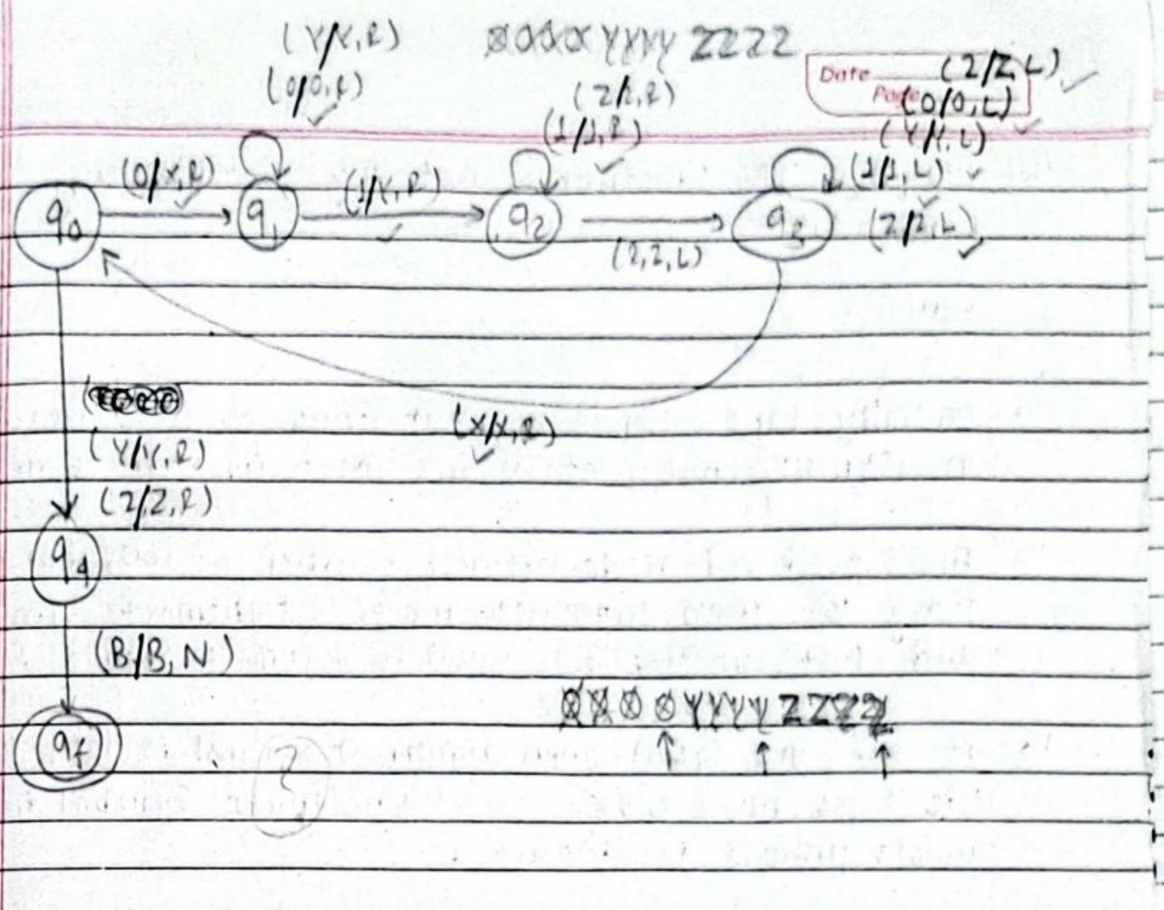
Q) Design a TM for $L = \{0^n 1^n 2^n : n \geq 1\}$

Solution: let TM be $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

$Q = \{$
$\Sigma = \{0, 1, 2\}$
$\Gamma = \{$
$q_0 = \{$
$B = \{B\}$
$F = \{$

δ:

Q/Σ	0	1	2	



Q) Design a TM whether or not any input string $w \in \{a, b\}^*$ is a palindrome.

Soln:

↳ Initially tape consists of input string w, tape head is on leftmost symbol of w and TM is in start state q_0.

↳ The tape head reads leftmost symbol of w, replace it by B. Then tape head moves to rightmost symbol and test whether it is equal to leftmost symbol.

↳ If they are equal then rightmost symbol is deleted, the tape head moves to new leftmost symbol and whole process is repeated

↳ if they are not equal, TM enters the reject state and computation terminates.

↳ TM enters accept state as soon as string currently stored on tape is empty.

