PIE Tech

POLLACHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE and Affiliated to Anna University) sky is the limit

Department of Computer Science and Engineering

Regulation 2021

II Year – IV Semester

CS3452- Theory of Computation

Introduction

- Toc is based on mathematical computation These computations are used to represent Various mathematical models.

- This subject we will study many interesting model such as Finite automata, push down automata, turing machines.

- This subject is a Fundamental subject and is very close to the subject like Compilers, operating system, system s/w.

Introduction to Formal Proofs

- The formal proof can be using deductive proof and Inductive proof.

* The deductive proof consists of Sequence of statements given with logical reasoning in order to prove the First or initial Statement.

- The inductive proof is a recursive kind of proof which consists of sequence of parameterized stmts.

Various Forms of proofs.

1. proofs about sets.
2. proofs by Contradiction.
3. proofs by counter example.

1. A proof about sets

The set is a collection of elements or items. By giving proofs about the sets we try to prove certain properties of the sets.

if there are two expressions. A & B.

Let $P \cup Q = Q \cup R$ if we map expression A with $P \cup Q$ and expression B with $Q \cup R$.

2. proof by contradiction

This type of Proof if A then B

if statement A is false we try to get conclusion of statement B.

$P \cup Q = Q \cup P$ is not true. That is $P \cup Q \neq Q \cup P$

3. Proof by Counter-Example

we need to see all Possible conditions in which that statement remains true.

There is no such pair of Integers

$a \bmod b = b \bmod a$

proof : consider $a=2$ and $b=3$. Then clearly

$2 \bmod 3 \neq 3 \bmod 2$.

Inductive Proof

- It is Special Proofs based on Some observations. It is used to prove recursively defined objects.

1. Basis - Here we assume the lowest possible value.

   - This is an initial Step in the proof of mathematical induction.

2. Induction hypothesis :

   Here we assign value of n to some other value k.

   That mean we will check whether the result is true for n=k or not.

3. Inductive Step:

   if n=k is true then we check whether the result is true for n=k+1 or not.

   If we get the same result at n=k+1 then we can state that given proof is true by principle of mathematical induction.

ex: Prove by induction on n that

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

1) Basis of induction

Assume n=1, Then

$LHS = n, = 1$

$RHS = \frac{n(n+1)}{2} = \frac{1(1+1)}{2} = \frac{2}{2} = 1$

2) Induction hypothesis.

Now we will assume n=k and will obt the result for it.

$$1+2+3+\ldots+k = \frac{k(k+1)}{2}$$

3) inductive step

Now we assume that equation is true for n=k. And we will than check it it is also true for n=k+1 or not.

Assume n=k+1

$$L.H.S = \underbrace{1+2+3\ldots+k}+k+1$$

$$\frac{n(n+1)}{2}$$

$$\frac{k(k+1)}{2}+k+1$$

$$= \frac{k(k+1)+2(k+1)}{2}$$

$$= \frac{(k+1)(k+2)}{2}$$

$$= \frac{(k+1)(k+1+1)}{2}$$

$$= R.H.S$$

Basic concept of Automata Theory

It has a basic Fundamental unit called set. The set is used to represent the mathematical model.

Set.

- set is defined as collection of objects
- These objects are called elements of the set.

- All the elements are enclosed within curly brackets { and }

* If 'a' is an element of set A then we say that $a \in A$

* if 'a' is not an element of A then we say that $a \notin A$.

Subset :

The subset A is called subset of set B if every element of set A is present in set B but reverse is not true.

It is denoted by $A \subseteq B$.

ex: $A = \{1, 2, 3\}$ & $B = \{1, 2, 3, 4, 5\}$

the $A \subseteq B$.

Empty set .

The set having no element is it is called empty set. ex: $A = \{\}$

Null String

The null element is denoted by ε (or) ^ character. But ε does mean ϕ

Power set :

The power set is a set of all the subset of its elements.

$A = \{1, 2, 3\}$

Power set ⓟ $= \{\phi, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{3, 2\}$
$\{2, 3\}, \{1, 2, 3\}\}$

Scanned by CamScanner

Equal set :

The two sets are said to be equal (A=B) if $A \subseteq B$ and $B \subseteq A$

ex :

$A = \{1, 2, 3\}$ and $B = \{1, 2, 3\}$ then $A = B$

$|A|$ denotes the length of set A.

$A = \{1, 2, 3, 4, 5\}$

$|A| = 5$

Operations on set

(i) union (ii) Intersection (iii) Difference

(iv) Complement.

(i) $A \cup B$ is union operation

$A = \{1, 2, 3\}$ $B = \{1, 2, 4\}$

$A \cup B = \{1, 2, 3, 4\}$

(ii) $A \cap B$ is intersection operation

$A = \{1, 2, 3\}$ and $B = \{2, 3, 4\}$

$A \cap B = \{2, 3\}$ Collection of common ele

From both the sets

(iii) $A - B$ is the difference operation

$A = \{1, 2, 3\}$ & $B = \{2, 3, 4\}$

$A - B = \{1\}$ elements which are there in

Set A but not

Scanned by CamScanner

(iv) $\bar{A}$ is a complement operation.

$\bar{A} = U - A$ where U is a universal set.

ex:

if $U = \{10, 20, 30, 40, 50\}$

$A = \{10, 20\}$

then $\bar{A} = U - A$

$= \{30, 40, 50\}$

Cartesian product of Two sets.

The cartesian product of two sets A and B is a set of all possible ordered pairs

cartesian product is denoted by $A \times B$

$\therefore A \times B = \{ \{a, b\} \mid a \in A$ and $b \in B\}$

ex: $A = \{a, b\}$ and $B \{0, 1, 2\}$

$A \times B = \{(a,0), (a,1) (a,2) (b,0), (b,1) (b,2)\}$

cardinality of sets

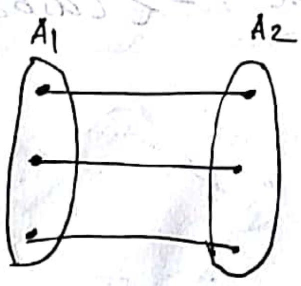
- It is nothing but the number of members in the set.

(i) one to one

(ii) one to many

(iii) many to one

(iv) many to many



cardinality of two sets.

Relations

- Relationship is a major aspect between two objects .

- one object can be relatee with the other object by a 'mother of relation'

* The relation R is a collection for the set S which represents the pair of elements .

Properties of relations

1. Reflexive if iRi for all i in S.
2. Irreflexive if iRi is false for all i in S.
3. Transitive if iRj and jRk imply iRk
4. Symmetric if iRj implies jRi
5. Asymmetric if iRj implies that jRj is false.

ex: if A = {a, b} then

reflexive relation R can be = {(a,a), (b,b)}

irreflexive = {(a,b)}

transitive = {(a,b), (b,a), (a,a)}

symmetric = {(a,b), (b,a)}

asymmetric = {(a,b)}

Kleen closures

- It is also called as Star closure.
- This is set of strings of any length (including null string ε).
- Each string is obtained from input set Σ then kleen Star of the empty language ϕ is the empty string ε.

Aritsmetic Expressions

It has done arithmetic operations are performed on operands.

The aritsmetic operators can be denoted as $S = \{+, -, *, /, \wedge\}$.

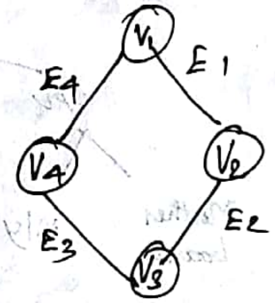
$R_1 = a+b, \ a-b, \ a*b, \ a/b, \ a \wedge b$.

Graphs

A graph denoted $G = (V, E)$ consists of finite set of vertices (or nodes)

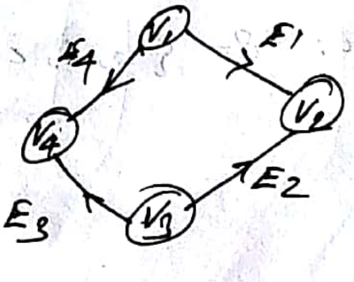
E = set of edges

V = Vertices.



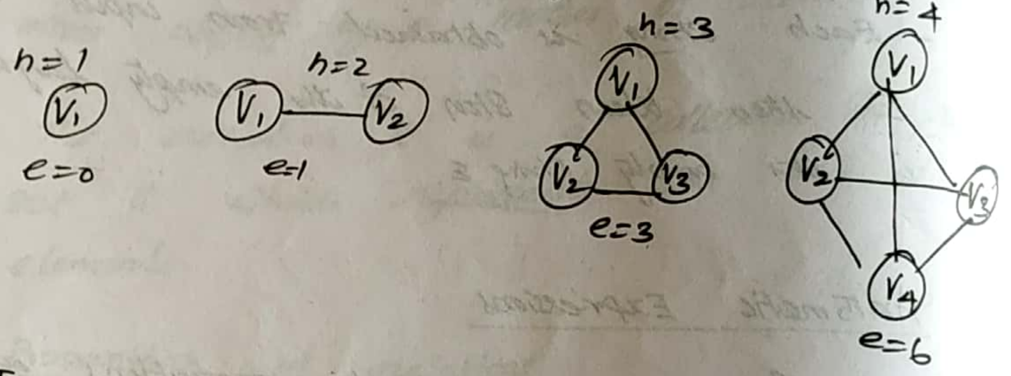
Directed Graph

It is also a collection of vertex and edges in which the direction are mentioned along the edges



Complete Graph

The complete graph is a simple undirected graph in which every pair of distinct Vertices is connected by unique edge.

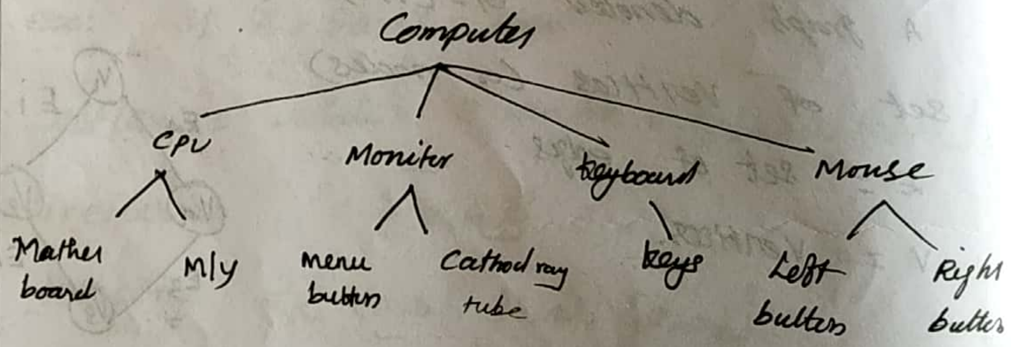


Trees

Tree is a collection of Vertices and edges

1. There is one Vertex, called root which has no predecessors.

2. From this root node all the Successors are ordered.



Alphabets

An alphabet is a finite collection of symbols.

ex: $S = \{a, b, c \ldots, z\}$

String: String is Finite collection of symbols from alphabet.

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

$\Sigma = \{ab, aa, aaa, bbb, ba, aba \ldots\}$

Symbols :

Language: The languages is a collection of appropriate strings.

$\Sigma = \{\epsilon, 0, 00, 000, \ldots\}$

Applications of Automata Theory

Automation is a kind of machine which takes some string as input.

1. Automata theory is the base for the formal language and these Formal languages are useful of the programming language.

2. Automata theory plays an important role in compiler design

3. To prove the correctness of the program automata theory is used.

4. Switching theory and design and analysis of digital circuits automata theory is applied.

5. Automata theory deals with the design Finite State machines.

Scanned by CamScanner

Finite State Systems

- The Finite State system represents a mathematical model of a system with certain input.

- The input when is given to the machine it is processed by various states, these states are called as intermediate states.

- It is a very good design tool for the programs such as
  - text editors
  - lexical analyzers.

Finite automata

It is a collection of 5-tuple $(Q, \Sigma, \delta, q_0, F)$

- Q is a Finite set of states, which is non empty.

- Σ is input alphabet

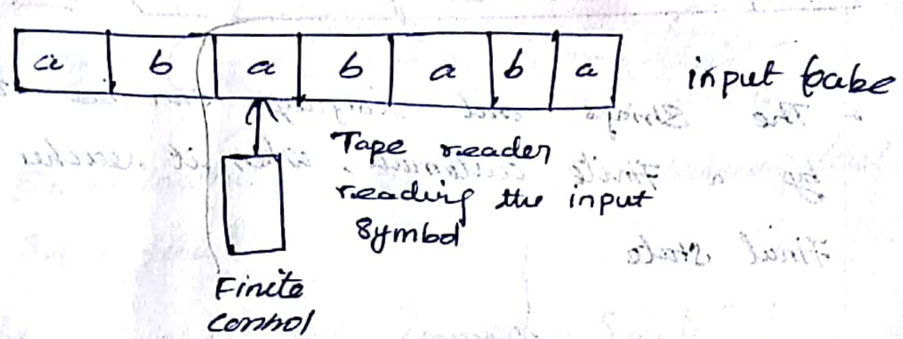
- q_0 is an initial state and q_0 is in Q $q_0 \in Q$

- F is a set of Final states.

- δ is a transition Function or mapping function.

Scanned by CamScanner

Finite Automata Model .



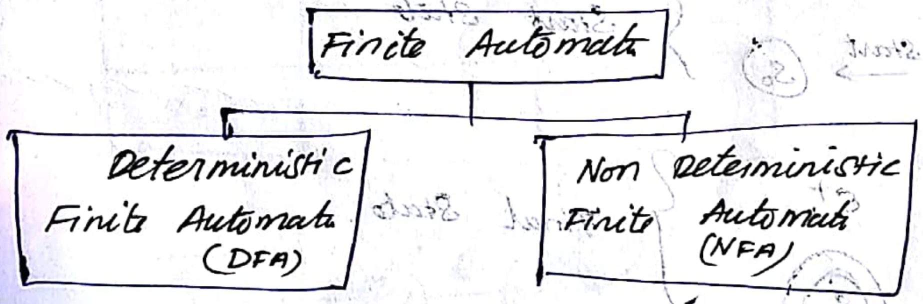
(i) input tape - It is a linear tape having some number of cells . Each i/p symbol is placed in each cell.

(ii) Finite control

It decides the next state on receiving particular input from input tape.

∴ The tape reader reads the cells one by one from left to right and at a time only one i/p symbol is read.

Types of Automata


Finite Automata
Deterministic Finite Automata (DFA)
Non Deterministic Finite Automata (NFA)

(i) DFA

Here each move in this automata is uniquely determined on current i/p & current state

(ii) NFA

— It is none deterministic in nature.

— B

Acceptance of Strings and Languages

- The Strings and language can be accepted by a Finite automata, when it reaches to a Final State.

1. Transition diagram

1) Finite set of State k.

2) Finite set of Symbols Σ.

3) A non empty set S of k. or Start State

4) A set $F \subseteq k$ of Final States.

5) A transition Function $k \times A \rightarrow k$ with k as State and A as input from Σ^*

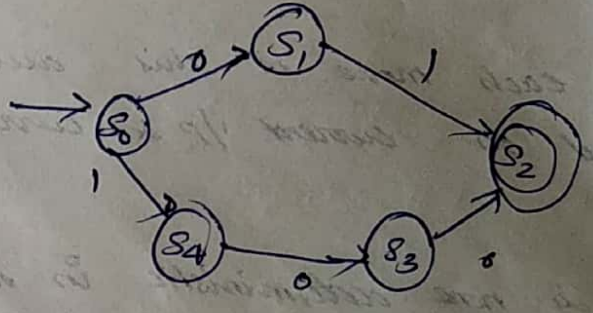
(S_1) - Represents the State

→ Represents transitions from one State to another.

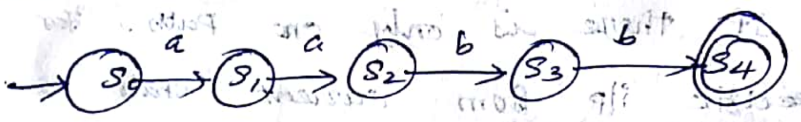
S_0^-
start → (S_0) } Start State

S_n^+
((S_n)) } Final State.

ex:



Another ex:



s₀ – initial s1, s2, s3 – intermediat

s₄ – final

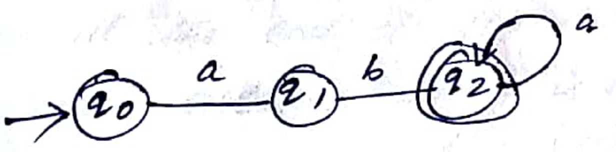
The input set is Σ = {a, b}

2. Transition table

– This is a tabular represtation of Finite automata.

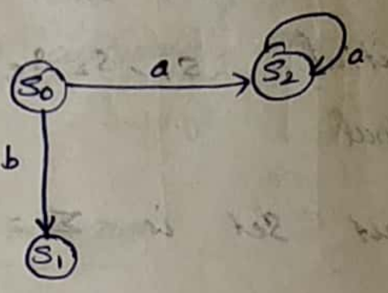
– For transition table the transition function is used.

I/P States	a	b
q0	q1	–
q1		q2
q2	q2	–



DFA (Deterministic Finite Automata)

- If there is only one path, for a specific i/p from current state to next sta



1) The finite set of states which can be denoted by Q.

2) The finite set of input symbols Σ.

3) The start state q_0 such that $q_0 \in Q$.

4) A set of final states F such that F such that $F \in Q$.

5) The mapping function or transition function denoted by δ.

ex : $q_1 = \delta(q_0, a)$

current state q_0, with i/p a the next state transition is q_1.

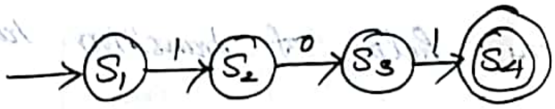
$$A = (Q, \Sigma, \delta, q_0, F)$$

Ex:

1) Design a FA which accepts the only input 101 over the input set $\Sigma = \{0,1\}$

Sol: Here only input 101 will be accepted.

- There is no other path shown for other i/p.



ex: Design a FA which checks whether the given binary number is even.

- The binary no is made up of 0's and 1's
- When any binary number end with 0 it is always even
- When a binary number ends with 1 it is always odd.

0100 is a even number = 4

0011 is a odd number = 3.

- The Finite Automata indicates clearly S_1 is a state which handles all the 1's and S_2 is a state which handles all the 0's.

01000 ⇒ $0S_21000$

$01S_1000$

$010S_200$

$0100S_20$

$01000S_2$

Now at the end of the state S_2 input is 0 so here accept. So it is a even number.

1011 ⇒ $1S_1011$

$10S_211$

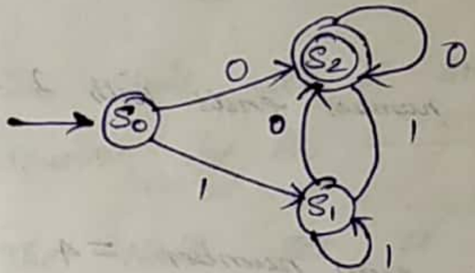
$101S_11$

$1011S_1$

Now we are at the State S_1 which is a non Final state.

So FA with the help of transition table

i/p States	0	1
→ S_0	S_2	S_1
S_1	S_2	S_1
S_2	S_2	S_1



ex:

Design FA which checks whether the given una number is divisible by 3.

- The unary number is made up of one's
- The number 3 can be written in unary form as 111

→ 5 can be written as 11111

→ Consider a number 111111 which is equal to 6 divisible by 3.

→ So complete scan of this number we reach to final state q_3.

Start 111111

State q_0

1 q_1 11111

11 q_2 1111

111 q_3 111

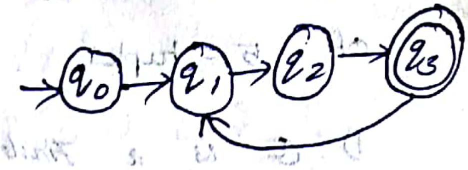
1111 q_1 11

11111 q_2 1

111111 q_3 — No we are in final state.

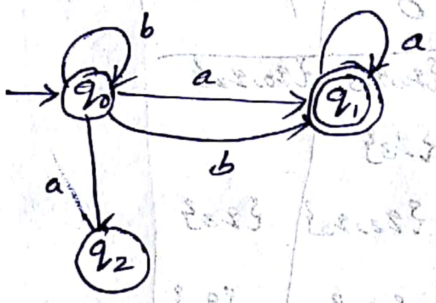
Scanned by CamScanner

input	1
→ q_0	q_1
q_1	q_2
q_2	q_3
q_3	q_1



NDFA or NFA

- It is exactly reverse of DFA.
- When there exists many paths For a Specific input from current State to next state.



- NFA shows from q_0 For i/p a there are two next states q_1 and q_2.
- Similarly from q_0 for input b the next states are q_0 and q_1.

Consider the input String bba

i/p b b a
Path q_0 q_0 q_1

i/p b b a
q_0 q_0 q_2

i/p b b a
q_0 q_1 q_1

The NFA can be formally defined as a collecti of 5-tuples.

1) Q is a finite set of states

2) Σ is a finite set of inputs

3) δ is called next state (or) transition function

4) q_0 is initial state

5) F is a final state $F \subseteq Q$.

example

Design the NFA transition diagram for the transition table as given below

	0	1
q_0	$\{q_0, q_1\}$	$\{q_0, q_2\}$
q_1	$\{q_3\}$	
q_2	$\{q_2, q_3\}$	$\{q_3\}$
q_3	$\{q_3\}$	$\{q_3\}$

Here the NFA is $N = (\{q_0, q_1, q_2, q_3\}, \delta, q_0, \{q_3\})$

$\delta(q_0, 0) = \{q_0, q_1\}$

$\delta(q_0, 1) = \{q_0, q_2\}$

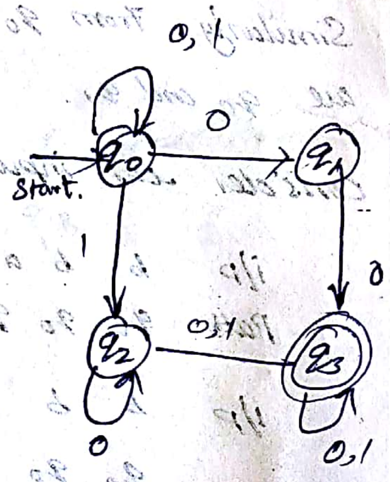
$\delta(q_1, 0) = \{q_3\}$

$\delta(q_2, 0) = \{q_2, q_3\}$

$\delta(q_2, 1) = \{q_3\}$

$\delta(q_3, 0) = \{q_3\}$

$\delta(q_3, 1) = \{q_3\}$



Scanned by CamScanner

ex: Construct a transition diagram for the NDFA $M = (\{q_1, q_2, q_3\}, \delta, q_1, \{q_3\})$ where δ is given by.

$\delta(q_1, 0) = \{q_2, q_3\}$, $\delta(q_1, 1) = \{q_1\}$

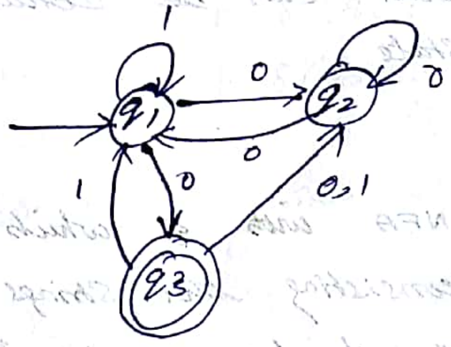
$\delta(q_2, 0) = \{q_1, q_2\}$ $\delta(q_2, 1) = \Phi$

$\delta(q_3, 0) = \{q_2\}$ $\delta(q_3, 1) = \{q_1, q_2\}$

We will design a transition table using the given mapping function δ.

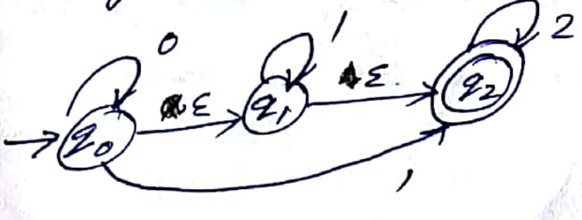
States \ i/p	0	1
$\rightarrow q_1$	$\{q_2, q_3\}$	q_1
q_2	$\{q_1, q_2\}$	Φ
(q_3)	q_2	$\{q_1, q_2\}$

The NDFA will be



Finite Automata with ε moves

- The ε-transitions (epsilon) in NFA are given in order to move from one state to another without having any symbol from i/p set Σ.



Significance of NFA with ε

- As construction of DFA is very difficult for certain input set. we construct NFA.

- ε is an empty String

- The ε- transitions are used simply to change one state to other.

- Sometimes to reach to Final state we do not get proper state from start state.

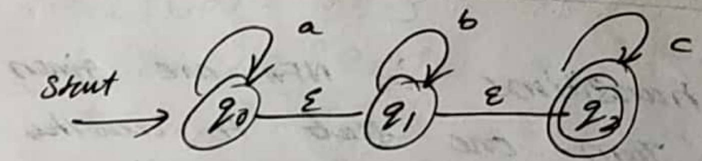
- In such a case we simply want to reach to certain state which leads to Final State.

- such a transition to their specific state should be without any i/p symbol.

- So we require some ε- moves by which a proper state can be obtained for reaching to Final state.

ex:

construct NFA with ε which accepts a language consisting the strings of any number of a's followed by any number of b's followed by any number of c's.



State \ I/P	a	b	c	ε
q_0	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
q_2	ϕ	ϕ	q_2	ϕ

We can parse the string aabbcc

$\delta(q_0, aabbcc) \vdash \delta(q_0, abbcc)$

$\vdash \delta(q_0, bbcc)$

$\vdash \delta(q_0, \varepsilon bbcc)$

$\vdash \delta(q, bbcc)$

$\vdash \delta(q_1, bcc)$

$\vdash \delta(q_1, cc)$

$\vdash \delta(q_1, \varepsilon cc)$

$\vdash \delta(q_2, cc)$

$\vdash \delta(q_2\ c)$

$\vdash \delta(q_2, \varepsilon)$

We reach to accept state, after scanning the compl i/p string.

ε - closure

The ε-closure(p) is a set of all states which are reachable from state p on ε- transition

(i) ε-closure(p) = p where $p \in Q$.

(ii) if there exists ε - closure(p) = {q} and δ(q,ε) = r then ε - closure(p) = {q, r}

Scanned by CamScanner

Regular Expression:- It used to denote regular language. It accepted by Finite automata.

Let Σ be an alphabet which is used to denote the input set. The regular expression over Σ can be denoted as

φ:

1. Φ is a regular expression which denotes the empty set.

epsilon

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

3. If r and s are regular expressions denoting the languages L_1 and L_2 respectively

$r+s$ is equivalent to $L_1 \cup L_2$ union

rs is equivalent to $L_1 L_2$ concatenation

r^* is equivalent to L_1^* closure.

The r^* is known as kleen closure. which indicates occurrence of r for ∞ number of time)

ex: if $\Sigma = \{a\}$ and we have regular expression $R = a^*$.

Then R is a set denoted by

$$R = \{\varepsilon, a, aa, aaa, aaaa, \ldots\}$$

indicat zero no of a by using ε charakter.

es' :

Design the regular expression (r.e) for the language accepting all combinations of a's except the null string over $\Sigma = \{a\}$.

language $L = \{a, aa, aaaa, ..\}$.

We can denote $\overset{\text{regular ex}}{\underline{\text{r.e}}}$ as

$R = a^+$ - Positive closure indicate the set of strings without $\overset{}{\text{a}}$ null strings

Design regular expression for the language containing all the strings containing any no of a's and b's

The regular expression will be

r.e $= (a+b)^*$

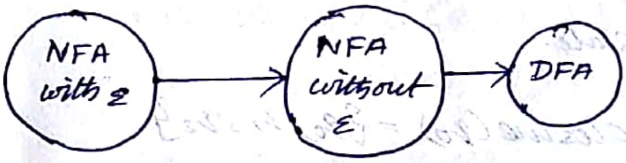
$L = \{\varepsilon, a, aa, ab, b, ba, bab, abab \cdots\}$

The $(a+b)^*$ means any combination with a and b even a null string.

Equivalence of NFA and DFA

- The NFA with ε can be converted to NFA without ε.

- NFA without ε can be converted to DFA.



. Moves to DFA

Scanned by CamScanner

Conversion of NDFA with ε to NDFA without ε

- Here we try to remove all the ε transitions from given NFA.

1. Find out all the ε transitions from each state from Q. That will be called as ε-closure {q_i} where $q_i \in Q$.

2. The δ' transitions can be obtained. The δ' transitions means an ε-closure on δ moves.

3. Step-2 is repeated for each input symbol and for each state of given NFA.

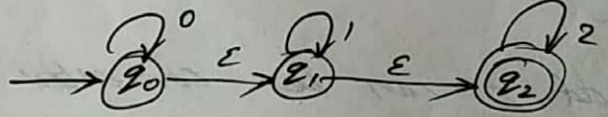
4. Using the resultant states the transition table for equivalent NFA without ε can be build.

Rule for conversion

$$\delta'(q,a) = \varepsilon\text{-closure}(\delta(\hat{\delta}(q,\varepsilon),a))$$

where $\hat{\delta}(q,\varepsilon) = \varepsilon\text{-closure}(q)$

ex: convert the given NFA with ε to NFA without ε.



We will find out ε-reachable states from current state.

ε- closure(q_0) = {q_0, q_1, q_2}

ε- closure(q_1) = {q_1, q_2}

ε- closure(q_2) = {q_2}

Scanned by CamScanner

$\delta'(q_0,0) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_0,\epsilon),0))$

$= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0),0))$

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

$= \epsilon\text{-closure}(\delta(q_0,0) \cup \delta(q_1,0) \cup \delta(q_2,0))$

$= \epsilon\text{-closure}(q_0 \cup \phi \cup \phi)$ empty set

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

$\delta'(q_0,0) = \{q_0,q_1,q_2\}$

$\delta'(q_0,1) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_0,\epsilon),1))$

$\epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0),1))$

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

$= \epsilon\text{-closure}(\delta(q_0,1) \cup \delta(q_1,1) \cup \delta(q_2,1))$

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

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

$\delta'(q_0,1) = \{q_1,q_2)$

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

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

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

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

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

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

$= \phi$

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

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

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

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

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

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

$= \{q_1, q_2\}$

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

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

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

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

$= \phi$

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

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

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

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

$\delta'(q_2,1) = \phi$

$\delta'(q_0,2) = \varepsilon\text{-closure}(\delta(\hat{\delta}(q_0,\varepsilon),2))$

$= \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_0),2))$

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

$= \varepsilon\text{-closure}(\delta(q_0,2) \cup \delta(q_1,2) \cup \delta(q_2,$

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

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

$= \{q_2\}$

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

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

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

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

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

$= \{q_2\}$

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

Now we will summorize all the computed δ' transition.

$\delta'(q_0,0) = \{q_0,q_1,q_2\}$, $\delta'(q_0,1) = \{q_1,q_2\}$

$\delta'(q_0,2) = \{q_2\}$

$\delta'(q_1,0) = \phi$, $\delta'(q_1,1) = \{q_1,q_2\}$, $\delta'(q_1,2] = \{q_2\}$

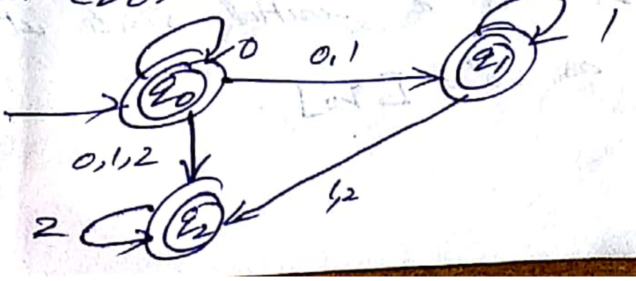
$\delta'(q_2,0) = \phi$ $\delta'(q_2,1) = \phi$ $\delta'(q_2,2) = \{q_2\}$

Transition table

I/P State	0	1	2
q_0	$\{q_0,q_1,q_2\}$	$\{q_1,q_2\}$	$\{q_2\}$
q_1	ϕ	$\{q_1,q_2\}$	$\{q_2\}$
q_1	ϕ	ϕ	$\{q_2\}$

NFA will be

Here q_0, q_1 & q_2 is a Final State because ε-closure (q_0) & q_1 & q_2 contain Final State q_2



CONVERSION OF NFA to DFA

- The finite automata can either be DFA or NFA

- who is better NFA or DFA

- which has more power?

- Here is a theorem which tell you that any NFA can be converted to its equivalent DFA

Theorem: Let L be a set accepted by non-deterministic finite automation. Then there exi a deterministic finite automation that accept L.

(or)

Prove that "A language L is accepted by some DFA if and only if L is accepted by some NFA.

Proof: Let

$M = (Q, \Sigma, \delta, q_0, F)$ be an NFA for language L.

Then define DFA M' such that

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

The ~~NFA~~ Q' will be denoted by $[q_1, q_2, q_3 \ldots q_i]$

The NFA q_0 is a initial state it is denoted in DFA as $q_0' = [q_0]$

$$\delta'([q_1, q_2, q_3, \cdots q_i], a) = [P_1, P_2, P_3 \cdots P_j].$$

At the current state $[q_1, q_2 \cdots q_1\}$ if we get Enput a and it goes to the next state $[P_1, P_2, P_3 \cdots P_j]$.

→ The theorem can be moved with the induction method by assuming length of i/p string x :

$$\delta'(q_0', x) = [q_1, q_2 \cdots q_i]$$

if and only if

$$\delta(q_0, x) = \{q_1, q_2 \cdots q_i\}$$

Basis : if length of i/p string is 0

$|x| = 0$, that means x is ε then $q_0' = [q_0]$

Determine DFA from given NFA.

ex: Let $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$ be NFA where $\delta(q_0, 0) = \{q_0, q_1\}$, $\delta(q_0, 1) = \{q_1\}$ $\delta(q_1, 0) = \Phi$, $\delta(q_1, 1) = \{q_0, q_1\}$

construct its equivalent DFA.

Solution:

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

The δ' function will be computed as

$$\delta(q_0, 0) = \{q_0, q_1\} = \delta'([q_0], 0) = [q_0, q_1].$$

Scanned by CamScanner

NFA the initial state is q_0, the DFA initial state contains $[q_0]$.

i/p State	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_1\}$
$\rightarrow q_1$	ϕ	$\{q_0, q_1\}$

The above transition table ~~can~~ we can compute $[q_0], [q_1], [q_0, q_1]$ states for its equivalent DFA.

now we need to compute $[q_0, q_1]$

$$\delta(\{q_0, q_1\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0)$$
$$= \{q_0, q_1\} \cup \phi$$
$$= \{q_0, q_1\}$$

$$\delta'([q_0, q_1], 0) = [q_0, q_1]$$

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

$$\delta'([q_0, q_1], 1) = [q_0, q_1]$$

Now NFA q_1 is a final state. Hence in the DFA whenever q_1 exists that state becomes a final state.

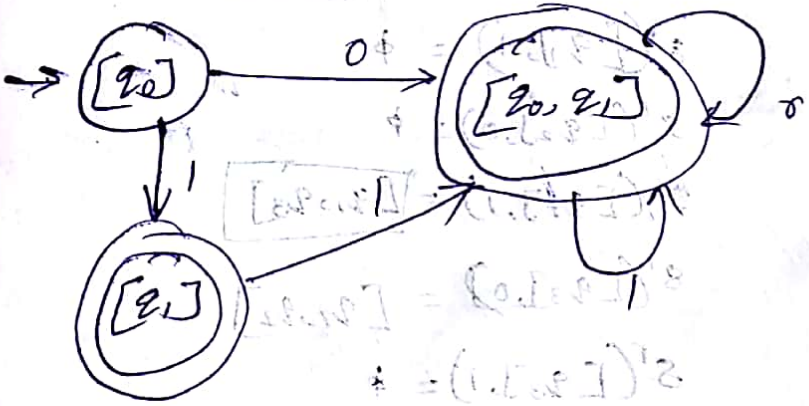
Scanned by CamScanner

Here in the DFA Final states are [q1] [q0, q1]

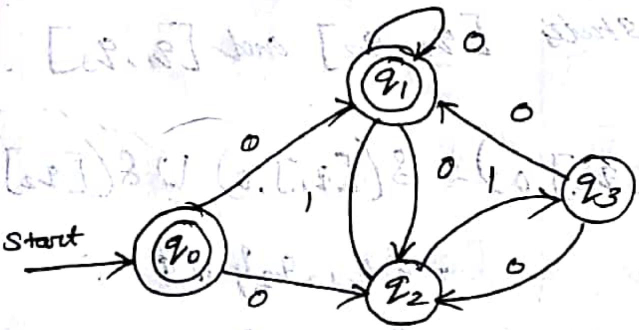
$F = \{[q_1], [q_0, q_1]\}$

The equivalent DFA is :

I/P State	0	1
$[q_0]$	$[q_0 . q_1]$	$[q_1]$
$[q_1]$	ϕ	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$



Ex : Convert the following NFA



State \ i/p	0	1
q_0	$\{q_1, q_2\}$	ϕ
q_1	$\{q_1, q_2\}$	ϕ
q_2	ϕ	$\{q_1, q_3\}$
q_3	$\{q_1, q_2\}$	ϕ

Now $\delta(q_0, 0) = \{q_1, q_2\}$

we can write this as

$$\delta'([q_0], 0) = \boxed{[q_1, q_2]}$$

$$\delta'([q_0], 1) = \phi$$

$$\delta'([q_1], 0) = \boxed{[q_1, q_2]}$$

$$\delta'([q_1], 1) = \phi$$

$$\delta'([q_2], 0) = \phi$$

$$\delta'([q_2], 1) = \boxed{[q_1, q_3]}$$

$$\delta'([q_3], 0) = [q_1, q_2]$$

$$\delta'([q_3], 1) = \phi$$

Now we will obtain δ' transitions on newly generated states $[q_1, q_2]$ and $[q_1, q_3]$.

$$\delta'([q_1, q_2], 0) = \delta([q_1], 0) \cup \delta([q_2], 0)$$

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

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

$$\delta'([q_1, q_2], 0) = [q_1, q_2]$$

Scanned by CamScanner

$\delta'([q_1, q_2], 1) = \delta([q_1], 1) \cup \delta([q_2], 1)$

$= \phi \cup \{q_1, q_3\}$

$= \{q_1, q_3\}$

$= [q_1, q_3]$

$\delta'([q_1, q_3], 0) = \delta([q_1], 0) \cup \delta([q_3], 0)$

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

$= \{q_1, q_2\}$

$= [q_1, q_2]$

$\delta'([q_1, q_3], 1) = \delta([q_1], 1) \cup \delta([q_3], 1)$

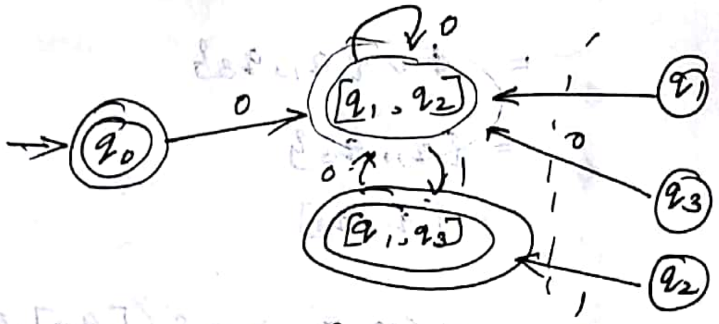
$= \phi \cup \phi$

$= \phi$

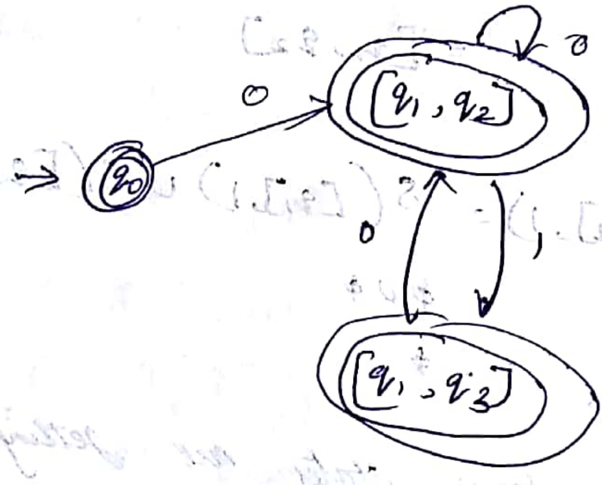
As now no new states are getting generated.