↳ Let, TM be $T = (Q, \Sigma, T, \delta, q_0, q_{accept}, q_{reject})$

where,

$Q = \{q_0, q_a, q_b, q_{a'}, q_{b'}, q_{accept}, q_{reject}\}$

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

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

$q_0 = q_0$

$q_{accept} = q_{accept}$

$q_{reject} = q_{reject}$

δ:

Q/Σ	a	b	B
q_0	(q_a, B, R)	(q_b, B, R)	q_{accept}
q_a	(q_a, a, R)	(q_a, b, R)	$(q_{a'}, B, L)$
$q_{a'}$	(q_L, B, L)	q_{reject}	q_{accept}
q_L	(q_L, a, L)	(q_L, b, L)	(q_0, B, R)
q_b	(q_b, a, R)	(q_b, b, R)	$(q_{b'}, B, L)$
$q_{b'}$	q_{reject}	(q_L, B, L)	q_{accept}

(q_0)

① abaa

⊢ q_0 abaa

⊢ B$q_a$baa

⊢ Bb$q_a$aa

⊢ Bbaq_aa

⊢ Bbaaq_aB

⊢ Bbaq_a'aB

⊢ Bbq_LaB

⊢ B$q_L$baB

⊢ Bbq_baB

⊢ Bq_b

① ababa

⊢ $q_0$ababa

⊢ B$q_a$baba

⊢ Bb$q_a$aba

⊢ Bba$q_a$ba

⊢ Bbabq_aa

⊢ Bbabaq_aB

⊢ Bbabq_a'aB

⊢ Bbaq_LbBB

⊢ Bb$q_L$abBB

⊢ B$q_L$babBB

⊢ q_LBbabBB

⊢ B$q_0$babBB

⊢ BB$q_b$abBB

⊢ BBaq_bbBB

⊢ BBaq_bbBB

⊢ BBabq_bBB

⊢ BBaq_b'bBB

⊢ BBq_LaBBB

⊢ Bq_LBaBBB

⊢ BBq_0aBBB

⊢ BBBq_aBBB

⊢ BBq_a'BBBB

⊢ q_{accept}.

(II) abaa

q_0 abaa

⊢ B$q_a$baa

⊢* Bbaaq_aB

⊢ Bbaq_a'aB

⊢ Bbq_LaBB

⊢* q_LBbaBB

⊢ B$q_0$baBB

⊢ BBq_baBB

⊢ BBaq_bBB

⊢ BBq_b'aBB

⊢ q_{reject}.

Turing Machine for computing a function:

↳ a TM can be used to compute a function

↳ For this, adopt following policy to input any string to TM which is input to computation function.

① String ω is presented into from BωB where B is blank symbol and placed on the tape the head is positioned at a blank symbol which is immediately follows the string ω.

(II) The symbol to the current position of tape head can be shown as (q, BωB) or presented by I.D of TM as Bωq B

(iii) TM is said to halt on input w if we can reach to halting state after performing some operation.

i.e. for $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

TM is said to halt on input w if and only if $BwqB$ ~~reject~~ yield to $B\alpha q_f B$ for some $\alpha \in T^*$

$$(q, BwB) \vdash^* (q_f, B\alpha B)$$

Formal defination:

↳ a function $f(x) = y$ is said to be computable by TM defined as $(q, \Sigma, \Gamma, \delta, q_0, B, \{q_f\})$ if

$$(q_0, BxB) \vdash^* (q_f, ByB)$$

eg. Design a TM which computes the function $f(x) = x+1$ for each x belonging to the set of natural numbers.

Soln: given function on $f(x) = x+1$, we represent x on tape by 1's on tape.

i.e. for $x=1$, input tape will be B1B
for $x=2$, input tape will be B11B
and so on...

→ similarly output can be seen by number of 1s on tape when machine halts.

↳ Let TM be $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where,

$Q = \{q_0, q_f\}$
$\Sigma = \{1\}$
$\Gamma = \{1, B\}$
$q_0 = q_0$
$B = B$
$F = \{q_f\}$

δ can be:

	1	B
q_0	$(q_f, 1, R)$	$(q_0, 1, N)$

For input $x=3$, in input tape B111B.
Now,

$(q_0, B111B)$ $\vdash B11q_0B$
$(q_0, B1111)$ $\vdash B111q_01$
$(q_f, B(111B)$ or, $\vdash B1111q_fB$

Hence, output is B1111B
i.e. 4.

Turing Machine and Halting:

↳ acceptance by halting.
↳ TM halts if it enters a state q, scanning a tape symbol x and there is no move in this situation i.e. $\delta(q,x)$ is undefined.
↳ use assume TM always halts when it is in accepting state i.e without changing language accepted we can make $\delta(q,x)$ undefined whenever q is in accepting state.

Example

For $L_{eq} = \{w w^R\}$

	0	1	B
q_1	-		(q_r, B, R)
q_2	$(q_2, 0, R)$	$(q_2, 1, R)$	

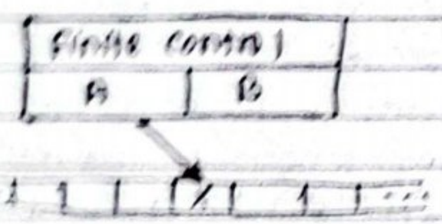
Why? For $w = 3$,
i.e. $\{w w^R \mid w \in \{0,1\}^*\}$

Using Machine with storage in the State:

- In TM, generally, any state represents the position in computation computation. But, the state can also be used to hold finite amount of data.