I/P State	0	1
$[q_0]$	$[q_1, q_2]$	ϕ
$[q_1]$	$[q_1, q_2]$	ϕ
$[q_2]$	ϕ	$[q_1, q_3]$
$[q_3]$	$[q_1, q_2]$	~~$[q_1, q_3]$~~ ϕ
$[q_1, q_2]$	$[q_1, q_2]$	$[q_1, q_3]$
$[q_1, q_3]$	$[q_1, q_2]$	ϕ

The transition diagram can be



→ q_1, q_2, q_3 is no use there is no path from start state to these state so we eliminate these states.



As states q_0 and q_1 are final states original NFA. So $[q_1, q_2]$ and $[q_1, q_3]$ are also final states.

Conversion of NFA with ε to DFA

- In this method we first convert NFA with ε to NFA without ε. Then the NFA without ε can be converted to its equivalent DFA

Step 1: Consider $M = (Q, \Sigma, \delta, q_0, F)$ is a NFA with ε. we have to convert this NFA with ε to equivalent DFA denoted by.

$$M_D = (Q_D, \Sigma, \delta_D, q_0, F_D)$$

Then obtain

$\varepsilon\text{-closure}(q_0) = \{P_1, P_2, P_3 \ldots P_n)$ becomes a Start State of DFA.

Now $[P_1, P_2, P_3 \ldots P_n] \in Q_D$

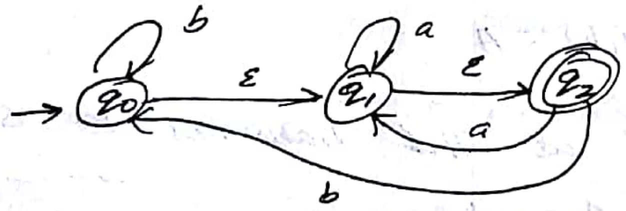
Step 2:

We will obtain δ transition on $[P_1, P_2, P_3 \ldots P_n]$ for each input.

$$\delta_D([P_1, P_2 \ldots P_n], a) = \varepsilon\text{-closure}(\delta(P_1, a) \cup \delta(P_2, a) \cup \ldots \delta(P_n, a))$$

Step 3: = The States obtained $[P_1, P_2, P_3 \ldots P_n] \in Q_D$. The States containing Final State in P_i is a Final State in DFA.

ex: Convert the following NFA with ε to equivalent DFA



Solution: To convert this NFA we will First Find ε-closures.

ε - closures $\{q_0\} = \{q_0, q_1, q_2\}$

ε - closures $\{q_1\} = \{q_1, q_2\}$

ε - closures $\{q_2\} = \{q_2\}$

ε-closures $\{q_0\} = \{q_0, q_1, q_2\}$ we will call the state as A.

$\delta'(A, a) = \varepsilon\text{-closure}\,(\delta(A,a))$

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

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

$= \varepsilon\text{-closure}\,\{q_1\}$

$= \{q_1, q_2\}$ Let us call it as state B

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

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

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

$= \varepsilon\text{-closure}\,\{q_0\}$

$= \{q_0, q_1, q_2\}$ i.e A.

$\delta'(A, a) = B$

$\delta'(A, b) = A$

Now let us find transitions for state

$B = \{q_1, q_2\}$

$\delta'(B, a) = \varepsilon\text{-closure}(\delta(q_1, q_2), a)$

$= \varepsilon\text{-closure}\,\{q_1\}$

$= \{q_1, q_2\}$ i.e B

$\delta'(B, b) = \varepsilon\text{-closure}\,(\delta(q_1, q_2), b)$

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

$= \varepsilon\text{-closure}\,\{q_0\}$

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

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

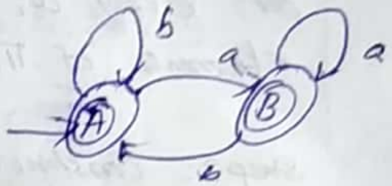
(q_1)

$= \varepsilon\text{-closure}$

$= (q_0, q_1, q_2)$ i.e. A

Hence the generalized DFA is

	a	b
(A)	B	A
(B)	B	A
(C)	B	A



$\delta'(C, b) = (\delta(q_2), b)$

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

$\varepsilon\text{-closure}\,(q_0)$

$= (q_0, q_1, q_2)$ i.e. A

Minimization of DFA

- The minimization of FSM means reducing the number of states from given FA.

- While minimizing FSM we first find out which two states are equivalent we can represent those two states by one representative state.

Method for construction of Minimum State Automata

Step 1: We will create a set Π_0 as $\Pi_0 = \{Q_1^0, Q_2^0\}$ where Q_1^0 is set of all final state and $Q_2^0 = Q - Q_1^0$ where Q is a set of all the state in DFA.

Step 2: We will construct Π_{K+1} from Π_K. Let Q_i^k be any subset in Π_k. If q_1 and q_1 are in Q_i^k they are $(k+1)$ equivalent provided $\delta(q_1, a)$ and $\delta(q_2, a)$ are k equivalent.

Scanned by CamScanner

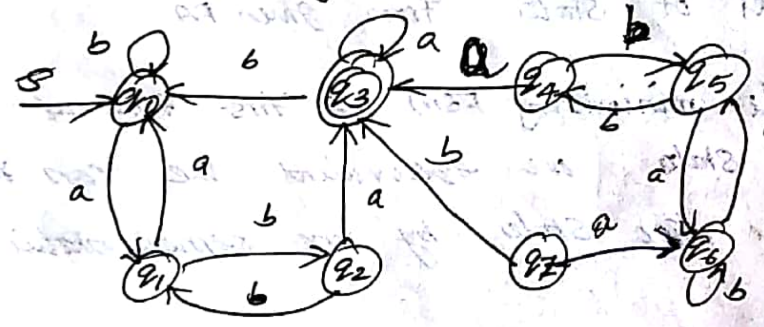
Find out whether $\delta(q_1, a)$ and $\delta(q_2, a)$ are residing in the same equivalence class π_k.

Then it is said that q_1 and q_2 are $(k+1)$ equivalent. Thus Q_i^k is further divided into $(k+1)$ equivalence classes. Repeat step 2 for every Q_i^k in π_k and obtain all the elements of π_{k+1}

Step3: Construct π_n for $n = 1, 2 \ldots$ until $\pi_n = \pi_{n+1}$

step4: Then replace all the equivalent states in one equivalence class by representative state.

ex: construct the minimum state automaton for the following transition diagram.



Solution

I/p State	a	b
q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
q_3	q_3	q_0
q_4	q_3	q_5
q_5	q_6	q_4
q_6	q_5	q_6
q_7	q_6	q_3

There is only one Final State i.e q_3. Hence we can Partition Q as Q_1^0 and Q_2^0

$Q_1^0 = F = \{q_3\}$

$Q_2^0 = Q - Q_1^0$

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

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

Creating π_1

Compare q_0 with q_2

	a	b
q_0	q_1	q_0
q_2	q_3	q_1

Under a - column of q_0 & q_2 we get q_1 & q_3 respectively. But q_1 & q_3 in different States.

Similarly q_0 & q_4

	a	b
q_0	q_1	q_0
q_4	q_3	q_5

→ Under - a-column of q_0 & q_4 we get q_1 & q_3 which in different sets. Hence q_0 is not 1-equivalent to q_4.

→ q_0 is not 1 - equivalent to q_7 (b-column of q_0 & q_7).

→ But q_0 is 1 - equivalent to q_1, q_5 & q_6.

$Q_1^1 = \{q_3\}$

$Q_2^1 = \{q_0, q_1, q_5, q_6\}$

q_2 is 1 - equivalent to q_4

$Q_3^1 = \{q_2, q_4\}$

Since under a-column we get q_3 only for q_2 & q_4.

⇒ under b- colum of q_2 & q_4 we get q_1 & q_5.

⇒ q_1 & q_5 lie in one set. Thus q_2 & q_4 are 1 - equivalent.

The only element left over in Q_2^0 is q_7.

∴ $Q_4^1 = \{q_7\}$

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

Creating π_2

$Q_1^2 = \{q_3\}$

Now q_0 is 2 - equivalent to q_6

	a	b
q_0	q_1	q_0
q_6	q_6	q_6

same set

but q_0 is not 2 - equivalent to q_1 & q_5

	a	b
q_0	q_1	q_0
q_1	q_0	q_2

	a	b
q_0	q_1	q_0
q_5	q_6	q_4

Dot nt lie in same Set.

$Q_2^2 = \{q_0 q_6\}$

but q_1 is 2- equivalent to q_5.

Hence $Q_3^2 = \{q_1, q_5\}$

Similarly q_2 is 2- equivalent to q_4

$Q_4^2 = \{q_2, q_4\}$

$Q_5^2 = \{q_7\}$

	a	b
q_1	q_0	q_2
q_5	q_6	q_4

Lie in one set

Lie in one set

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

Creating π_3

$Q_1^3 = \{q_3\}$

As q_0 is 3- equivalent to q_6

$Q_2^3 = \{q_0, q_6\}$

As q_1 is 3- equivalent to q_5,

$Q_3^3 = \{q_1, q_5\}$

As q_2 is 3- equivalent to q_4

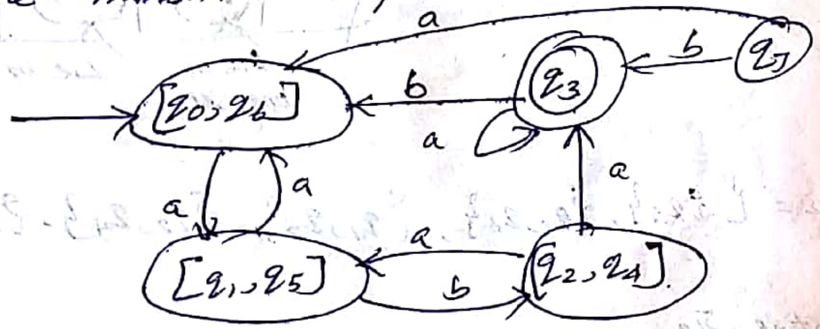
$Q_4^3 = \{q_2, q_4\}$ & $Q_5^3 = \{q_7\}$

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

As $\pi_3 = \pi_2$ we have got equivalence class from π_2.

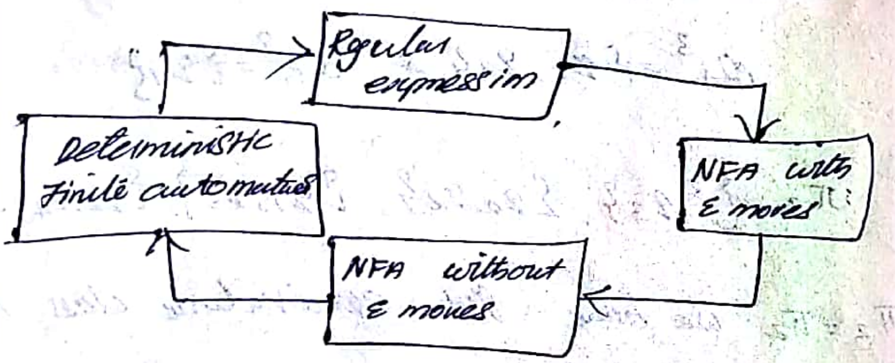
4A	a	b
$[q_0,q_6]$	$[q_1,q_5]$	$[q_0,q_6]$
$[q_1,q_5]$	$[q_0,q_6]$	$[q_2,q_4]$
$[q_2,q_4]$	$[q_3]$	$[q_1,q_5]$
$[q_3]$	$[q_3]$	$[q_0,q_6]$
$[q_7]$	$[q_0,q_6]$	$[q_3]$

The transition diagram with minimized States is



Equivalence of Finite Automation and Regular Expressions.

- There is a close relationship b/w a Finite automata and the regular expression



Relationship b/w FA and regular expression

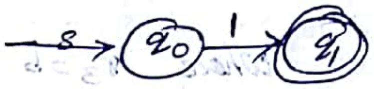
Equivalence of NFA and Regular Expression

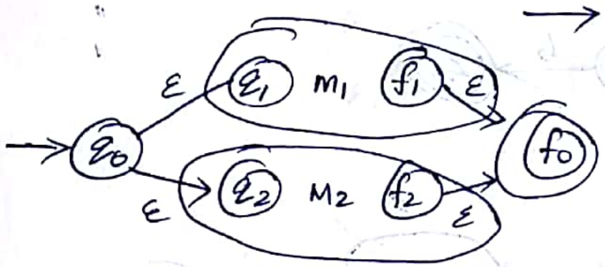
Here we are going to using three cases to construct NFA for the Regular Expression

(i) Union case

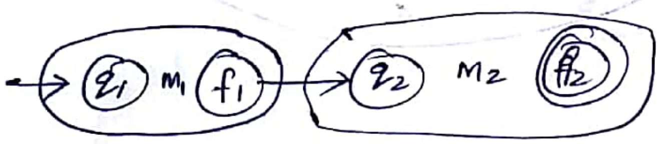
(ii) Concatenation case

(iii) Closure case

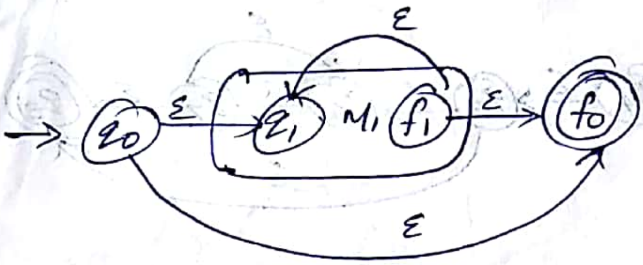




Unione:



concatenation



Closure case:

ex:

construct NFA for the regular expression $b+ba^*$

Solution: The regular expression

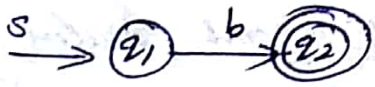
$r = b+ba^*$ can be broken into r_1 & r_2

$r_1 = b$

$r_2 = ba^*$

4b

$r_1 \Rightarrow$

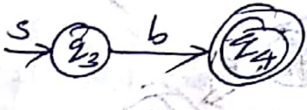


We will go for $r_2 = ba^*$ this can be broken into r_3 and r_4

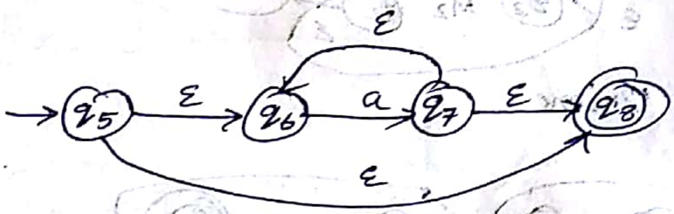
where $r_3 = b$

$r_4 = a^*$

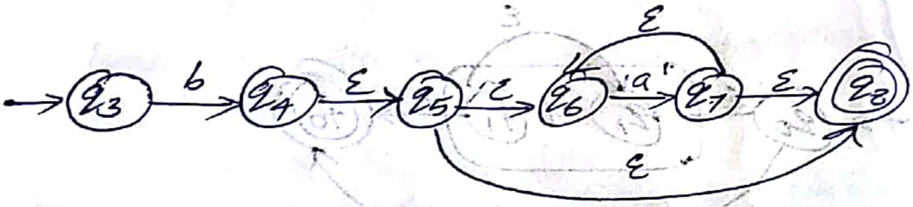
$r_3 \Rightarrow$



$r_4 \overset{a^*}{\Rightarrow}$

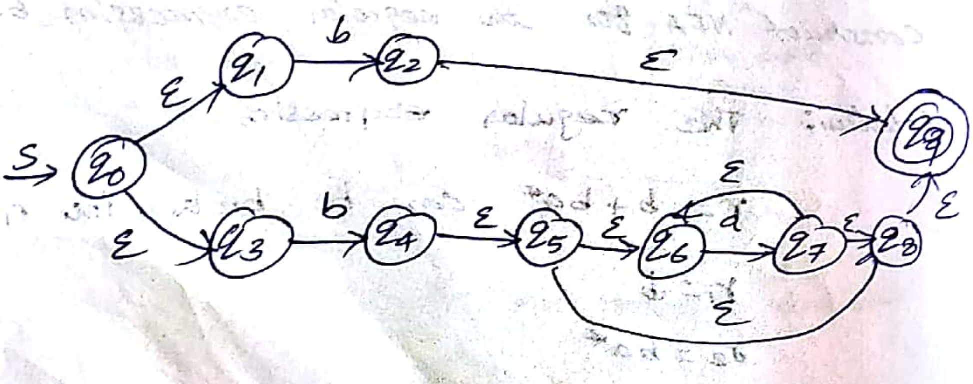


r_2 will be $r_2 = r_3 . r_4$ Concadination.



Now we will draw NFA $r = r_1 + r_2$

$r = b+ba^*$



ex:2

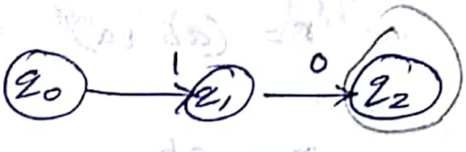
Construct an NFA equivalent to the following regular expression $((10)+(0+1)^*)\ 01$.