- We can use the finite control not only to represent position in computation but also to hold a finite amount of data.

- Hence, a state is considered as a tuple (state, data).



In this model of computation, δ is defined by

$$\delta([q_i, a], b) = [[q_j, c], D, E]$$

- It means that if q_i is the state and data is a, then symbol scanned on the tape is b, then the position of state will move right and the tape symbol b is replaced by c.

Eg:

Design a TM that recognizes a language $L = \{w w^R \mid w \in \{0,1\}^*\}$ where it set a final symbol either 0 or 1 and then checks that it is present or not in the remaining input.

Solution:

- Here, State is thought of pair with two components:

(i) a control portion q_0 or q_1 where q_0 indicates that TM has not yet reads its first symbol while q_1 indicates that it has read symbol and is checking that it doesn't appear elsewhere by moving right and hoping to reach blank cell.

(ii) a data portion which remembers the first symbol seen, which must be 0 or 1. The symbol B in this component means no symbol has been read.

here,

δ :→

1) $\delta([q_0, B], a) \longrightarrow ([q_1, a], a, R)$, for $a = 0$ or 1

2) $\delta([q_1, a], a') \rightarrow ([q_1, a], a', R)$, for $a = 0 \Rightarrow a' = 1$
$a = 1 \Rightarrow a' = 0$

3) $\delta([q_1, a], B) \longrightarrow q_{accept}$

4) $\delta([q_1, a], a) \longrightarrow q_{reject}$.

↳ if TM reaches first blank it enters accepting state.

↳ if TM reaches or encounters second occurrence of same symbol it halts without accepting.

Now, TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

where,

$Q = \{q_0, q_1, q_{accept}, q_{reject}\}$

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

$\Gamma = \{0, 1, B\}$

$q_0 = q_0$

$q_{accept} = q_{accept}$

$q_{reset} = q_{reject}$.

▷ 1000	1001
$\vdash [q_0, B]\ 1000$	$\vdash [q_0, B]\ 1001$
$\vdash 1\ [q_1, 1]\ 000$	$\vdash 1 [q_1, 1]\ 001$
$\vdash 10\ [q_1, 1]\ 00$	$\vdash 10\ [q_1, 1]\ 01$
$\vdash 100\ [q_1, 1]\ 0$	$\vdash 100\ [q_1, 1]\ 1$
$\vdash 1000\ [q_1, 1]\ B$	↑ ↑
$\vdash$ ~~100~~ q_{accept}	$\vdash q_{reject}$.

Recursively Enumerable language:

↳ the set of R.E languages are precisely those language that can be accepted by TM.

↳ strings that are not in the language may be rejected or may cause TM to go into infinite loop.

Recursive language:

↳ a language is recursive if there exists a TM that accepts every string of the language and rejects strings that are not in language.

↳ recursive language always halts TM.

Turing Recognizable language.

↳ a language L is Turing recognizable if there exists.

a TM such that for all strings w.

- if $w \in L$, eventually TM enters qaccept
- if $w \notin L$, either TM enters qreject or TM never terminates.

Turing Decidable language:

↳ a language L is turing decidable if there exists a TM such that for all strings w

- if $w \in L$, TM enters qaccept
- if $w \notin L$, TM enters qreject.

↳ decidable language always terminates.
↳ Recognizable language can run forever without deciding

Extension of TM

① Multiple TM:

↳ Multiple TM consists of a finite control and finite no. of tapes.
↳ each tape is divided into cells and each cell can hold any symbol of finite tape alphabets.
↳ the set of tape symbols include a blank and input symbols.
↳ In multiple TM initially,

i) the input w is placed on the first tape
ii) all other cells of the tapes hold blank
iii) control is in initial state q_0
iv) the head of first tape is at left end of I/P.
v) all other tape heads are at some arbitary cells.

↳ A move of multiple TM depends on the state and the symbol scanned by each of tape head.
↳ In one move, two multiple TM does following

i) the control enters in a new state which may be same previous state.
ii) on each tape, a new symbol is written on the cell scanned, these symbols may be same as symbols previously there.
iii) each of tape head make a move either left or right or remains stationary different head may have different direction independently. i.e. if head of first tape moves leftward at same time other heads can move another directions or remain stationary.

$\delta(q_1, a_1, a_2, a_3) \Rightarrow (1, b_1, b_2, b_3, L, R, N)$

Date
Page

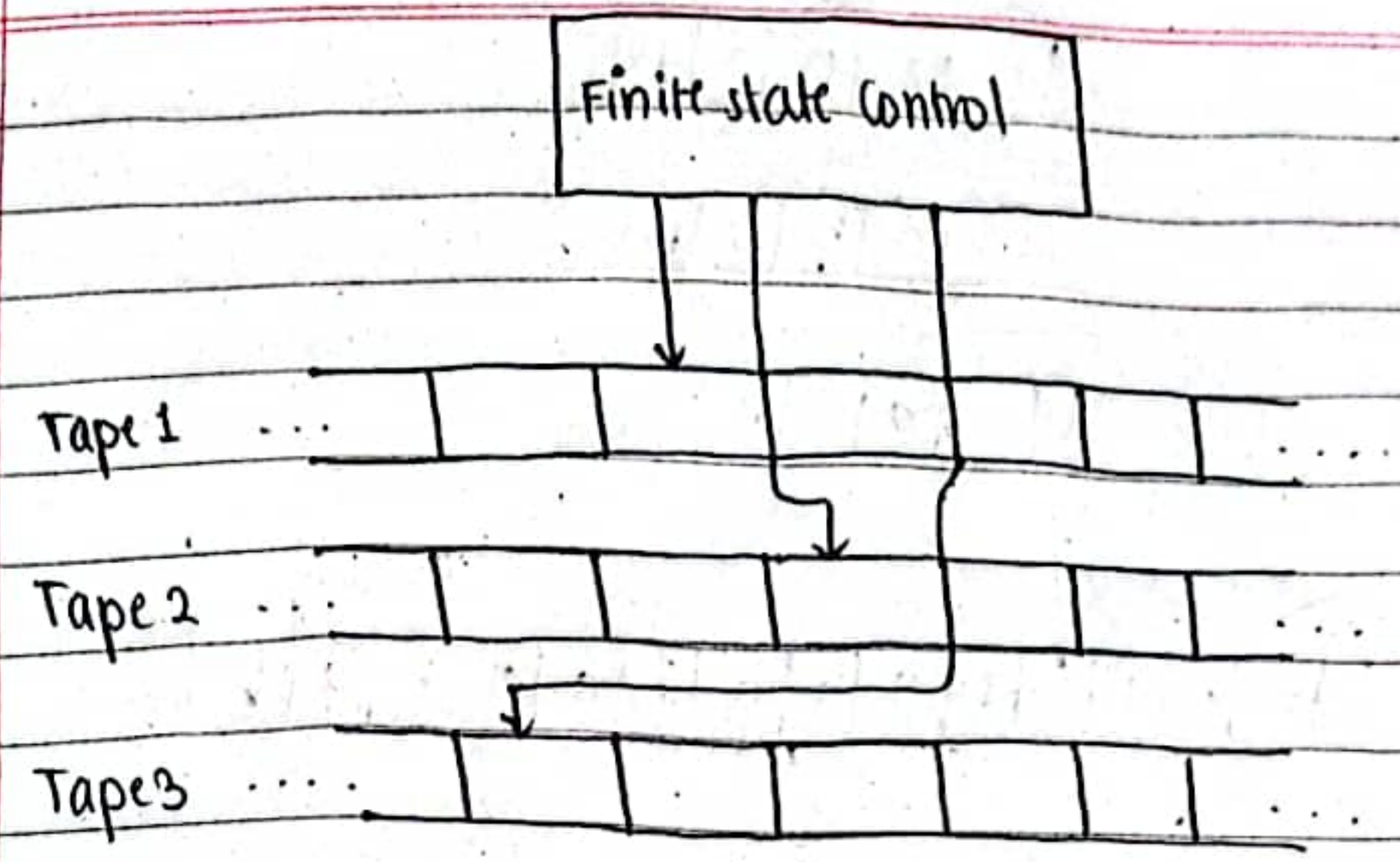
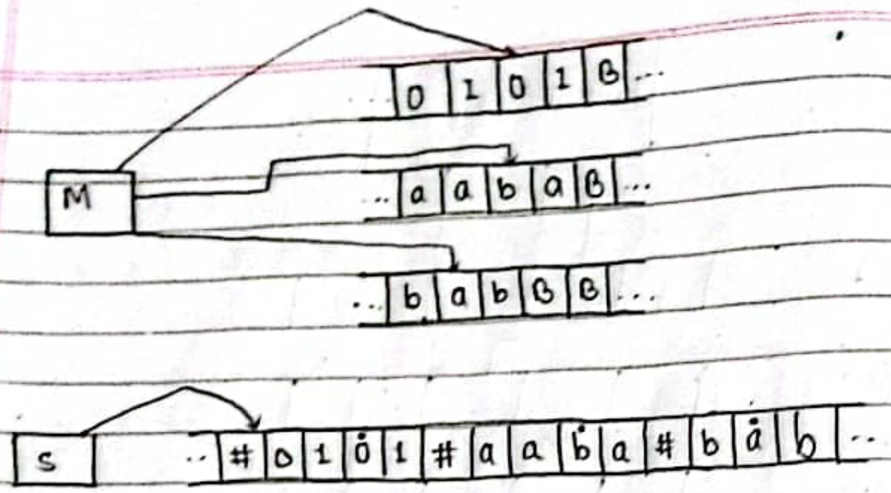


Fig. Multitape TM

- Let $K \geq 1$ be an integer. ~~but~~ Any K-tape TM can be converted to an equivalent one Tape TM

Here,

- Convert a multitape TM, M, to an equivalent single taple TM s.
- Key idea is to show how to simulate M with s.
- Let, M has K-tapes, then s simulates the effect of K-tapes by storing their information on its single tape.
- It uses the new symbol # as a delimiter to separate the contents of the different tapes.
- In addition to the contents of these tapes, s must keep track of the locations of the header by writing a tape symbol with a dot (.) above it to mark the place where the head on that tape would be
- Following fig. illustrates how one tape can be used to represent three tapes.