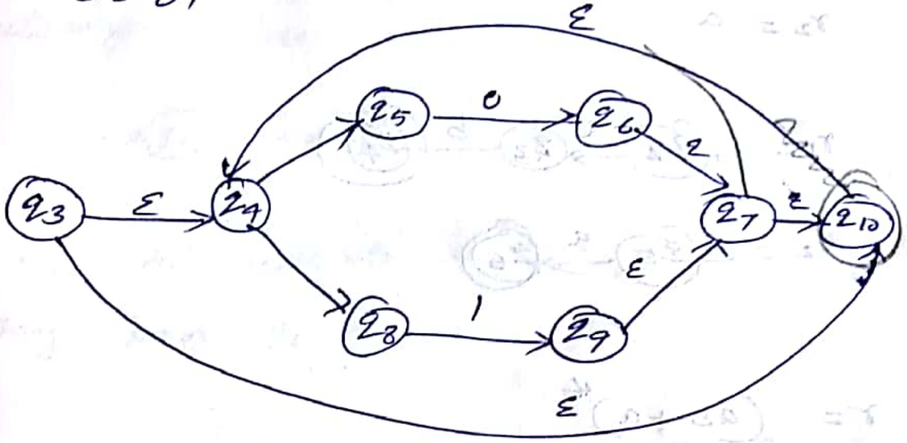
$r.e = (r_1+r_2)\,r_3$ concadin

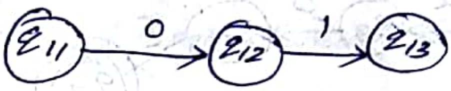
$r_1 = 10$

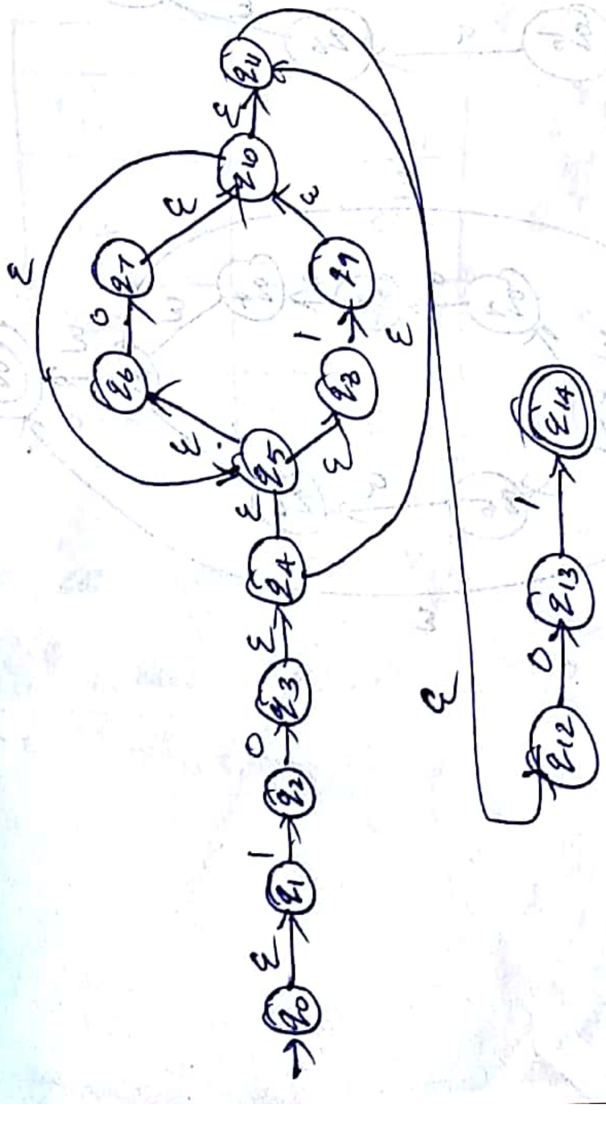
$r_2 = (0+1)^*$

$r_3 = 01$









ex:3 48

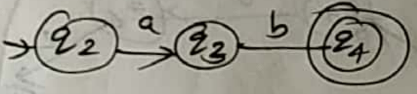
Construct Finite Automata equivalent to the regular expression $(ab+a)^*$

$r = (ab+a)^*$

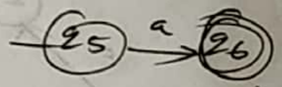
$r_1 = ab$

$r_2 = a$

$r_1 =$

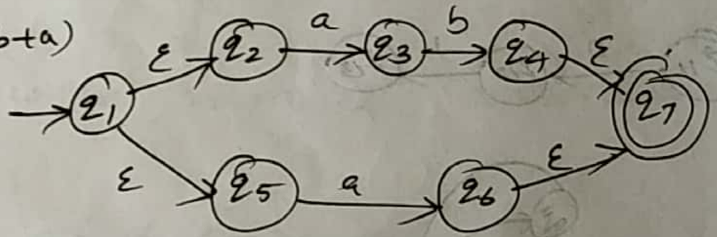


$r_2 =$

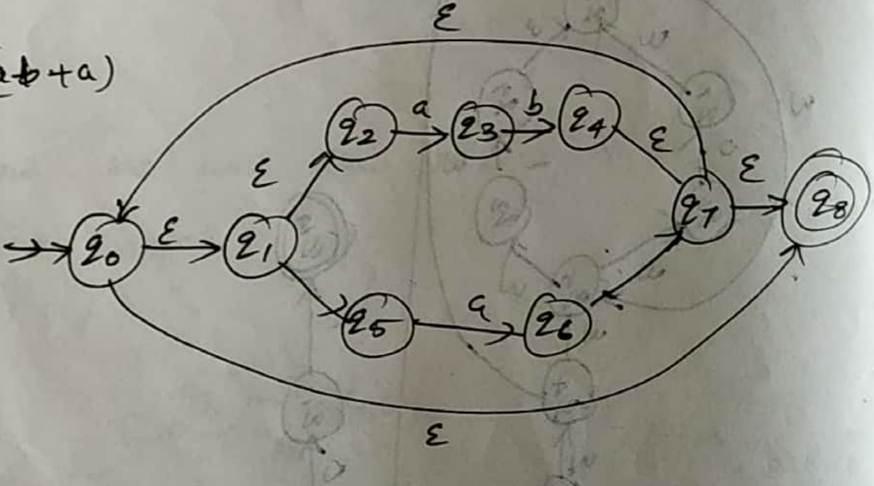


$r = (ab+a)^*$

(ab+a)



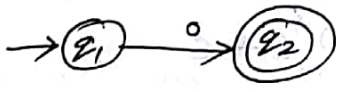
r=(ab+a)



Conversion of Finite Automata to Regular Expression

$$(r.e) = r_{ij}^{k} = (r_{ik}^{k-1})(r_{kk}^{k-1})^{*}(r_{kj}^{k-1}) + r_{ij}^{k-1}$$

① Construct the regular expression for the Finite automata given



→ r_{ij} will indicate the set of all the i/p string from qi to j.

→ if $i=j$ then we add ε with the i/p string.

→ if $i \neq j$ and there is no path from qi to qi then we add φ

	$k=0$
r_{11}	ε
r_{12}	0
r_{21}	φ
r_{22}	ε

$r_{11}^{0} = ε$

$r_{12}^{0} = 0$

$r_{21}^{0} = φ$

$r_{22}^{0} = ε$

Let us compute r_{11}^{0}

r_{11}^{0} where $i=1, j=1; k=0$

Let us build the table when $k=1$

r_{11}^{1} $\quad$ $r_{ij}^{k} = r_{ik}^{k-1}(r_{kk}^{k-1})^*(r_{kj}^{k-1}) + r_{ij}^{k-1}$

$i=1,\ j=1,\ k=1$

$$r_{11}^{1} = r_{11}^{0}(r_{11}^{0})^{*} r_{11}^{0} + r_{11}^{0}$$

$$= \varepsilon(\varepsilon)^{*}\varepsilon + \varepsilon$$

$$r_{11}^{1} = \varepsilon$$

r_{12}^{1} $\quad$ $i=1,\ j=2 \quad k=1$

$$r_{12}^{0} = r_{11}^{0}(r_{11}^{0})^{*}(r_{12}^{0}) + r_{12}^{0}$$

$$= \varepsilon(\varepsilon)^{*}(0)^{0} + 0$$

$$= \varepsilon . 0 + 0$$

$$= 0 + 0$$

$$= 0$$

r_{21}^{1} $\quad$ $i=2,\ j=1 \quad k=1$

$$r_{21}^{1} = r_{21}^{0}(r_{11}^{0})^{*}(r_{11}^{0}) + r_{21}^{0}$$

$$= \phi(\varepsilon)^{*}(\varepsilon) + \phi$$

$$= \phi + \phi$$

$$= \phi$$

r_{22}^1 $i=2,\ j=2$ $k=1$

$$r_{22}^1 = r_{21}^0 (r_{11}^0)^* (r_{12}^0) + r_{22}^0$$

$$= \phi.(\varepsilon)^*(0) + \varepsilon$$

$$= \phi + \varepsilon$$

$$= \varepsilon .$$

Now let us compute for Final state.

→ r_{12}^2 will be computed, because there are total 2 states and Final state is q_2 where start is q_1,

$$r_{12}^2 = (r_{12}^1)(r_{22}^1)^*(r_{22}^1) + (r_{12}^1)$$

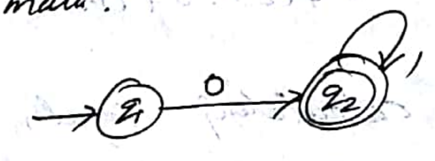
$\boxed{\varepsilon^* = 0}$

$$= 0(\varepsilon)^*(\varepsilon) + 0$$

$$= 0 + 0$$

$r_{12}^2 = 0$ which is a Final regular expression.

② construct regular expression From given Finite automata.



Let us compute when $k=0$

	$k=0$
r_{11}	ε
r_{12}	0
r_{21}	ϕ
r_{22}	$1+\varepsilon$

Now we will calculate for $k=1$

$k=1$

r_{11} $i=1,\ j=1,\ k=1$

$$r_{11}^{1} = r_{11}^{0}\,(r_{11}^{0})^{*}\,(r_{11}^{0}) + r_{11}^{0}$$

$\varepsilon^{*/0}$

$$= \varepsilon\,.\,(\varepsilon)^{*}.\ \varepsilon + \varepsilon$$

$$r_{11}^{1} = \varepsilon$$

r_{12} $i=1,\ j=2,\ k=1$

$$r_{12}^{1} = r_{11}^{0}\,(r_{11}^{0})^{*}\,r_{12}^{0} + r_{12}^{0}$$

$$= \varepsilon\,.\,0 + 0$$

$$r_{12}^{1} = 0$$

r_{21} $i=2,\ j=1,\ k=1$

$$r_{21}^{1} = r_{21}^{0}\,(r_{11})^{*}\,r_{11}^{0} + r_{21}^{0}$$

$$= \phi\,.\ \varepsilon\,.\ \varepsilon + \phi$$

$$r_{21}^{1} = \phi$$

r_{22} $i=2,\ j=2,\ k=1$

$$r_{22}^{1} = r_{21}\,(r_{11})^{*}\ r_{12}^{0} + r_{22}^{0}$$

$$= \phi(\varepsilon)^{*}.\ 0 + (1+\varepsilon) = 1+\varepsilon$$

Now for calculating regular expression we should compute for the path from start state to final. That is from q_1 to q_2

$1+\varepsilon = 1$ $\qquad P=1,\ j=2,\ k=2$

$(1+\varepsilon)^{*} = 1^{*}$

$$r_{12}^{2} = r_{12}^{1}\,(r_{22}^{1})^{*}\ r_{22}^{1} + r_{12}^{1}$$

$$= (0.(1+\varepsilon)^{*}(1+\varepsilon)) + 0$$

$= *$

$$r_{12}^{2} = 01^{*} + 0$$

Scanned by CamScanner

Arden's Method for converting DFA to RE

- The Arden's theorem is useful for checking the equivalence of two regular expression as well as in conversion of DFA to rw.

Algorithm:

1. Let q_1 be the initial state.

2. There are $q_2, q_3, q_4, \dots q_n$ number of states. The final state may be some q_j where $j \leq n$.

3. Let α_{ji} represents the transition from q_j to q_i.

4. Calculate q_i such that

$$\boxed{q_i = \alpha_{ji} \cdot q_j}$$

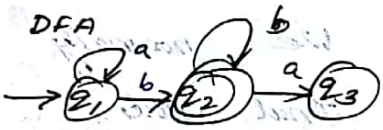
If q_i is a start state

$$\boxed{q_i = \alpha_{ji}\, q_j + \varepsilon}$$

5. Similarly compute the final state which ultimately gives the regular expression r.

ex: 1

Construct r.e from given DFA



Solution:

$$q_1 = q_1 a + \varepsilon$$

q_1 is a start state ε will be added.

$$q_2 = q_1 b + q_2 b$$

Scanned by CamScanner

Let us simplify q_1 first

$$q_1 = q_1 a + \varepsilon$$

We can re- write it as

$$q_1 = \varepsilon + q_1 a$$

$$\Uparrow$$

Similar to $R = Q + RP$

Gets reduced to $R = QP^*$

Assuming $R = q_1$, $Q = \varepsilon$, $P = a$

We get

$$q_1 = \varepsilon . a^* \quad \therefore \varepsilon . R = R$$

$$q_1 = a^*$$

Substituting value of q_1 in q_2

$$q_2 = q_1 b + q_2 b$$

$$q_2 = a^* b + q_2 b \qquad R = q_2, Q = a^* b, P = b$$

$$\therefore \quad q_2 = a^* \underline{b . b^*}$$

As $RR^* = R^+$

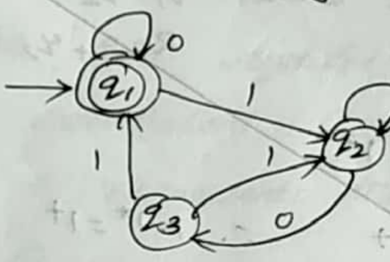
$$q_2 = a^* . b^+$$

We normally calculate the equation for final state.

$$q_2 = a^* b^+ .$$

ex: 2

Find out the regular expression from given DFA



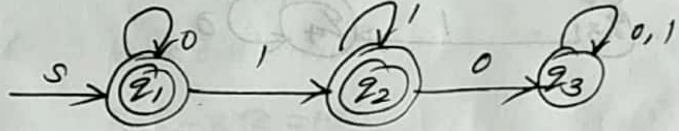
$q_1 = q_1 0 + q_3^{0+\varepsilon}$

$q_2 = q_2^{1} + q_3^{1} + q_1^{1}$

———x———x———x———x———x———

ex: 3

Construct r.e for the given DFA



$q_1 = q_1^{0+\varepsilon}$

$q_2 = q_1^{1} + q_2^{1}$

$q_3 = q_2^{0} + q_3^{(0+1)}$

Final states are q_1 & q_2, we are interested in solving q_1 & q_2 only.

$q_1 = \varepsilon + q_1^{0}$ $\qquad R = q_1$, $Q = \varepsilon$, $P = 0$

$R = Q + RP$, $\quad R = QP^*$

$q_1 = \varepsilon.(0)^*$

$q_1 = 0^*$ $\qquad \therefore \varepsilon.R = R.$

Substituting this value into q_2 $\qquad Q = 0^* 1$, $P = 1$

$q_2 = q_1^{1} + q_2^{1}$

$q_2 = 0^* 1 + q_2^{1}$ $\qquad R = Q + RP \Rightarrow QP^*$

$q_2 = 0^* 1 (1)^*$

The regular expression is given by

Here Two Final state q_1 q_2

$r = q_1 + q_2$ $\quad R = q_1 + q_2$

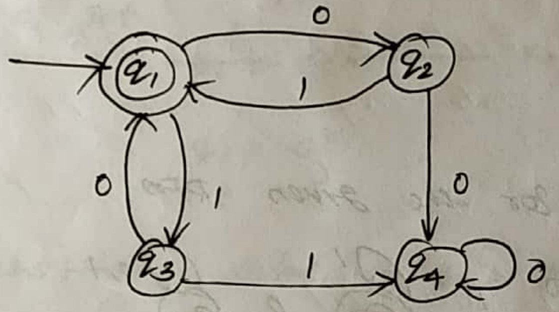
$= 0^* + 0^* 1 . 1^*$

$\therefore 1 . 1^* = 1^+$

$r = 0^* + 0^* 1^+$

ex:4

Construct r.e for the DFA given in



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

$q_2 = q_1 0$

$q_3 = q_1 1$

$q_4 = q_2 0 + q_3 1 + q_4 (0+1)$

Let us solve q_1 First

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

$q_1 = q_1 01 + q_1 10 + \varepsilon$

$q_1 = q_1 (01+10) + \varepsilon$ $\quad \therefore R = Q + RP \Rightarrow QP^*$

$q_1 = \varepsilon . (01+10)^*$ $\quad R = q_1, \; Q = \varepsilon$

$P = (01+10)$

$q_1 = (01+10)^*$ $\quad \boxed{\varepsilon . R = R}$

The Regular expression will be

$r = (01+10)^*$

Identity Rules

The two regular expression P and Q are equivalent (denoted as $P=Q$) if and only if P represents the same set of strings as Q

1. $\varepsilon R = R\varepsilon = R$

2. $\varepsilon^* = \varepsilon$

3. $(\phi)^* = \varepsilon$

4. $\phi R = R\phi = \phi$

5. $\phi + R = R$

6. $R + R = R$

7. $RR^* = R^*R = R^*$

8. $(R^*)^* = R^*$

9. $\varepsilon + RR^* = R^*$

10. $\varepsilon + R^* = R^*$

11. $\varepsilon + 0^* = 0^*$

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

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

Solution. LHS

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

we will take $(1+00^*1)$ as a common Factor.

$$1+00^*1 \; (\underbrace{\varepsilon + (0+10^*1)^*(0+10^*1)})$$

$(\varepsilon + R^*R)$ where $R=(0+10^*1)$

Scanned by CamScanner

$(\varepsilon + R^*R) = (\varepsilon + RR^*) = R^*$

$\therefore (1+00^*1)((0+10^*1)^*)$ out of this consid.

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

↓

Taking 1 as a common fact.

$(\varepsilon + 00^*)1\ (0+10^*1)^*$

Applying $\varepsilon + 00^* = 0^*$

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

= RHS.

---

ex 2: show that $(ab)^* \neq (a^*b^*)$

LHS = $(ab)^* = \{\varepsilon, ab, abab, ababab \ldots\}$

RHS $(a^*b^*) = \{\varepsilon, a, aa, aaa, b, bb, bbb, ab\}$

LHS ≠ RHS.

---

ex:

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

LHS

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

$= \{\varepsilon, 0, 00, 1, 11, 111, 01, 10 \ldots\}$

RHS = $(0+1)^*$

$= \{\varepsilon, 0, 00, 1, 11, 111, 01, 10 \ldots\}$

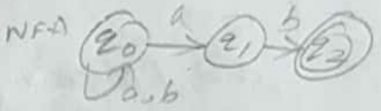
LHS = RHS.

Pumping Lemma 8or Regular Sets

- This is a basic and Important theorem used 8or checking whether given string is accepted by regular expression or not.

If 'L' is a regular language represented with an automation with a maximum of 'n' states.

then there is a word in 'L' such that (the length) $|z| \geq n$



$|A| \geq 3$ $n=3$ $z=abab = 4$

We may write $z = uvw$ in such a way that $|uv| \leq n$, $|v| \geq 1$ and for all $i \geq 0$, uv^iw is in L

loopin

↳ Some way the i/p is pumped.

Ex: 1

Pumping lemma to prove

$L = \{1^P \mid P \text{ as Prime number}\}$

is not regular.

Rule: ω = word (or) i/p $\in L$

$|Z| \geq n$

$Z = UVW \in L$

either $uv^iw \in L$

Step 1 : Assume given language L is regular.

Step 2 :

$$z = uvw$$

i.e $i \geq 0$ $\quad |uvw| = p \quad |v| \geq 1$

$$|uv^{i}\overset{w}{z}| \qquad i = p+1$$

$$|uv^{p+1}\underset{w}{\cancel{z}}| = |uv\cancel{w}| + |v|^{p}$$

$$= p + p(|v|)$$

$$|uv^{i}z| = p(1+|v|)$$

a prime no should not be multiple of two letter

prime no either multiple itself or 1 but here its not is 1, 1 + some value

Prove $L = \{a^n b^n / n \geq 1\}$ not regular

Step 1: Assume given language L is regular

step2: if L is regular then $z = uvw$ & $|z| \geq n$, $|v| \geq 1$ then $uv^{i}w \in L$

step3: $z = \underbrace{aa}_{x} \cdots \underbrace{a}_{y} \underbrace{bb \cdots b}_{z}$

$u\ \cancel{x} = a^{n-s}$

$v\ \cancel{y} = a^{s}$

$w\ \cancel{z} = b^{n}$

uv^iw $i \geq 0$

$a^{n-s}(a^s)^i b^n \in L$

$i = 2$

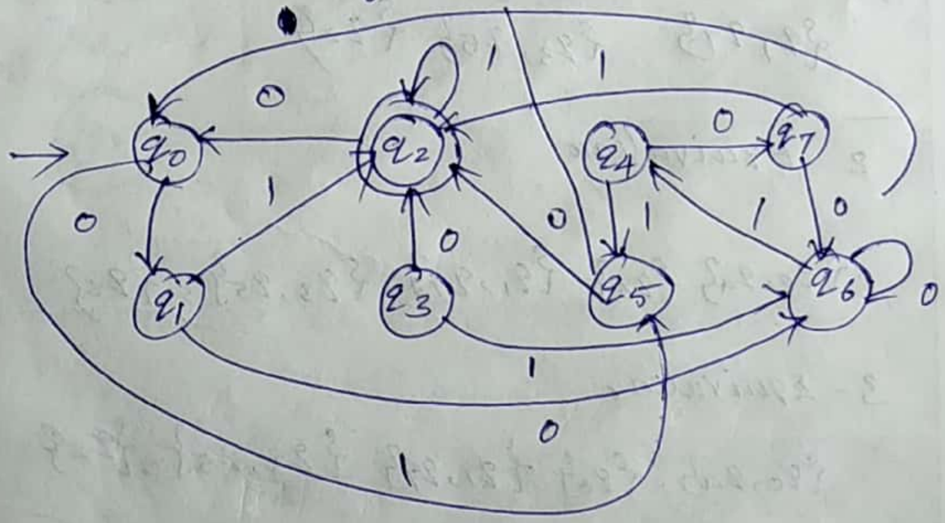
$a^{n-s} a^{2s} b^n$

$a^{n-s+2s} b^n$

$a^{n+s} b^n$ ∴ There the no of a's will be more no of b's. In our language is no of a's equale to mo of b^s. So Here contradaction will be arraise.