- On input $w = w_1 \dots w_n$

1) First S puts its tape into the formal that represents all K-tapes of M. The formatted tape contains

$$\# \dot{w_1} w_2 \dots w_n \# \dot{B} \# \dots \#$$

2) To simulate a single move, S scans its tape from the first #, which makes lefthand end, to the (K+1)# which makes righthand, and in order to determine the symbols under virtual heads. Then S makes a second path to update the tape according to way that M's transition function dictate.

3) If at any point S moves one of the virtual head to the right onto a #, this action signifies that M has moved the corresponding head onto a previous unread blank portion of that tape. so, S write a blank symbol on this tape cell and shifts the tape contents, from this cell until rightmost # one unit to the right. Then, it continues the simulation as before.

Non Deterministic Turing Machine

- is the one which at any point in a computation may processed according to several possibilities.
- the transition functions δ are subsets rather than single element of set

$$Q \times \Gamma \times \{L, R, S\}$$

- Here, transition function δ is such that for each state q and tape symbol x, $\delta(q, x)$ is set of triples $\{(q_1, x_1, D_1), (q_2, x_2, D_2) \dots (q_k, x_k, D_k)\}$ where K is finite integer.

- The NTM can choose at each step any triples to be the next move but it can't however pick a state from one, a tape symbol from another and direction from yet another.
- The computation of a NTM is a tree whose branches corresponds to different possibilities for the machine.
- If ~~the~~ some branch of the computation leads to the accept state, the machine accepts its input.
- If M_N is a non deterministic TM, then there is a deterministic TM M_D such that

$$L(M_N) = L(M_D)$$

Chapter 5

UNDECIDABILITY

Encoding of TM

- Formulate a notational system, where we can encode both an arbitary TM T1 and an i/p string x over an arbitrary alphabet as strings $e(T1)$ and $e(x)$ over some fixed alphabets.
- This encoding must not destroy any information i.e. we must be able to reconstruct TM T1 and string x.
- For encoding of TM, use alphabet $\{0,1\}$ although TM may have much larger alphabet.
- start assigning positive integer to each state, each tape symbol and each of three directions in TM T1 we want to encode.
- Here, we assume two fixed infinite sets

  $Q = \{q_1, q_2, \ldots\}$ and $S = \{a_1, a_2, \ldots\}$

  so that for any TM $T = (Q_1, \Sigma, \Gamma, \delta, q_0, B, F)$

  $Q_1 \subseteq Q$

  $\Gamma \subseteq S$

- We can represent a state or a symbol by a string of 0's of appropriate length. Here; 1's are used as separator.
- Once, we have established an integer to represent each state, symbol and direction, we can encode the transition function δ.

- Let one transition rule is

  $\delta(q_i, a_j) = (q_k, a_l, D_m)$ for some integer i, j, k, l, m.
- then we can code this rule by strings

  $s(q_i)\,1\,s(a_j)\,1\,s(q_k)\,1\,s(a_l)\,1\,s(D_m)$ say m_1

  where s is encoding function

- A code for entire TM T1 consists of all codes for transition in same order, separated by pair of 1's like

  $m_1 11\, m_2 11 \ldots m_n$
- Now code for TM and i/p string x will be formed by separating them by three consecutive 1's

  $e(TM)\,111\,e(x)$

Encoding function (s)

- First associate a string of 0's to each state, each tape symbol and each of three directions.
- Let s is defined as,

  $s(B) = 0$

  $s(S) = 0$

  $s(L) = 00$

  $s(R) = 000$

  $s(a_i) = 0^{i+2}$ for each $a_i \in S$

  $s(q_i) = 0^{i+2}$ for each $q_i \in Q$

Q) Consider an example where TM T is defined as $T = (\{q_1, q_2, q_3\}, \{a, b\}, \{a, b, B\}, \delta, q_1, B, F)$ where δ is defined as:

(i) $\delta(q_1, b) = (q_3, a, R)$

(ii) $\delta(q_3, a) = (q_1, b, R)$

(iii) $\delta(q_3, b) = (q_2, a, N)$

(iv) $\delta(q_3, B) = (q_3, b, L)$

Encode TM T with string x where x = 'abaa'.

PTO

Ans: $s(B) = 0$

$s(L) = 00$

$s(R) = 000$

$s(S \text{ or } N) = 0$

$s(a) = s(q_1) = 0^{1+1} = 00$

$s(b) = s(q_2) = 0^{2+1} = 000$

$s(q_1) = 0^{1+2} = 000$

$s(q_2) = 0^{2+2} = 0000$

$s(q_3) = 0^{3+2} = 00000$

For $\delta(q_1, b) = (q_3, a, R)$,

$s(q_1) 1 s(b) 1 s(q_3) 1 s(a) 1 s(R)$

$= 000100010000010010000$

$= e(m_1)$

For $\delta(q_3, a) = (q_1, b, R)$,

$s(q_3) 1 s(a) 1 s(q_1) 1 s(b) 1 s(R)$

$= 00000 1 00 1 000 1 000 1 000$

$= e(m_2)$

For $\delta(q_3, b) = (q_2, a, N)$,

$s(q_3) 1 s(b) 1 s(q_2) 1 s(a) 1 s(N)$

$= 00000 1 000 1 0000 1 00 1 0$

$= e(m_3)$

For $\delta(q_3, B) = (q_3, b, L)$,

$s(q_3) 1 s(B) 1 s(q_3) 1 s(b) 1 s(L)$

$= 00000 1 0 1 00000 1 000 1 00$

$= e(m_4)$

$e(T) = e(m_1) 11 e(m_2) 11 e(m_3) 11 e(m_4)$

$= 0001000100000100100011000001001000100011$

$00000100010000100101100000101000001000100$

Encoding for string:

$e(n) = s(a) 1 s(b) 1 s(a)$

$= 00 1 000 1 00$

So, $e(T) 111 e(n) =$

Review of Set, Function, Relation

Set:
- is a group of objects represented as a unit
- Objects in set are called its element or members.
- set is described by listing its elements inside braces
  e.g. $A = \{2, 6, 11, 15\}$
- Symbol $\in$ for set membership
- Symbol $\notin$ for set nonmembership
  e.g. $2 \in \{2, 6, 11, 15\}$
  $7 \notin \{2, 6, 11, 15\}$
- Neither the order of ~~descending~~ describing a set nor repetition of its element matters
  i.e. $\{2, 6, 11, 15\} = \{2, 11, 2, 15, 6, 11, 2\}$
- Two sets are equal if and only if they have same elements.
- A set with only one element is called singleton.
- A set with no elements at all is called empty set, denoted by ϕ.
- A set that contains infinitely many elements is called infinite set.
  e.g. $A = \{1, 3, 6, 7, 9 \ldots\}$
- A set that is not infinite is finite set.

Subset
- For any two sets A and B, A is subset of B if every member of A is also member of B
- Written as $A \subseteq B$.

Proper Subset
- For two sets A and B, A is proper subset of B if A is subset of B and not equal to B
- written as $A \subset B$

- Any set is subset of itself.
- Empty set is subset of every set, including the empty set itself.

- Set containing elements can be described according to some rule.
  i.e. $\{n \mid \text{rule about } n\}$

e.g. 1) $\{n \mid n = m^2 \text{ for some } m \in \mathbb{N}\}$
$\{1, 4, 9, 16, \ldots\}$

2) $A = \{n^2 - 4 : n \text{ is an integer and } 0 \leq n \leq 10\}$
$A = \{-4, -3, 0, 5, 12, 21, 32, 45, 60, 77, 96\}$

Set Operations:

1) Union:
- For two sets A and B, the union of A and B is the set combining all the elements in A and B into a single set.
  $A \cup B = \{n : n \in A$ ~~and~~ or $n \in B\}$

2) Intersection:
- For two sets A and B, the intersection of A and B is the set of elements that are both in A and B
  $A \cap B = \{n : n \in A \text{ and } n \in B\}$

3) Complement
- For a given set A, complement of A is the set of all elements that are not in A.

$$\bar{A} = \{x : x \notin A\}$$

4) Difference
- For two sets A and B, the difference of A and B is the set of all elements of A but not in B.
- $A - B = \{x : x \in A \text{ and } x \notin B\}$

- Let A, B, C are three arbitary sets, $\cup$ represents universal set and ϕ is empty set.

Then,

(i) Idempotency Law
$A \cup A = A$
$A \cap A = A$

(ii) Commutative Law
$A \cup B = B \cup A$
$A \cap B = B \cap A$

(iii) Absorptive Law
$A \cup (A \cap B) = A$
$A \cap (A \cup B) = A$

(iv) Associative Law
$A \cup (B \cup C) = (A \cup B) \cup C$
$A \cap (B \cap C) = (A \cap B) \cap C$

(v) Distributive Law
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

(vi) Demorgan's law
$(A \cup B)' = A' \cap B'$
$(A \cap B)' = A' \cup B'$

(vii) Laws of complements
$(A')' = A$
$A \cap A' = \phi$
$A \cup A' = \cup$

(viii) Laws of empty set
$A \cup \phi = A$
$A \cap \phi = \phi$

(ix) Universal Set Laws
$A \cup \cup = \cup$
$A \cap \cup = A$

Disjoint set
Two sets are disjoint if they have no elements in common i.e their intersection is empty.

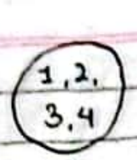
Power set
For given set A, power set of A represented as 2^A is the collection of all subsets of a set A.