Mimimization of DFA

construct a minimum DFA equivalent to the DFA description by



62

	0	1
→ q_0	q_1	q_5
q_1	q_6	q_2
q_2	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

0 Equivalence

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

1 Equivalence

$\{q_0, q_4, q_6\}$

$\{q_1, q_7\}$ $\{q_3, q_5\}$ $\{q_2\}$

2 Equivalence

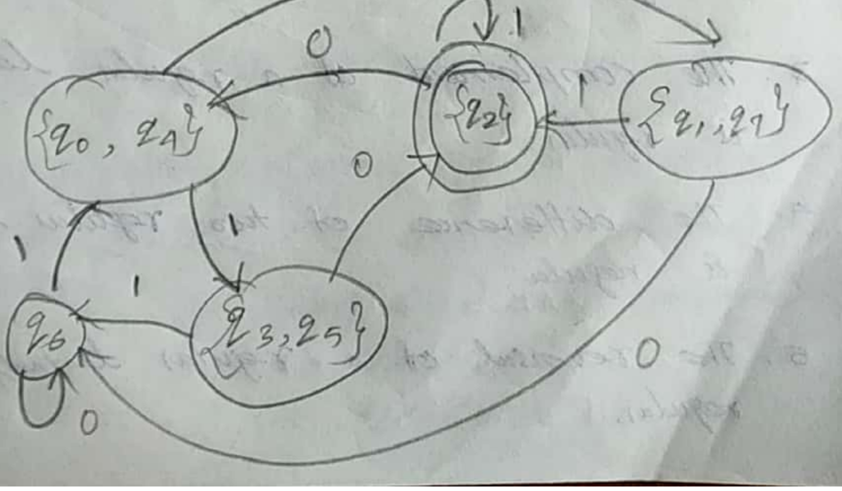
$\{q_0, q_4\}$ $\{q_6\}$ $\{q_1, q_7\}$ $\{q_3, q_5\}$; $\{q_2\}$

3 - Equivalence

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

	0	1
q_0	q_1	q_5
q_1	q_6	q_2
q_2	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

	0	1
$\{q_0, q_4\}$	$\{q_1, q_7\}$	$\{q_3, q_5\}$
$\{q_6\}$	$\{q_6\}$	$\{q_0, q_4\}$
$\{q_1, q_7\}$	$\{q_6\}$	$\{q_2\}$
$\{q_3, q_5\}$	$\{q_2\}$	$\{q_6\}$
$\{q_2\}$	$\{q_0, q_4\}$	$\{q_2\}$



Applications of Regular Expression

- The regular expressions can be modelled finite automata.

- As we have solved many examples and experienced.

(I) Text editors:

It is a program which are used for processing the text and substituting the string

s/bbb */b/

2. Lexical analyzers

compiler uses this program of lexical analyser in the process of compilation.

lexical analyzers is to scan the input program and separate out the 'tokens'.

Closure properties of Regular Language.

1. The union of two regular language is regular.

2. The intersection of two regular languages is regular.

3. The complement of a regular language is regular.

4. The difference of two regular language is regular.

5. The reversal of a regular language is regular.

6. The closure operation on a regular language is regular.

7. The concatenation of regular language is regular.

8. A hommorphism of regular language is regular.

9. The inverse homomorphism of regular language is regular.

UNIT-2

Grammars

Grammar.

- Grammars express languages.

ex:

<sentence> → <noun_phrase> <predicate>

<noun-phrase> → <article> <noun>

<predicate> → <verb>

<article> → a, the

<noun> → boy, dog

<verb> → runs, walks.

ex: The boy walks.

<sentence> → <noun_phrase> <predicate>

⇒ <noun-prase> <~~predicate~~ verb>

⇒ <article> <noun> <verb>

⇒ The <noun><verb>

⇒ The boy <verb>

⇒ The boy walks.

Types of Grammar

1. Type 3 Grammar

This type grammar generates the regular language.

Production rule

$A \rightarrow a$

&

$A \rightarrow aB$

- This grammar can be modeled using Finite automata.

2. Type 2 Grammar

→ This grammar generates the context free languages.

Production rule

$A \rightarrow \gamma$

Where A is non terminal γ is string of non terminal, and terminal

→ This grammar can be modeled using Push down automata.

3. Type 1 Grammar

- This grammar generates the context sensitive language.

Production rule.

$\alpha AB \rightarrow \alpha \gamma B$

A - non terminal, $\alpha \gamma B$ is a string of terminal & non-terminal.

This grammar can be modeled using linear bounded automata.

4. Type 0 Grammar,

- This type of grammar generates recursively enumarable lenguage.

Production rule

$\alpha \rightarrow B$

- This grammar can be modeled using Turing Machine.

Context Free Grammars and Language.

→ It has compare to the hyther level langguage as a regular language

→ The CFG can be formally defined as a set denoted by G = (V, T, P, S) where V and T are set of non-terminals and terminals respectively.

P is set of production rules.

S is a start symbol.

1. The capital letter are used to denotes the non-terminals

2. The lower case letters are terminals.

G = (V, T, P, S)

V = set of vaibiables (or) Non-Terminus symbol,

T = Terminal symbol

S = start symbol

P = production rule.

Scanned by CamScanner

Derivations and Languages

- Production rule are used to derive certain strings.

- The generation of language using specific rules is called derivation.

ex: 1

For generating a language that generates equal number of a's and b's in the form $a^n b^n$. the CFG will be defined as

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

$S \to aAb$

$\to aaAbb$ (by $A \to aAb$)

$\to aaaAbbb$. (by $A \to aAb$)

$\to aaabbb$ (by $A \to \epsilon$)

$\to a^3 b^3 \Rightarrow a^n b^n$

ex:2 construct CFG for the language L which has all the strings which are all paliodrome over $\Sigma = \{a, b\}$.

→ The string "madam" is a palindrome becas

read → Madum ← read

Same ↑

→ We want the production rules to be build a's and b's. As ε can be the palidrom.

Scanned by CamScanner

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

P can be $S \to aSa$

$S \to bSb$

$S \to a$

$S \to b$

$S \to \epsilon$

aSa

abSba

abaSaba

abaϵaba

abaaba

which is a Palindrome.

---

ex:3 Try to recognize the language L for given CFG. $G = [\{S\}, \{a,b\}, P, \{S\}]$

where $P = \left\{ \begin{array}{l} S \to aSb \\ S \to ab \end{array} \right\}$

Solutin

$S \to aSb \mid ab$ is a production rule

| indicates the 'or' operator.

S

aSb

↓

aaSbb

↓

aaaSbbb

↓

aaaabbbb

→ We can have any number of a's first then equal no of b's.

→ we guess the language as $\{ L = a^n b^n$ where $n \geq 1 \}$

→ The only way to recognize the language is to try out various strings from the given production rules.

Derivation Trees

- Derivation Tree is a graphical representati

- It is the Simple way to show how the derivation can be done.

- The derivation tree is also called parse tree.

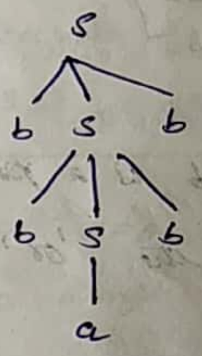
Properties

1. The root node is always a node indicating start Symbol.

2. The derivation is read from left to Right

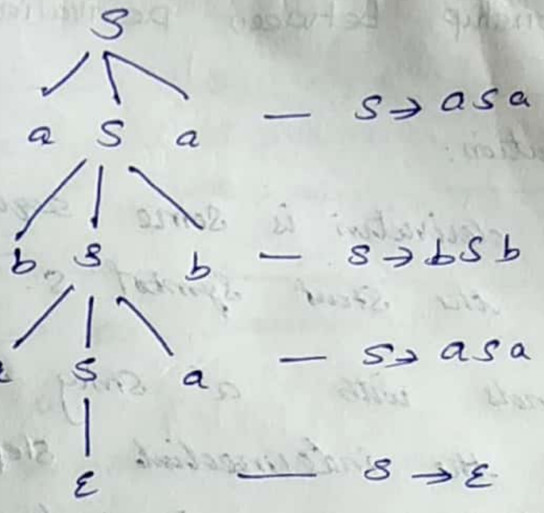
3. The leaf nodes are always terminal nodes.

4. The interior nodes are always the non-terminal nodes.



ex: Draw a derivation tree for the string abaaba for the CFG given by

G where $P = \{ S \rightarrow aSa$
$S \rightarrow bSb$
$S \rightarrow a|b|\epsilon \}$



⇒ abaaba

ex:2 construct the derivation tree for the String aabbabba from the CFG given by

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

S

↓

aB — $S \rightarrow aB$

a$\boxed{aBB}$ $S \rightarrow aBB$

aa$\boxed{bS}$B $B \rightarrow bS$

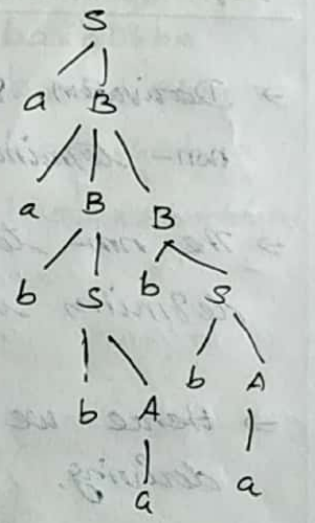
aab$\boxed{bA}$B $S \rightarrow bA$

aabb$\boxed{a}$B $A \rightarrow a$

aabba$\boxed{bS}$ $B \rightarrow bS$

aabbab$\boxed{bA}$ $S \rightarrow bA$

aabbabba $A \rightarrow a$



Relationship between Derivation and Derivation Trees.

Derivation:

- A derivation is some sequence that starts with the start symbol S.

→ ends with a string in the language and the intermediate steps may contain terminals and nonterminals.

→ Each step in the sequence expands one nonterminal using a production rule.

Derivation tree.

A parse tree is rooted at the start symbol, has terminal symbols as leaves and the children of a node correspond to a production rule.

Left Derivation and Rightmost Derivation

→ Derivation for any string means replacement of non-terminal by its appropriate definition.

→ The non-terminal should be replaced by its definition is sometimes confusing to decide.

→ Hence we normally apply two methods of deriving.

→ The leftmost derivation is a derivation in which the leftmost non-terminal is replaced first from the sentential form.

- The rightmost derivation is a derivation in which rightmost non-terminal is replaced first from the sentential form.

$S \rightarrow xyx$	$S \rightarrow xyx$
$S \rightarrow ayx$	$S \rightarrow xya$
$S \rightarrow abx$	$S \rightarrow xba$
$S \rightarrow aba$	$S \rightarrow aba$

ex: Let G be the grammar

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

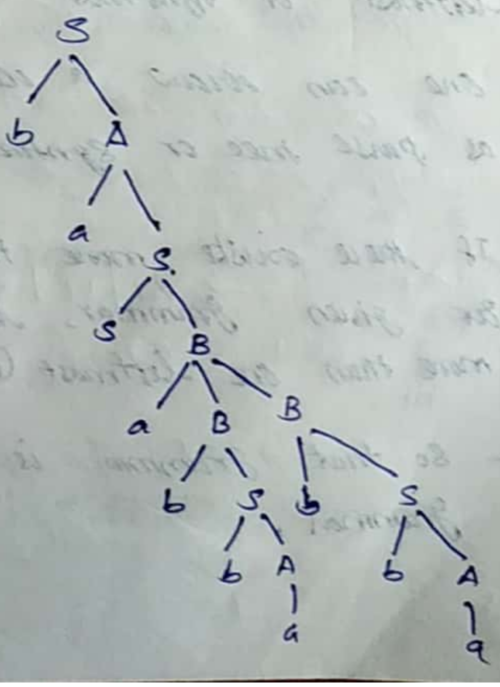
For the string baaabbabba. Find leftmost derivation, rightmost derivation and parse tree.

Leftmost

S
bA
baS
baaB
baaB
baaaBB
baaabSB
baaabbAB
baaabbaB
baaabbabS
baaubbabbA
baaabbabba

Rightmost

S
bA
baS
baaB
baaaBB
baaaBbS
baaaBbbA
baaaBbba
baaabSbba
baaabbAbba
baaabbabba



ex: 2 (TA)

Derive the string 1000111 for leftmost and rightmost derivation using CFG.

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

$V = \{S, T\}$

$T = \{0, 1\}$

$P = \{ S \rightarrow T00T$

$T \rightarrow 0T \mid 1T \mid \varepsilon \}$

left most

S	
TooT	S → TooT
1TooT	T → 1T
10TooT	T → 0T
10εooT	T → ε
1000T	
10001T	T → 1T
~~10001T~~	~~T → 1T~~
100011T	T → 1T
1000111T	T → 1T
1000111ε	T → ε
1000111	

Right

S	
TooT	
Too1T	T → 1T
Too11T	T → 1T
Too111T	T → 1T
Too111ε	T → ε
Too111	
1Too111	T → 1T
10Too111	T → 0T
10εoo111	T → ε
1000111	

Ambiguity

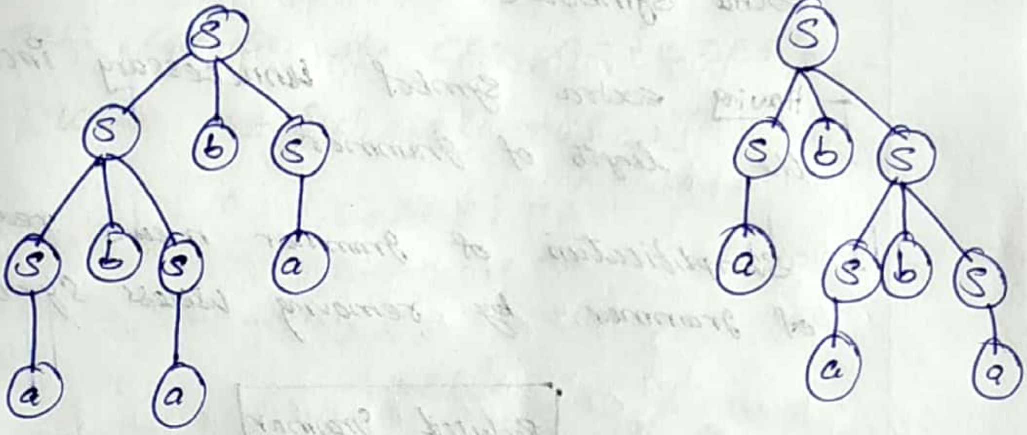
- The grammar can be derived in either leftmost or rightmost derivation.
- One can draw a derivation tree called as parse tree or syntax tree.
- If there exists more than one parse trees for given grammar, that means there could be more than one leftmost (or) rightmost possible.
- So that grammar is said to be ambiguous grammar.

① If G is the grammar $S \to SbS \mid a$. Show that G is ambiguous.

Let $w = ababa$ be the String such that $w \in G$ by production rule.

$S \to SbS$

$S \to a$



LM ⇒ ababa ⇐ RM.

② The CFG given by $G = (V, T, P, S)$ where

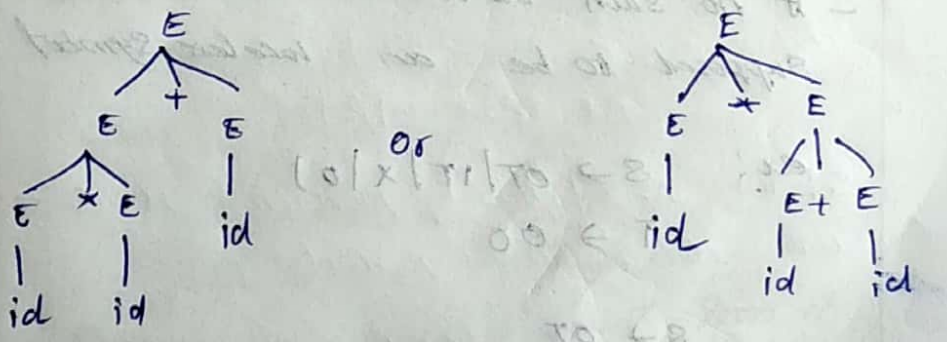
$V = \{E\}$

$T = \{id\}$

$P = \{E \to E+E,\ E \to E*E,\ E \to id\}$

$S = \{E\}$

Is the grammar ambiguous? G= id*id+id



⟶ id*id+id ⟵

Simplification of CFG

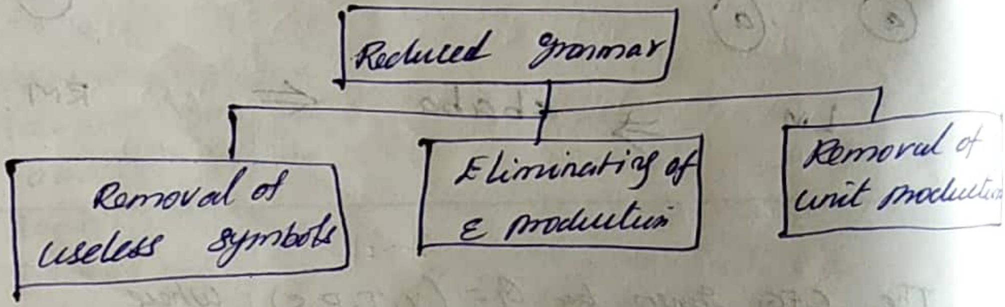
- A languages can effectively be represented by CFG.

- All the grammars are not always Optim

- That means grammar may consists of Son extra symbols (non-terminals).

- Having extra symbol unnecessary increase the length of grammar.

- Simplification of grammar means reduction of grammar by removing useless symbols.


Reduced grammar
Removal of useless symbols
Elimination of ε production
Removal of unit production

Removal of useless symbols

- Any symbol is useful when it appears on the right hand side. in the production rule generates some terminal string

- If no such derivation exist then it is supposed to be an useless symbol

ex: $S \to 0T \mid 1T \mid x \mid 0$

$T \to 00$

$S \to 0T$

000

$S \to 1T$

100

if $S \to x$ then there is no further rule as a definition to x

- Hence we can declare that x is a useless.

$P = \{ S \to 0T \mid 1T \mid 0 \mid 1$

$T \to 00\}$

---

① ex: Consider the CFG $G = \{V, T, P, S\}$

Where $V = \{S, A, B\}$ $T = \{0, 1\}$

$P = \{ S \to A11B \mid 11A$

$S \to 1B1 \mid 1$

$A \to 0$

$B \to BB\}$

For removing useless symbol.

$S \to$ ~~A11B~~ $\mid 11A$

$S \to$ ~~1B1~~

$A \to 0$

~~$B \to BB$~~

$S \to 11A$ ✓

110 $A \to 0$

→ Here B does not give any terminal String.

B 's no significant String.

- Hence we can declare B as useless symbol.

---

② Consider the following CFG,

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

Where $V = \{S, X, Y\}$

$T = \{0, 1\}$

$P = \{ S \to XY \mid 0,\ X \to 1)$

Remove the useless symbol from it.

$S \to XY$

$S \to 0$

$X \to 1$

- There is no derivation for Y. Hence we ca eliminate the production with Y.

$S \rightarrow 0$

$X \rightarrow 1$

Elimination of ε Productions from Grammar

- The finite automata and regular expression th ε or a null string indicates a string with no value.

- Also NFA contains ε moves we can conver that NFA with ε to NFA without ε.

- Even in CFG, if at all there is ε producti we can remove it, without changing the mean of the grammar.

$S \rightarrow 0S \mid 1S \mid \varepsilon$

$S \rightarrow 0S$ and $S \rightarrow \varepsilon \rightarrow S \rightarrow 0$

$S \rightarrow 1S$ and $S \rightarrow \varepsilon \rightarrow S \rightarrow 1$

Here we can rewrite the rule as

$S \rightarrow 0S \mid 1S \mid 0 \mid 1$

① Eliminate the ε productions from the CFG below

$A \rightarrow 0B1 \mid 1B1$

$B \rightarrow 0B \mid 1B \mid \varepsilon$

Solution: Now the ε production is $B \rightarrow \varepsilon$. will delete this production.

$A \rightarrow 0B1$

$A \rightarrow 01 \Rightarrow B \rightarrow \varepsilon$

$A \rightarrow 1B1$.