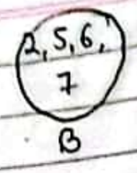
Cartesian Product
For two sets A and B cartesian product is defined as

$$A \times B = \{(x, y) : x \in A \text{ and } y \in B\}$$

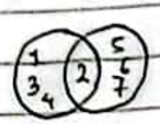
e.g.
$A = \{2, 4, 6\}$, $B = \{1, 3\}$
$A \times B = \{(2,1), (2,3), (4,1), (4,3), (6,1), (6,3)\}$

Venn Diagram
Set diagram is a diagram that show all possible logical relation between finite collection of sets.
e.g. $A = \{1, 2, 3, 4\}$, $B = \{2, 5, 6, 7\}$

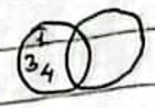




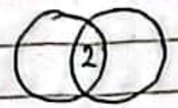
A∪B ⇒



A-B ⇒



A∩B ⇒



Sequence and Tuples

- Sequence of object is a list of objects in same order.
  e.g. (2,5,9,11)
  (2,5,9,11) ≠ (2,11,5,9)
  (2,5,9,11) ≠ (2,11,2,5,9)

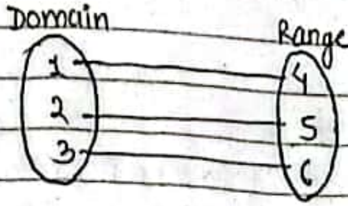
- As set sequence may be finite or infinite.
- Finite sequence often are called tuples.
- A sequence with k-elements is called k-tuple.
- A 2 tuple is called a pair.

Function and Relation

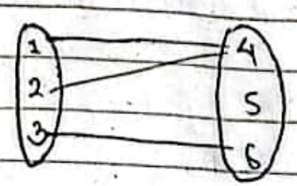
- Function is an object that sets up an input output relation.
  i.e. it takes input and produce an output.
- If f is a function with input a and output b then f(a)=b
- Function is also termed as mapping i.e. f(a)=b ; f maps a to b.
- The set of possible input to the function is called its domain.
- The output of function is called its range.
- f: D→R ⇒ f is a function with domain D and range R

Date
Page

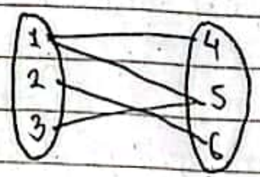
- Function is a well behaved relation.
- in function, for (x,y) each x has only one y.
- For relation, one, some or all x have more than ~~one~~ y.
- All function are relation but not all relations are function.



function



function



Relation

- A function f: A→B is one to one (or injective) if for any two distinct elements a & a' in A, we have f(a) = f(a').
- A function f is onto (or subjective), if for each elements b∈B, there exists an element a∈A, such that f(a) = b i.e. the range of f is equal to set B.
- A function f is bijection if f is both injective & subjective.

Relation

Relation is the subset of cartesian product i.e. for two sets A & B then relation R is,

$$R \subseteq A \times B$$

Types of Relation

1. Reflexive Relation
- R is reflexive relation if for every element in $a \in A$, we have $(a,a) \in R$.

2. Symmetric/Anti symmetric Relation
- R is symmetric for all a & b in A, if $(a,b) \in R$ or $(b,a) \in R$.
- If relation isn't symmetric then it is anti-symmetric.

3. Transitive Relation
- R is transitive if for all a,b & c in A.
  $(a,b) \in R$ & $(b,c) \in R$, then also $(a,c) \in R$.

4. Equivalent Relation
- Relation $R \subseteq A \times A$ is an equivalent relation, if it is reflexive, symmetric & transitive.

5. Partial Order Relation
- Relation R defined on set A is partial order relation if it is reflexive, anti-symmetric & transitive.

6. Total order relation
- A partial order relation is total order, if for all a & b in A, either $(a,b) \in R$ or $(b,a) \in R$.

Q) Let, $A = \{1,2,3\}$ then find all relations.

Equivalence class:

Let 'R' be an equivalence relation defined on set A, then equivalence class is

$$[a] = \{x : a \in A, (a,x) \in R\}$$

e.g. Let $A = \{1,2,3\}$

$R = \{(1,1),(2,2),(3,3),(2,1),(1,2)\}$

Then equivalence class be

$[1] = \{1,2\}$

$[2] = \{1,2\}$

$[3] = \{3\}$

Proof Techniques:

1. Mathematical Induction
2. Piegon hole principle
3. Diagonalization

1. Mathematical Induction

- is a method of mathematical proof typically used to establish a given statement for all natural numbers.
- Let A be a set of natural numbers such that
  i) $0 \in A$
  ii) for each natural no n, if $\{1 \ldots n\} \subseteq A$
  then $n+1 \in A$
  then $A = N$

- Let 'P' be a property defined on the set of natural number which is true for $n=1$.

- It is assured that p is true for $n=k$.
- if we can show p is true for $n=k+1$ we can conclude that p is always true.