$11 \quad \Rightarrow B \rightarrow \varepsilon$

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

$B \rightarrow 0B$

$B \rightarrow 0 \quad \Leftrightarrow B = \varepsilon$

$B \rightarrow 1B$

$B \rightarrow 1 \quad \Leftrightarrow B = \varepsilon$

$B \rightarrow 0B \mid 1B \mid 0 \mid 1$

Collectively, we can write

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

$B \rightarrow 0B \mid 1B \mid 0 \mid 1$

② For the CFG given below remove the ε production

$S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow \varepsilon$

Sol :

$S \rightarrow aSa$

$\rightarrow aa \quad$ if $\Rightarrow S \rightarrow \varepsilon$

$S \rightarrow bSb$

$\rightarrow bb \quad$ if $\Rightarrow S = \varepsilon$

Thus Finally the rules are

$S \rightarrow aSa \mid bSb \mid aa \mid bb$

Scanned by CamScanner

Removing Unit Productions

- The unit productions are the productions which one non-terminal gives anoth non terminal.

ex:

$X \to Y$

$Y \to Z$

$Z \to X$

if $A \overset{*}{\Rightarrow} B$

$B \to x_1, x_2, x_3 \cdot \cdot x_n$

$A \to x_1, x_2, x_3 \cdot \cdot x_n$

① if the CFG is as below

$S \to 0A \mid 1B \mid \textcircled{C}$

$A \to 0S \mid 00$

$B \to 1 \mid \textcircled{A}$

$C \to 01$

then remove the unit production

solution:

Step:1 $S \to C$ is a unit production.

$S \to C$

$01 \quad \therefore C = 01$

$S \to 0A \mid 1B \mid 01.$

Step2:

$B \to A$ is also a unit production

$B \to A$

$\to 0S \mid 00 \quad A = 0S \mid 00$

Scanned by CamScanner

B → 1|0S|00

Finally we can write CFG without unit Production as

S → 0A|1B|01

A → 0S|00

B → 1|0S|00

C → 01

Normal Forms

- The grammar can be simplified by reducing the ε productions removing, removing useless symbols, and unit productions.

- There is also a need to have grammar in some specific form.

- In CFG at the right hand of the production there are any no of terminal (or) non termi. symbol in any combination.

- So need the grammar in some specific Format. ~~and~~

- There should be Fixed no of terminal and non terminals.

- Here two important Normal Forms.

  → Chomsky's Normal Form

  → Greibach Normal Form

Chomsky's Normal Form (CNF)

Non terminal → Non terminal . Non terminal

Non terminal → terminal

The given CFG should be converted in the above Format.

- Before converting the grammer into CNF it should be in reduced Form.

→ That means remove all the useless Symbols, ε production and unit production.

① context free grammar

$S \to ABA \mid BaA \mid A$ $\quad$ $A \to BA$

$A \to Ba \mid S \mid \varepsilon$ $\quad$ $A \to a$

$B \to Ba \mid b \mid Ca$

$C \to Ca$ $\quad$ convert to CNF

$D \to DaD \mid a$

Preliminary Steps

1) Elimination of useless symbols

2) Elimination of ε production

3) Elimination of unit production.

Step 1:

Elimination of useless symbols

which can't derives sum set of symbols

non generating $\quad$ not reachable

Scanned by CamScanner

S – A → Ba → ba ✓

A – Ba → ba ✓

B – b

C → Ca → ca ×

D – a ✓

non ~~ge~~ generati symbol.

If any symbol can a terminal generation symbol

S → ABA | BaA | A

A → Ba | S | ε

B → Ba | b

D → DaB | a

not reachable → that can't be reach starting state

S → ABA | BaA | A

S → A, B

A → B, S

B → B

Here D ⇒ is not given starting symbol so it called as not unreachable symbol. So it can eliminate from CFG

⇓

S → ABA | BaA | A

A → Ba | S | ε

B → Ba | b

Step : 2 eliminating ε production

A → ε to eliminate ε

S → ABA → | BA | AB | B

$\downarrow$

ε

S → BaA | Ba

S → A → ε

~~S → ABA | BaA~~ | A

S → ABA | BA | AB | B | BaA | Ba | A | ε

B → Ba/B A → Ba/S

(84) S → ε

S → ABA | BA | AB | B | BaA | Ba | A | ~~ε~~ elemo

A → Ba | S | ε

B → Ba | b

Here again A → ε but we have already remove A → ε so directly remove ε.

⇓

rewrite again

S → ABA | BA | AB | B | BaA | Ba | A

A → Ba | S |

B → Ba | b

Step 3 : unit production .

ex: S → B A →
S → A ,

Variable then's to Single Variable

S → ABA | BA | AB | B | BaA | Ba | A

A → Ba | S

B → Ba | b

⇓

S → ABA | BA | AB | Ba|b | BaA | Ba | Ba | ~~S~~

A → Ba | ABA | BA | AB | ~~Ba~~ | b | BaA

B → Ba | b

Here S → S
so there is no
meaning so dir
eliminat

Now Chomsky normal form

$S \rightarrow \overset{\times}{ABA} \mid \overset{\checkmark}{BA} \mid \overset{\checkmark}{AB} \mid \overset{\times}{Ba} \mid \overset{\checkmark}{b} \mid \overset{\times}{BaA}$

$A \rightarrow \overset{\times}{Ba} \mid \overset{\times}{ABA} \mid \overset{\checkmark}{BA} \mid \overset{\checkmark}{AB} \mid \overset{\checkmark}{b} \mid \overset{\times}{BaA}$

$B \rightarrow \overset{\times}{Ba} \mid \overset{\checkmark}{b}$

$\boxed{\begin{array}{l} \text{CNF} \\ A \rightarrow BA \\ A \rightarrow a \end{array}}$

$S \rightarrow \underbrace{AB}_{X_1} A$

$S \rightarrow X_1 A$

$X_1 \rightarrow AB$

$S \rightarrow Ba$

$X_2 \rightarrow a$

$S \rightarrow B X_2$

$S \rightarrow BaA$

$S \rightarrow \underbrace{BX_2}_{X_3} A$

$X_3 \rightarrow BX_2$

$S \rightarrow X_3 A$

$S \rightarrow X_1 A \mid BA \mid AB \mid BX_2 \mid b \mid X_3 A$

$X_1 \rightarrow AB$

$X_2 \rightarrow a$

$X_3 \rightarrow BX_2$

$A \rightarrow BX_2 \mid X_1 A \mid BA \mid AB \mid b \mid X_3 A$

$B \rightarrow BX_2 \mid b$

② Convert the grammar

$S \rightarrow AB \mid aB$

$A \rightarrow aab \mid \varepsilon$ into CNF

$B \rightarrow bbA$

Here Step 1: Eliminate useless symbol not applicable

Step 2: Eliminating ε production

$\boxed{A \rightarrow \varepsilon}$

$S \rightarrow AB \rightarrow B$

$S \rightarrow AB \mid B$

$S \rightarrow AB \mid B \mid aB$

$A \rightarrow aab$

$B \rightarrow bbA$

86

Step 3: Unit production

$S \rightarrow AB \mid \boxed{B} \mid aB$

$A \rightarrow aab$

$B \rightarrow bbA$

⇓

$S \rightarrow AB \mid bbA \mid aB$

$A \rightarrow aab$

$B \rightarrow bbA$

Now CNF

$S \rightarrow AB \mid bb\overset{x}{A} \mid a\overset{x}{B}$

$A \rightarrow a\overset{x}{a}b$

$B \rightarrow b\overset{x}{b}A$

$S \rightarrow AB \mid X_1 A \mid X_2 B$

$X_1 \rightarrow bb$

$X_2 \rightarrow a$

$A \rightarrow X_2 X_3$

$X_3 \rightarrow X_2 b$

$B \rightarrow X_1 A$

$S \rightarrow \underline{bb}A$
X_1

$S \rightarrow X_1 A$

$X_1 \rightarrow bb$

$S \rightarrow aB$

$X_2 \rightarrow a$

$S \rightarrow X_2 B$

$A \rightarrow aab$

~~A~~ -

$X_2 \rightarrow a$

$A \rightarrow X_2 X_2 b$

$X_3 \rightarrow X_2 b$

$A \rightarrow X_2 X_3$

$S \rightarrow AB \mid X_1 X_2 \mid X_3 B$

$X_1 \rightarrow b$

$X_2 \rightarrow X_1 A$

$X_3 \rightarrow a$

$A \rightarrow X_4 X_5$

$X_4 \rightarrow X_3 X_3$

$X_5 \rightarrow b$

$B \rightarrow X_1 X_2$

$S \rightarrow bbA$

$X_1 \rightarrow b$

$S \rightarrow X_1 \underline{X_1} A$

$X_2 \rightarrow X_1 A$

$S \rightarrow X_1 X_2$

$S \rightarrow aB$

$X_3 \rightarrow a$

$S \rightarrow X_3 B$

$A \rightarrow aab$

X_4

$A \rightarrow X_3 X_3$

$X_4 \rightarrow X_3 X_3$

$X_5 \rightarrow b$

$A \rightarrow X_4 X$

$S \rightarrow AB \mid X_1X_2 \mid X_3B$

$X_1 \rightarrow b$

$X_2 \rightarrow X_1A$

$X_3 \rightarrow a$

$A \rightarrow X_4X_1$

$X_4 \rightarrow X_3X_3$

$B \rightarrow X_1X_2$

$S \rightarrow bbA$

$X_1 \rightarrow b$

$S \rightarrow X_1X_1A$

$X_2 \rightarrow X_1A$

$S \rightarrow X_1X_2$

$S \rightarrow aB$

$X_3 \rightarrow a$

$S \rightarrow X_3B$

$A \rightarrow aab$

$X_4 \rightarrow a$

$A \rightarrow X_4X_4X_1$

$X_5 \rightarrow X_4X_4$

$A \rightarrow X_5X_1$

$A \rightarrow aab$

$X_3X_3X_1$

$X_4 \rightarrow X_3X_3$

$A \rightarrow X_4X_1$

Greibach Normal Form

A CFG is in Greibaach Normal Form if the productions are in the following Forms:

$NT \rightarrow t.NT$	$A \rightarrow b$ $A \rightarrow bC_1C_2 \ldots C_n$

where $A, C_1, \ldots, C_n$ are Non-Terminals and b is a Terminals.

Steps to covert a given CFG to GNF

Step 1: Check if the given CFG has any unit productions or Null productions and Remove if there are any.

Step 2: Check whether the CFG is already in chomsky Normal Form (CNF) and convert it to CNF if it is not.

Step 3: Change the names of the Non-Terminal symbols into Some A_i in ascending order of i.

Example :

$S \to CA \mid BB$

$B \to b \mid SB$

$C \to b$

$A \to a$

Replace

$S \to A_1$

C with A_2

A with A_3

B with A_4

We get :

$A_1 \to A_2 A_3 \mid A_4 A_4$

$A_4 \to b \mid A_1 A_4$

$A_2 \to b$

$A_3 \to a$

Step 4:

Alter the rules so that the Non-Termi are in ascending order. Such that if the production is of the form $A_i - A_j$ x then $i<j$ and should never be $i \geq j$.

Check whether the condition $i<j$ $A_1 < A_2$ ✓ $A_1 < A_3$ ✓ $A_1 < A_4$ ✓

$A_1 \to A_2 A_3 \mid A_4 A_4$

$A_4 \to b \mid \underline{A_1} A_4$

$A_4 < A_1$ x

$A_4 \to b \mid \underline{A_2} A_3 A_4 \mid A_4 A_4 A_4$

$A_4 < A_2$

$A_4 \to b \mid b A_3 A_4 \mid A_4 A_4 A_4$

$A_4 \to b$ ✓

$A_4 \to b A_3 A_3$ ✓

$A_4 \to A_4 A_4 A_4$ x

$A_2 \to b$

$A_3 \to a$

Left Recursion

Scanned by CamScanner

Step 5 : Remove Left Recursion

→ Introduce a New Variable to remove the Left Recursion.

$A_4 \rightarrow b \mid b A_3 A_4 \mid \underline{A_4 A_4 A_4}$

$Z \rightarrow A_4 A_4 Z \mid A_4 A_4$

$$\boxed{A_4 \rightarrow b \mid b A_2 A_4 \mid bZ \mid bA_3 A_4 Z}$$

Now the grammar is

~~A~~ Aj

$A \rightarrow t . Nt$

$\rightarrow A_1 \rightarrow \underline{A_2}\overset{\times}{} \underline{A_3}\overset{\times}{} \mid A_4 A_4$ — Here not

Now check grammar at top to bottom

$A_4 \rightarrow b \mid b A_3 A_4 \mid bZ \mid b A_3 A_4 Z$

$Z \rightarrow A_4 A_4 \mid A_4 A_4 Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

$A_1 \rightarrow b A_3 \mid bA_4 \mid b A_3 A_4 A_4 \mid bZA_4 \mid bA_3 A_4 Z A_4$

$A_4 \rightarrow b \mid b A_3 A_4 \mid bZ \mid b A_3 A_4 Z$

$Z \rightarrow bA_4 \mid bA_3 A_4 A_4 \mid bZA_4 \mid bA_3 A_4 Z A_4 \mid$
$bA_4 Z \mid bA_3 A_4 A_4 Z \mid bZA_4 Z \mid bA_3 A_4 Z A_4 Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

② Convert the given CFG to GNF

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

Here A & C is GNF but S ~~is~~ not GNF.

$S \to CA$

$S \to aBA \mid bA$

we can write equivalent GNF as

$S \to aBA \mid bA$

$A \to a$

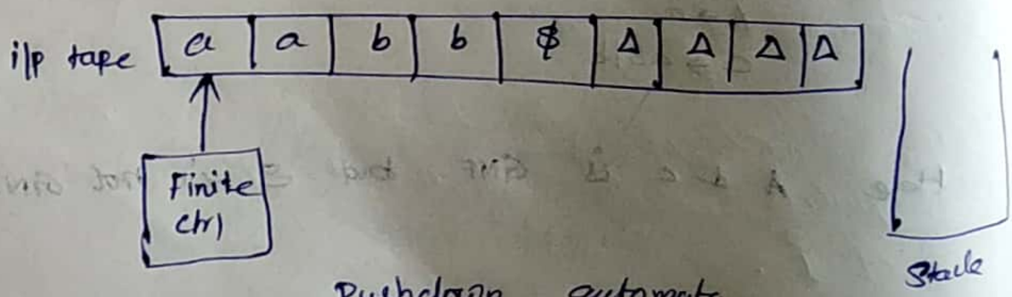
$C \to aB \mid b$

UNIT-3

Pushdown Automata

Pushdown Automata

- The push down automata will have input tape, finite control and stack.
- The i/p tape is divided in many cells. At each cell only one input symbol is placed. thus certain i/p string is placed on tape.
- The finite control has some pointer which point the current symbol which is to be read.
- At the end of input $ or Δ (blank) symbol is placed which indicate end of i/p.



Pushdown automata.

- The Pushdown automata are using the Stack for Storing the items temporarily inserting the symbol onto the Stack is called push operation.

- removing a symbol from stack is called Pop operation.

PDA components

1. The Finite set of states Q.

2. The input set Σ.

3. Γ is a stack alphabet

4. q_0 is initial stage, $q_0 \in Q$.

5. ~~q_0 is initial stage. $q_0 \in Q$.~~

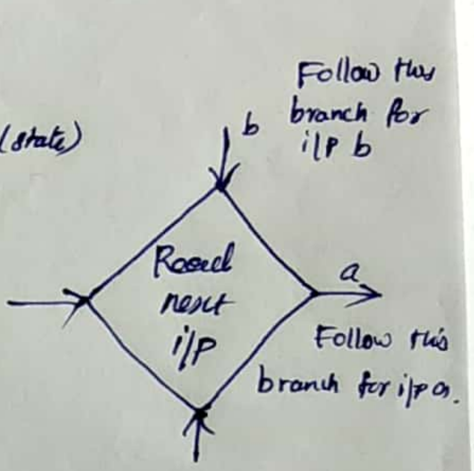
5. z_0 is a start symbol which is in Γ.

6. Set of Final states $F \subseteq Q$.

7. δ is mapping Function used for moving from current state to next state.

Start	Start symbol
Accept	Accept state (Which is always is Final state)
Reject	Reject state
↓	Connecting edge

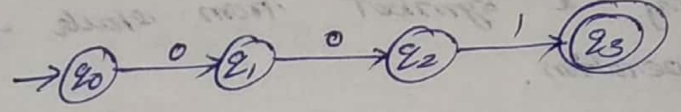
Symbol used in PDA



Read symbol

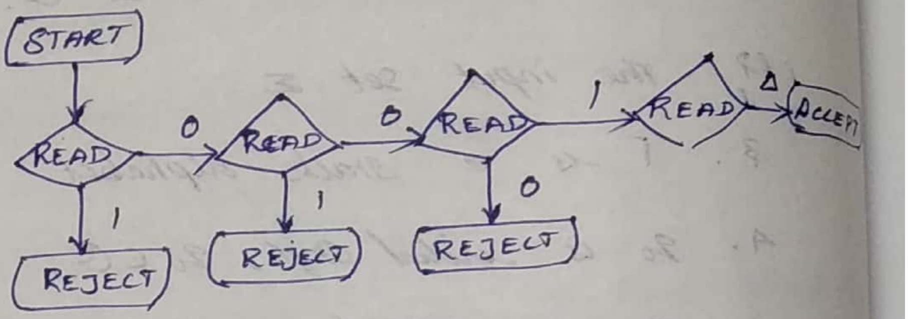
1) Design PDA for the language $L = \{001\}$

The simple FA for this language is



Where q_0 is a start state & q_3 is a accept state.

We can draw an equivalent PDA as

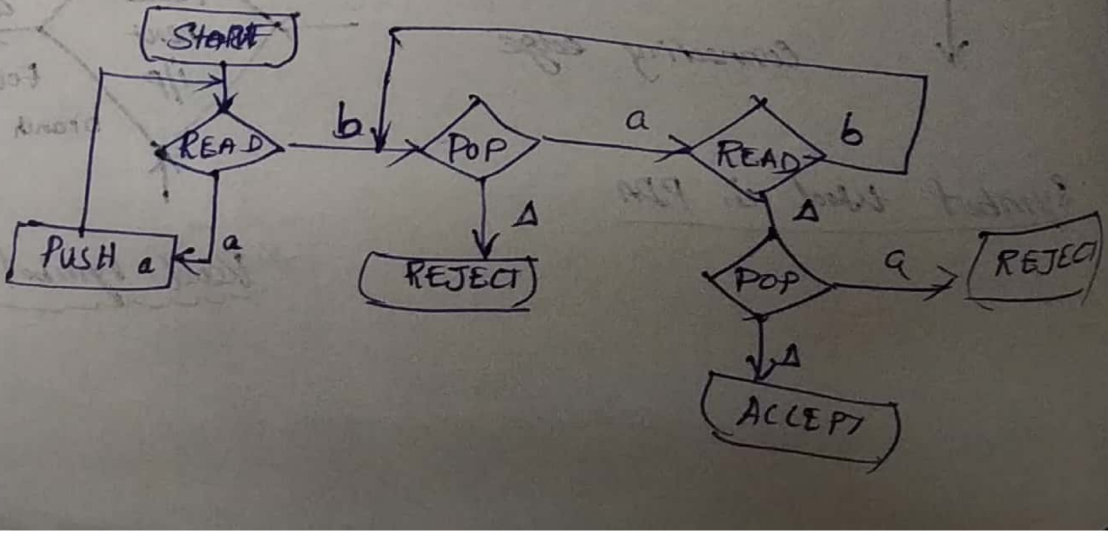


2) Design a PDA for the language $L = \{a^n b^n\}$ where $n \geq 1$

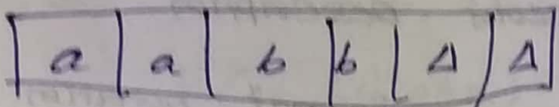
Here we can't draw Finite automata for this problem because here the condition is that when many no of a's are occuring there many no of b's.

if $n=5$ then string will be aaaaabbbbb

so here no memory in FA.



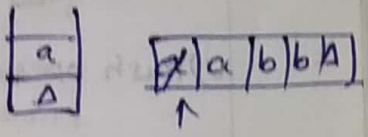
Input tape for $a^n b^n$ now $n=2$



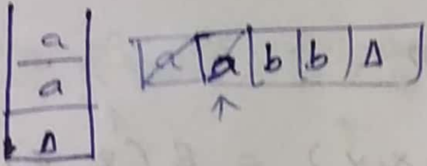
Step 1 : We will start with Start symbol

Step 2 : We will reach to a read State, we will read First a and push it onto the Stack

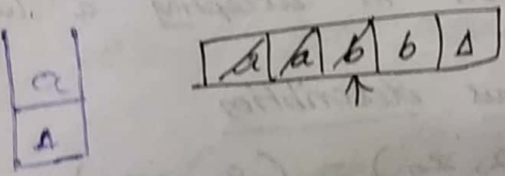
The stack initially contains Δ that means it is empty.



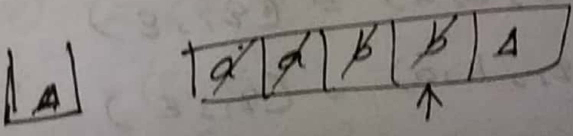
Step 3 : Read next symbol from i/p tape.



Step 3 : Read b from the tape and pop from the Stack a.



After reading second b we pop one more a



both tape and Stack are empty.

Instantaneous Description .

- It is an informal notation

- How PDA computes a input String and mak a decision that String is accepted or not .

- there is an effective use of Stack as a memory .

(i) Push operation

current Stack top

Push x onto the Stack

$$\delta(q_0, x, z_0) = \delta(q_1, x, z_0)$$

read i/p on the tape

Change the State from q_0 to q_1

(ii) Pop operation

current Stack top

change the State from q_0 to q_1

$$\delta(q_0, x, y) = \delta(q_1, \varepsilon)$$

read i/p on tape

→ Pop the Stack

① Design a PDA For accepting a language $\{L = a^n b^n | n \geq$

<u>Instantaneous</u> <u>describtion</u>

$$\delta(q_0, a, z_0) = (q_0, az_0)$$

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

$$\delta(q_0, b, a) = (q_1, \varepsilon)$$

$$\delta(q_1, b, a) = (q_1, \varepsilon)$$

$$\delta(q_1, \varepsilon, z_0) = (q_2, \varepsilon)$$

Where q_0 is a start State and q_2 is accept State .

Scanned by CamScanner

$(q_0, aabb, z_0) \vdash (q_0, abb, az_0)$

$\vdash (q_0, bb, aaz_0)$

$\vdash (q_1, b, az_0)$

$\vdash (q_1, \varepsilon, z_0)$

$\vdash (q_2, \varepsilon)$

Accept State.

2. construct PDA for the language $L = \{a^n b^{2n} \mid n \geq 1)$

- Here number of a's followed by 2n number of b's.
- If we read single 'a' we will push two a's onto the stack.
- If we read 'b' then for every single 'b' only one 'a' should get popped from the stack.

$\delta(q_0, a, z_0) = (q_0, aaz_0)$

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

$\delta(q_0, b, a) = (q_1, \epsilon)$

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

$\delta(q_1, \epsilon, z_0) = (q_2, \epsilon)$

Let us simulate this PDA for some input string "aaabbbbbb"

$\delta(q_0, aaabbbbbb, z_0) \vdash (q_0, aabbbbbb, aaz_0)$

$\vdash (q_0, abbbbbb, aaaaz_0)$

$\vdash (q_0, bbbbbb, aaaaaaz_0)$

$\vdash (q_1, bbbbb, aaaaaz_0)$

$\vdash (q_1, bbbb, aaaaz_0)$

$\vdash (q_1, bbb, aaaz_0)$

$\vdash (q_1, bb, aaz_0)$

$\vdash (q_1, b\cancel{\#}, az_0)$

$\vdash (q_1, \epsilon, z_0)$

$\vdash (q_2, \epsilon)$

Final state (or) Accept State.

(3) Construct the PDA for $L = \{ww^R \mid w \text{ is in } (a+b)^+\}$

It is like a Palindrom why mean ww^R

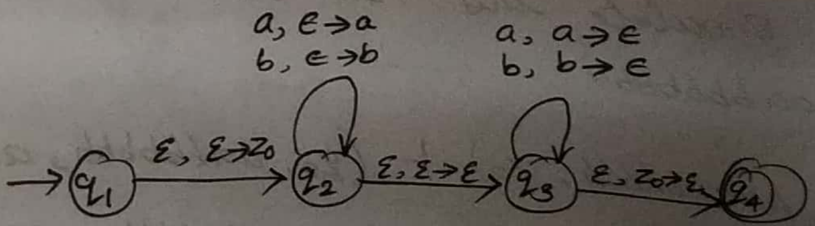
R is a Reverse process

→ ww ← R

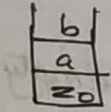
Palindrom		
	NOON - (4)	Even Palindrom
	MADAM - (5) x	
	123321 - (6) ✓	
	abba - (4) ✓	

+ closure - It may not contain empty symbol have a one symbol

* closure - It may contain empty symbol.

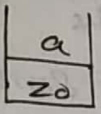


$\underline{a}\ \underline{b}|ba$



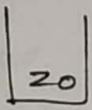
Push b

$ab|\underline{b}a$



Pop

$ab|b\underline{a}$



$z_0 \to \varepsilon$

→ Stack is empty

so It Accepted.

$\delta(q_1, abba, z_0) \vdash \delta(q_2\,\underline{a}bba, az_0)$

$\vdash \delta(q_2\ \underline{b}ba, baz_0)$

$\vdash \delta(q_3\ ba, baz_0)$

$\vdash \delta(q_3\ \underline{b}a, az_0)$

$\vdash \delta(q_3\ \underline{a}, z_0)$

$\vdash \delta(q_4, \varepsilon, z_0)$

ACCEPT.

Languages of PDA

1. Acceptance by Final State
2. Acceptance by empty Stack.

1. Acceptance by Final State.

The PDA accepts its input by consuming it and then it enters in the Final State.

2. Acceptance by empty stack

On reading the input string from initial configuration for some PDA, the stack of PDA gets empty.

④ Design PDA to accept $\{0^n 1^n \mid n>1\}$. Draw the transition diagram for the PDA. Show by instantaneous description that the PDA accepts the string 0011.

- Push all 0's onto the stack
- Then on reading every single 1, each 0 is popped
- Thus we read all 1's and pop all 0's one by one.
- When the stack is empty then the string is said to be accepted.

$$\delta(q_0, 0, z_0) = (q_0, 0z_0)$$
$$\delta(q_0, 0, 0) = (q_0, 00)$$
$$\delta(q_0, 1, 0) = (q_1, \epsilon)$$
$$\delta(q_1, 1, 0) = (q_1, \epsilon)$$
$$\delta(q_1, \epsilon, z_0) = (q_2, \epsilon)$$

Simulation of the string 0011

$(q_0, \underline{0}011, z_0) \vdash (q_0, \underline{0}11, 0z_0)$

$\vdash (q_0, \underline{1}1, 00z_0)$

$\vdash (q_1, 1, 0z_0)$

$\vdash (q_1, \epsilon, z_0)$

$\vdash (q_2, \epsilon)$ ACCEPT

Non Deterministic Pushdown Automata (NDPDA)

(i) NPDA is very much similar to NFA

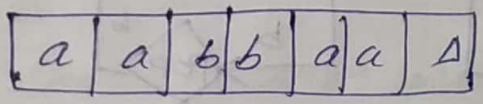
(ii) The CFG which accepts deterministic PDA accepted non determinisitic PDA's as well.

(iii) There are some CFG's which can be accepte only by NDPA and not by DPDA.

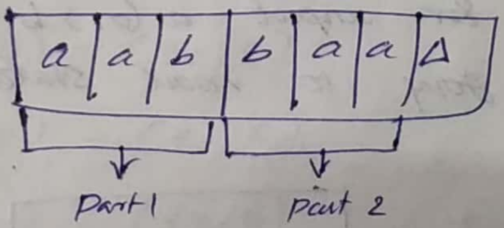
(iv) Thus NDPDA is more powerfull then DPDA.

① construct a NPDA for the language L containing all the strings which are palindrim over Σ = {a, b}.

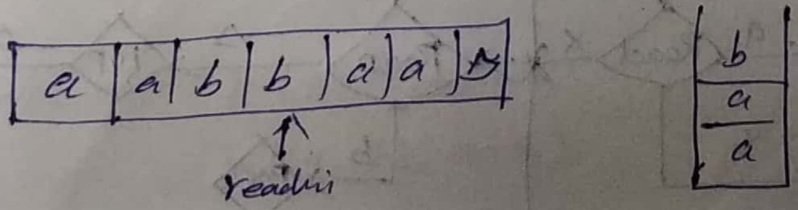
if the string is

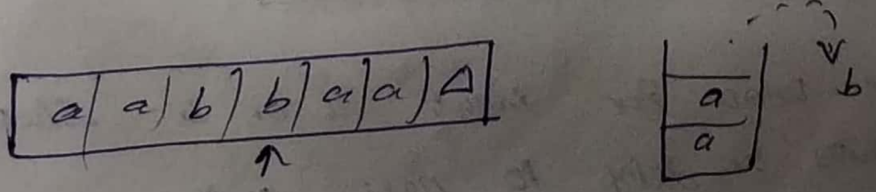


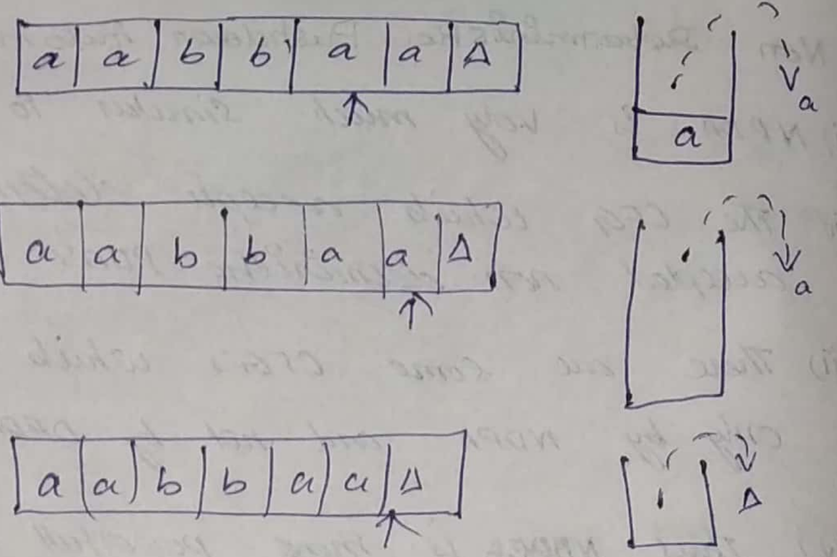
Now the list can be divided in two halves



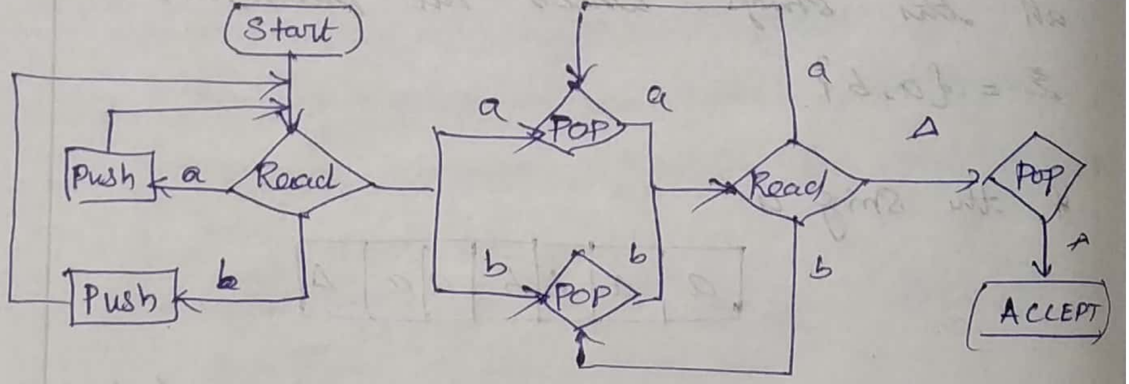
We will push all the elements of part 1 onto the stack



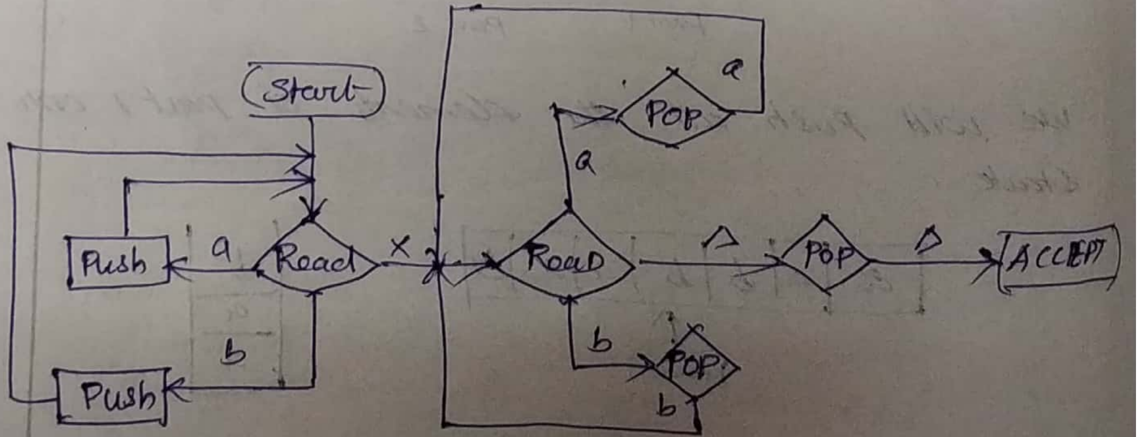




NPDA



The NPAD for input a (or) b there are multiple Paths for going to next state.



The DPAD for input a or b thea are multi paths for going to next state.

Deterministic Pushdown Automata

DPDA is transit to next state one input and one path only at a time.

DPDA can be defined as a collection of

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

Q is a finite set of states

Σ is a finite set of input

Γ is a finite set of state symbol

δ is a mapping function

q_0 is initial state

Z_0 is a initial symbol in stack

F is a finite set of final states.

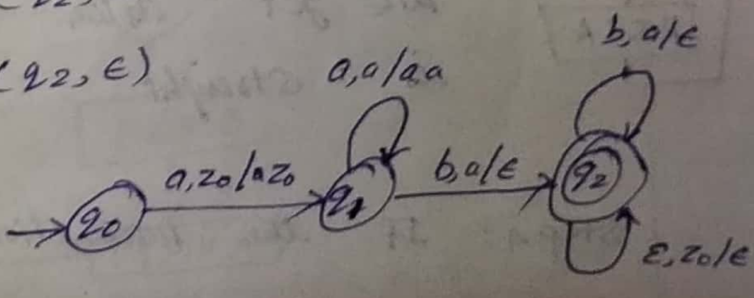
① Design DPDA for $L = a^n b^n$ where $n \geq 1$

$\delta(q_0, a, Z_0) = (q_1, aZ_0)$

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

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

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



Simulation

$\delta(q_0, aabb, Z_0) \vdash (q_1, abb, aZ_0)$

$\vdash (q_1, bb, aaZ_0)$

$\vdash (q_2, b, aZ_0)$

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

$\vdash (q_2, \epsilon)$

Scanned by CamScanner

Equivalence of Pushdown Automata and CFL

→ Construction of PDA from CFG

→ Construction of CFG from PDA

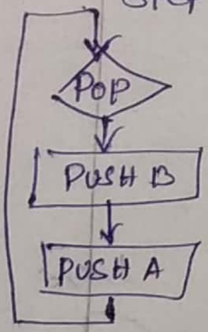
Construction of PDA from CFG

Step 1: convert the given CFG to Chomsky normal form.

Step 2: The PDA should start by pushing start symbol onto the stack. To derive further production rules for the start sym we immediately perform the pop operati

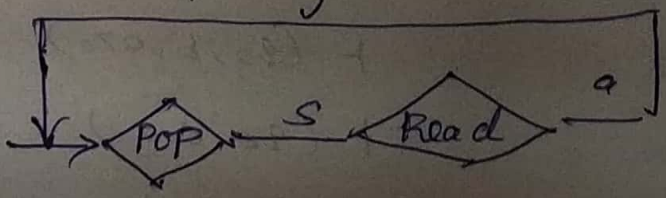
Step 3: If the production is of the form S → AB; we push A and B onto the stack in reverse order.

We get after popping the reverse ord is straight



Step 4: If the production is of the form S → a we design as

This means replacing a nonterminal S by a.



Step 5 : Finally when the complete i/p is read from the tape, we encounter with Δ.

Popping Δ we get ensured with the fact that stack is also empty.

1) Construct PDA for following grammar.

S → AB

A → CD

B → b

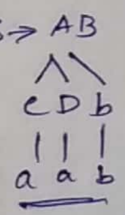
C → a

D → a

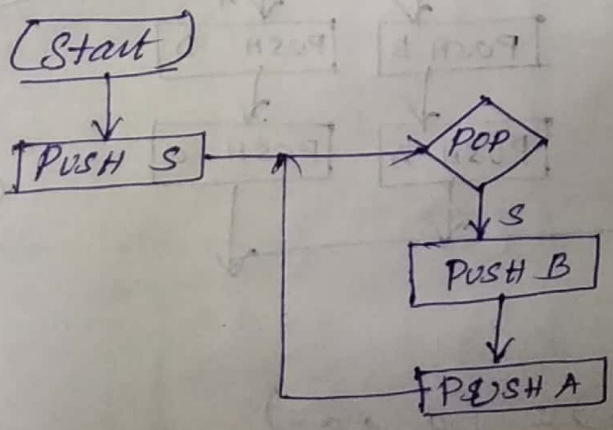
NT. NT or T

ex: CNF → AB
a
b

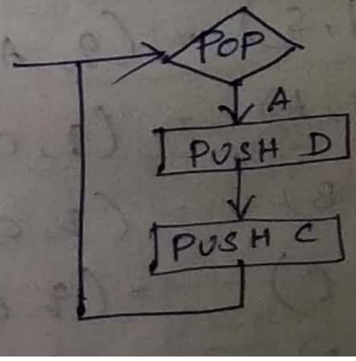
Step 1 : The grammar alread chomsky's normal form



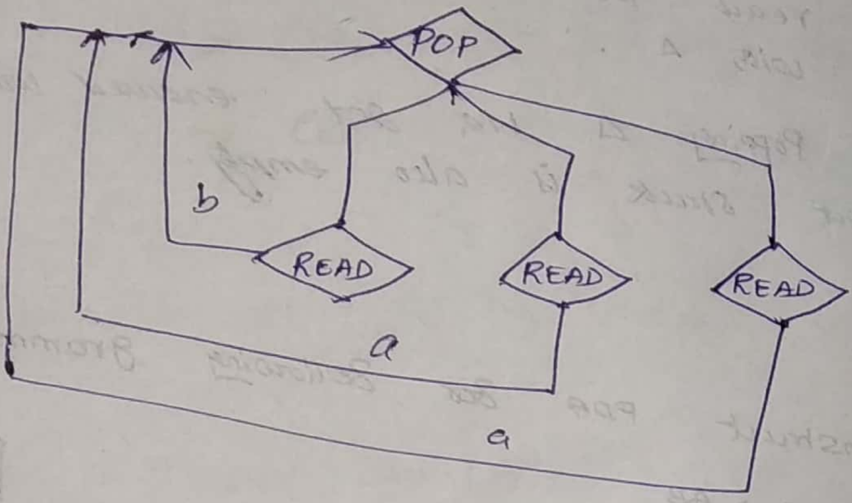
Step 2 : (i) S → AB



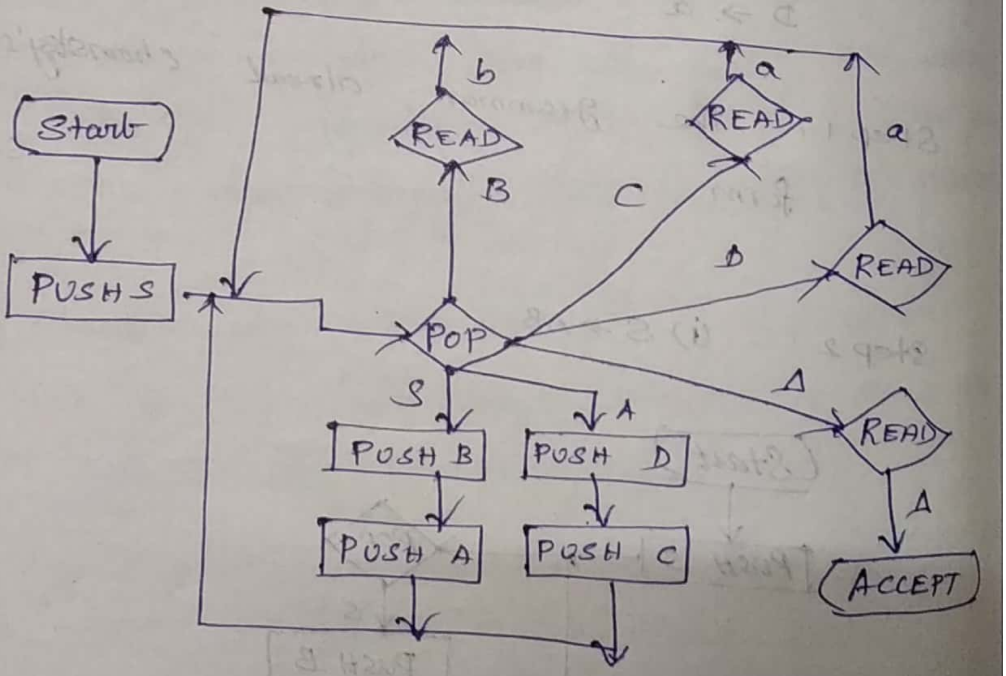
(ii) A → CD



(iii) $B \rightarrow b$, $C \rightarrow a$, $D \rightarrow a$



Finally We could draw the complete PDA as



$\delta(q_0, \epsilon, z_0) = (q, Sz_0)$

Where W is some

$\delta(q, W, S) = (q, AB)$

$\delta(q, W, A) = (q, CD)$

$\delta(q, b, B) = (q, \epsilon)$

$\delta(q, a, C) = (q. \epsilon)$

$\delta(q, a, D) = (q, \epsilon)$

We Will now simulate the string "aab"