First step: Basic step

- Here, we try to show given statement is true for $n=0$ or $n=1$.
- if not true for $n=0$, or $n=1$, we can assume higher values for n.

second step: Induction Hypothesis

- It is assumed that given statement is true for $n=k$.

Third step: Induction step

- Try to prove given statement true for $n=k+1$
- If this step is valid, then we can conclude that given statement is always true.

Q) Prove that sum of first n natural is given by

$$1+2+3+\ldots+n = \frac{n(n+1)}{2} \quad : n \geq 1$$ using mathematical induction.

Sol^n

1) Basic step:

for $n=1$, LHS $= 1$

$$\text{RHS} = \frac{1(1+1)}{2} = 1$$

$\therefore$ LHS = RHS

i.e. from basic step, given statement is true for $n=1$.

2) Induction hypothesis

for $n=k$, given statement becomes,

$$1+2+3+\ldots+k = \frac{k(k+1)}{2}$$

3) Induction step

for $n=k+1$,

$$\text{LHS} = 1+2+3+\ldots+k+k+1$$

$$= \frac{k(k+1)}{2} + k+1$$

$$= \frac{k(k+1)+2(k+1)}{2}$$

$$= \frac{(k+2)(k+2)}{2}$$

$$\text{RHS} = \frac{(k+1)(k+1+1)}{2}$$

$$= \frac{(k+1)(k+2)}{2}$$

LHS = RHS

i.e. given statement is true for $n=k+1$

Hence,

$$1+2+3+\ldots+n = \frac{n(n+1)}{2}$$ Proved.

Q) Using mathematical induction, prove that sum of first n positive odd no $1+3+5+\ldots+2n-1 = n^2$

Sol^n

Basic step:

$n=1$, $1=1$

Induction hypothesis:

$1+3+5+\ldots+2K-1=K^2$

Induction step:

$n=K+1$

$LHS = 1+3+5+\ldots+2k-1+2(K+1)-1$

$= K^2+2K+2-1$

$= K^2+2K+1$

$= (K+1)^2$

$RHS = (K+1)^2$

LHS = RHS

i.e. given statement is true for

Hence, $1+3+5+\ldots+2n-1=n^2$ Proved.

2. Pigeon hole principle

- If n items are put into m containers with $n>m$, then at least one container must contain more than one item.
- If there are n pigeon holes and n+1 pigeons, then at least one pigeon hole is occupied by two or more pigeons.
- Mathematically,

Let A and B be two finite sets with $|A|>|B|$ then one to one function doesn't exists from A to B.

3. Proof by contradiction

- Establishes the truth or validity of propsition by showing that proposition being false would imply a contradiction.

Q) Prove that sum of rational and irrational no. is irrational.

Sol

Let r is rational and n is irrational no. form statement (r+n) is irrational.

Now, - assume (r+n) is rational

- we know if r is rational then -r is also rational also sum of two rational no. is also rational.

$\therefore (r+n)+(-r) \Rightarrow$ rational

$r+n-r \Rightarrow$ rational

$n \Rightarrow$ rational

This contradicts our assumption that n is irrational.

Hence, r+n is irrational i.e. sum of rational and irrational number is irrational.

Proved.

4. Diagonalization Principle

- Let R be a binary relation defined on set A.
- Let D be a diagonal set defined on relation R as,

$\{a : a \in A \text{ and } (a,a) \in R\}$

- For each $a \in A$, row sets are defined as,

$R_a = \{b : b \in A \text{ and } (a,b) \in R\}$

Then, D is distinct from each R_a.

PTO →

Date
Page

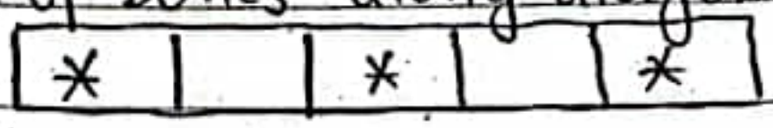
e.g.) Let A = {1,2,3,4,5} and R be any relation defined on A and given by R = {(1,1), (1,4), (2,3), (2,4), (2,5), (3,1), (3,3), (4,2), (4,5), (5,5)}

Sol?

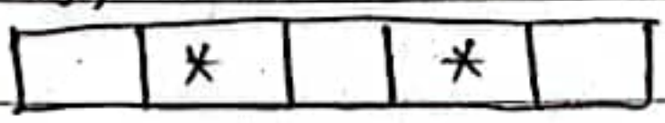
Verifying diagonalization principle, given R can be pictured as,

	1	2	3	4	5
1	*			*	
2			*	*	*
3	*		*		
4		*			*
5					*

Now, sequence of bones along diagonal is,



its complement is,



i.e. D = {2,4}

again,

Row lets are $R_1 = \{1,4\}$

$R_2 = \{3,4,5\}$

$R_3 = \{1,3\}$

$R_4 = \{2,5\}$

$R_5 = \{5\}$

Here, diagonal set differs from each row sets R_1 to R_5.

∴ Diagonalization principle is verified.