$\delta(q, \epsilon, z_0) = (q, Sz_0)$

$\delta(q, aab, S) \vdash (q, aab, AB)$

$\vdash (q, \underline{a}ab, \underline{C}DB)$

$\vdash (q, \underline{a}b, \underline{D}B)$

$\vdash (q, \underline{b}, \underline{B})$

$\vdash (q, \epsilon)$

ACCEPTED

(3) construct PDA For the given CFG

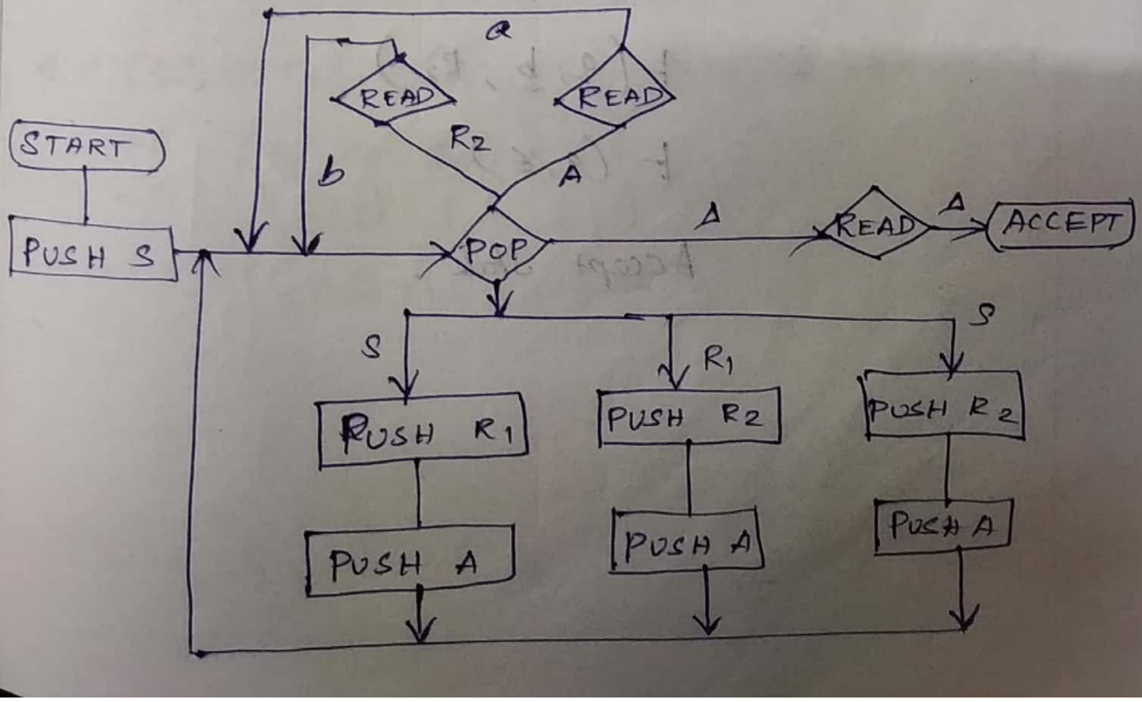
$S \rightarrow aSb$

$S \rightarrow ab$

Step 1: Convert this CFG to CNF.

$S \rightarrow aSb$ $S \rightarrow ab$	$\Rightarrow$	$S \rightarrow AR_1$ $S \rightarrow AR_2$ $R_1 \rightarrow SR_2$ $R_2 \rightarrow b$ $A \rightarrow a$

The PDA can be graphically represented as



The graphical PDA is

$\delta(q, \epsilon, Z_0) = (q, SZ_0)$

Now place string w on input tape

$\delta(q, w, S) = (q, AR_1)$

$\delta(q, w, S) = (q, AR_2)$

$\delta(q, w, R_1) = (q, SR_2)$

$\delta(q, b, R_2) = \delta(q, \epsilon)$

$\delta(q, a, A) = \delta(q, \epsilon)$

Consider now input "aabb" for simulatin

$(q, \epsilon, Z_0) \vdash (q, SZ_0)$

$(q, aabb, S) \vdash (q, aabb, AR_1)$

$\vdash (q, abb, R_1)$

$\vdash (q, abb, SR_2)$

$\vdash (q, \underline{a}bb, AR_2R_2)$

$\vdash (q, \underline{b}b, R_2R_2)$

$\vdash (q, \underline{b}, R_2)$

$\vdash (q, \epsilon)$

Accept State.

Construct of CFG to PDA

$P = (Q, \Sigma, F, \delta, q_0, Z_0, q_n)$ is a PDA there exists CFG G which is accepted by PDA P.

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

Algorithm

1. If q_0 is start state in PDA and q_n is Final state of PDA then $[q_0 Z q_n]$ becomes start state of CFG.

2. The production rule for the ID of the form $\delta(q_i, a, Z_0) = (q_{i+1}, Z_1, Z_2)$ can be obtained as

$\delta(q_i Z_0 q_{i+k}) \rightarrow a(q_{i+1} Z_1 q_m)(q_m Z_2 q_{i+k})$

→ where q_{i+k}, q_m represents the intermediate States

→ Z_0, Z_1, Z_2 are stack symbols and a is input symbol.

3. The production rule for the ID of the form

4. $(q_i, a, Z_0) = (q_{i+1}, \epsilon)$ can be converted as $(q_i Z_0, q_{i+1}) \rightarrow a$

① Obtain CFG for the PDA given as below

$A = (\{q_0, q_1\}, \{0, 1\}, \{A, Z\}, \delta, Z, \{q_1\})$ where δ is as given below

$\delta(q_0, 0, Z) = (q_0, AZ)$

$\delta(q_0, 1, A) = (q_0, AA)$

$\delta(q_0, 0, A) = (q_1, \epsilon)$

Solution :

1. $S \rightarrow [q_0, Z, q_0]$
2. $S \rightarrow [q_0, Z, q_1]$

$\underline{\delta(q_0, 0, Z) = (q_0, AZ)}$ can be converted using rule 2.

⇑

$\delta(q_i, a, Z_0) = (q_{i+1}, Z_1 Z_2)$ givers

$\delta(q_i\ Z_0\ q_{i+k}) \rightarrow a(q_{i+1}\ Z_1\ q_m)(q_m\ Z_2\ q_{i+k}$

Here $(q_i = q_0,\ a = 0,\ Z_0 = Z) \Rightarrow (q_{i+1} = q_0,\ Z_1 = A,\ Z_2 =$

$\delta(q_0, 0, Z) = (q_0, AZ)$ is

3. $(q_0\ Z\ q_0) \rightarrow 0(q_0\ A\ q_0)(q_0\ Z\ q_0)\ |\ 0(q_0\ A\ q_1)(q_1$

4. $(q_0\ Z\ q_1) \rightarrow 0(q_0 A q_0)(q_0 Z q_1)\ |\ 0(q_0 A q_1)(q_1 Z q_1$

$\delta(q_0, 1, A) = (q_0, AA)$

Here

$(q_i = q_0, a = 1, Z_0 = A) \Rightarrow (q_{i+1} = q_0,\ Z_1 = A,\ Z_2 = A)$

5. $(q_0\ A\ q_0) \Rightarrow 1(q_0\ A\ q_0)(q_0\ A\ q_0) \mid 1(q_0 A q_1)(q_1 A q_0)$

6. $(q_0\ A\ q_1) \rightarrow 1(q_0\ A\ q_0)(q_0 A q_1) \mid 1(q_0\ A\ q_1)(q_1\ A\ q_1)$

$\delta(q_0, 0, A) = (q_1\ \epsilon)$ Here Apply rule 3

if $\delta(q_i\ a, z_0) = (q_{i+1}, \epsilon)$ the $(q_i\ z_0\ q_{i+1}) \rightarrow a$

$q_i = q_0,\ a = 0,\ z_0 = A,\ q_{i+1} = q_1,\ \epsilon = \epsilon$

7. $(q_0\ A\ q_1) \rightarrow 0$

To summerize equivalent CFG as

$S \rightarrow [q_0, \cancel{z_0}, q_0]$ $S \rightarrow [q_0, z, q_1]$

$(q_0 Z q_0) \rightarrow 0(q_0\ A q_0)(q_0 Z q_0) \mid 0(q_0 A q_1)(q_1 Z q_0)$

$(q_0 Z q_1) \rightarrow 0(q_0 A q_0)(q_0 Z q_1) \mid 0(q_0 A q_1)(q_1\ Z q_1)$

$(q_0\ A\ q_0) \rightarrow 1(q_0 A q_0)(q_0 A q_0) \mid 1(q_0 A q_1)(q_1 A q_0)$

$(q_0 A\ q_1) \rightarrow 1(q_0 A q_0)(q_0 A q_1) \mid 1(q_0 A q_1)(q_1 A q_1)$

$(q_0 A q_1) \rightarrow 0$.

convert PDA to CFG. PDA is given by

$P = (\{p, q\}, \{0, 1\}, \{x, z\}, \delta, q, z)$, δ is defined by

$\delta(p, 1, z) = \{(p, xz)\}$

$\delta(p, \epsilon, z) = \{(p, \epsilon)\}$

$\delta(p, 1, x) = \{(p\ xx)\}$

$\delta(q, 1, x) = (q, \epsilon)$

$\delta(p, 0, x) = (q, x)$

$\delta(q, 0, z) = (p, z)$

Solution:

For the start state the production rule can be.

1) $S \rightarrow [p, z, p]$

2) $S \rightarrow [p, z, q]$

$\delta(p, 1, z) = (p, xz)$

$\delta(q_i, a, z_0) = (q_{i+1}, z_1, z_2)$ gives

$\delta(q_i\ z_0\ q_{i+k}) \rightarrow a(q_{i+1}\ z_1\ q_m)(q_m\ z_2\ q_{i+k})$

$$\boxed{q_i = p,\ a = 1,\ z_0 = z,\ q_{i+1} = p,\ \overset{=x}{z_1},\ z_2 = z}$$

3) $\delta(p, z, p) \rightarrow 1\ (p, x, p)(p, z, p) \mid 1\ (p, x, q)(q, z, p)$

4) $\delta(p, z, q) \rightarrow 1\ (p, x, p)(p, z, q) \mid 1\ (p, x, q)(q, z, q)$

Scanned by CamScanner

$\delta(P, \epsilon, Z) = (P, \epsilon)$

$\delta(q_i, a, z_0) = (q_{i+1}, \epsilon)$ then

$(q_i\ z_0\ q_{i+1}) \rightarrow a$

$$\boxed{q_i = P,\quad a = \epsilon,\quad z_0 = Z,\quad q_{i+1} = P,\quad \epsilon \rightarrow \epsilon}$$

5) $(P\ Z\ P) \rightarrow \epsilon$

$\delta(P, 1, x) = (P\ x\ x)$

$$\boxed{q_i = P,\quad a = 1,\quad z_0 = x,\quad q_{i+1} = P\quad z_1 = x\quad z_2 = x}$$

6) $(P, x, P) \rightarrow 1\,(P, x, P)\,(P, x, P) \mid 1\,(P, x, q)\,(q, x, P)$

7) $(P, x, q) \rightarrow 1\,(P, x, P)\,(P, x, q) \mid 1\,(P, x, q)\,(q, x, q)$

$\delta(q, 1, x) = (q, \epsilon)$

$\delta(q_i, a, z_0) = (q_{i+1}, \epsilon)$ then

$(q_i, z_0, q_{i+1}) \rightarrow a$

$$\boxed{q_i = q,\quad a = 1,\quad z_0 = x,\quad q_{i+1} = q,\quad \epsilon \rightarrow \epsilon}$$

8) $(q, x, q) \rightarrow 1$

$\delta\{P, 0, x) = (q, x)$

$q_i = P,\ a = 0,\ z_0 = x,\ q_{i+1} = q,\ z_1 = x$

9) $(P, x, P) \rightarrow 0\,(q, x, P)$

10) $(P, x, q) \rightarrow 0\,(q, x, q)$

$\delta(q, 0, Z) = (P, Z)$

11) $(q, Z, P) \rightarrow 0\,(P, Z, P)$

12) $(q, Z, q) \rightarrow 0\,(P, Z, q)$

Scanned by CamScanner

Pumping Lemma for CFL

- The pumping lemma for regular sets states that every sufficiently long string in a regular set contains a short substring that can be pumped.

- If a long string is given and if we push, pump any number of substrings in any number of times then we always get a regular set.

Lemma.

Let L be any context free language, then there is a constant n, which depends only upon L, such that there exist a string $W \in L$ and $|W| \geq n$ where $w = pqrst$ such that

1. $|qs| \geq 1$
2. $|prs| \leq n$
3. $i \geq 0$ $pq^i r s^i t$ is in L

① Use Pumping lemma to prove that the following is not CFL

$$\{ a^n b^m a^n b^{n+m} \mid m,n \geq 0 \}$$

Solution:

$L = \{a^n b^m a^n b^{n+m}\}$ can be written as

$L = \{a^n b^m a^n b^n b^m\}$

case 1:

$z = a^n b^m a^n b^n b^m \in L$

we will map $z = pqrst$ to given z

$$z = \underbrace{\overbrace{a\ldots a}^{n}}_{p}\ \underbrace{\overbrace{b\ldots b}^{m}\ \overbrace{a\ldots a}^{n}\ \overbrace{b\ldots b}^{n}}_{qrs}\ \underbrace{\overbrace{b\ldots b}^{m}}_{t}$$

by pumping lemma, $z = pq^i rs^i t \in L$

Let $i=2$ the z becomes

$z = pqqrsst$

$z = a^n b^{m+t} a^{n+u} b^{n+v} b^m \in a^n b^m a^n b^n b^m$

Hence $z \notin L$

case 2: Strings of a's only

$$z = \underbrace{\overbrace{a\ldots a}^{n}}_{pqrs}\ \underbrace{\overbrace{b\ldots b}^{m}\ \overbrace{a\ldots a}^{n}\ \overbrace{b\ldots b}^{n}\ \overbrace{b\ldots b}^{m}}_{t}$$

Pumming lemma $z = pq^i rs^i t$. if $i=2$ then $z = pq^2 rs^2 t$.

~~$z = a^n b^m a^n b^n b^{m+t+v} \notin a^n b^m a^n b^n b^m$~~

$z = a^{n+t+v} b^m a^n b^n b^m \notin a^n b^m a^n b^n b^m$

Scanned by CamScanner

case 3: strings of b's only

$$z = \underbrace{\overbrace{a\ldots a}^{n}\ \overbrace{b\ldots b}^{m}\ \overbrace{a\ldots a}^{n}\ \overbrace{b\ldots b}^{n}}_{\downarrow\ b}\ \underbrace{\overbrace{b\ldots b}^{m}}_{qrst}.$$

$z = pq^i rs^i t$. if $i=2$ then $z = pq^2 rs^r t$.

$$z = a^n b^m a^n b^n b^{m+t+v} \notin a^n b^m a^n b^n b^m$$

② Show that $L = \{a^n b^n c^n \mid n \geq 0\}$ is not a context fr

language

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

$$w = \overbrace{a\ldots a}^{n}\ \overbrace{b\ldots b}^{n}\ \overbrace{c\ldots c}^{n}$$

Case 1:

$$w = \overbrace{\underbrace{a\ldots a}_{p}}^{n}\ \overbrace{\underbrace{b\ldots b}_{qrs}}^{n}\ \overbrace{\underbrace{c\ldots c}_{t}}^{n}$$

$w = pq^i rs^i t \in L$ when $i \geq 0$

if we assume $i=0$

$$w = \overbrace{\underbrace{a\ldots a}_{p}}^{n}\ \overbrace{\underbrace{b\ldots b}_{\underbrace{q^0 r s^0}_{\downarrow\ r}}}^{n}\ \overbrace{\underbrace{c\ldots c}_{t}}^{n}$$

$$w = a^n b^{n-m-k} c^n \notin a^n b^n c^n$$

Hence $w1 \notin L$

case 2:

$$w = \underbrace{\overbrace{a\ldots a}^{n}}_{pqrs}\ \underbrace{\overbrace{b\ldots b}^{n}\ \overbrace{c\ldots c}^{n}}_{t}$$

$W = pq^i rs^i t \in L$. if $i=2$

$$w = \underbrace{\overbrace{a\ldots a}^{n+m+k}}_{pq^2rs^2}\ \underbrace{\overbrace{b\ldots b}^{n}\ \overbrace{c\ldots c}^{k}}_{t}$$

$W = a^{n+m+k} b^n c^n \notin L$.

closure properties of context Free Languages

1. The context free languages are closed under union

2. The context free languages are closed under concatenation

3. The context free languages are closed under kleen closure.

4. The context free languages are not closed under intersection.

5. The context free languages are not closed under complement.

1) Pump a Substring so that we are getting a newstring and checking new string is presented a language are not.

1. string $z = pqrst$

2. $|z| \geq n$

3. $|qs| \geq 1$

4. $|qrs| \leq n$

5. $pq^irs^it \notin L$

1) $L = \{a^p \mid p \text{ is a prime number}\}$ is not CFL

Solution

prime numbers are 2, 3, 5, 7, 11, 13

Take $L = \{a^2, a^3, a^5, a^7 \ldots\}$

$L = \{aa, aaa, aaaaa, aaaaaaa \ldots\}$

Apply the rules.

1. Take any string $z = pqrst$ $\quad p=a,\ q=a,\ r=a,\ s=a,\ t=a$

$z = aaaaa$

Here n is 5

2. $|z| \geq n$

$|aaaaa| \geq 5$

$5 \geq 5$

3. $|qs| \geq 1$ $\quad q=a,\ s=a$

$|aa| \geq 1$

$|2| \geq 1$

4. $|qrs| \leq n$

$|aaa| \leq n$

$3 \leq 5$

5. $pq^i rs^i t \notin L$

$a a^i a a^i a$ $\quad i=2$

$a a^2 a a^2 a$

$a\ aa\ a\ aa\ a$

$a^7 \in L$ it is CFL

$i=3$

$a a^3 a a^3 a$

$a\ aaa\ a\ aaa\ a$

$a^9 \notin L$

Hence proved.

UNIT-4

TURING MACHINE

Turing Machine

- Alan Turing is father of a model which has computing capability of general purpose computer.

Features.

1. It has external memory which remembers arbitrarily long sequence of input
2. It has unlimited memory capability
3. The model has a facility by which the input at left or right on the tape can be read easily.

4. The machine can produce certain output based on its input.

turing Machine is a collection of component

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \Delta \text{ or } B, F)$$

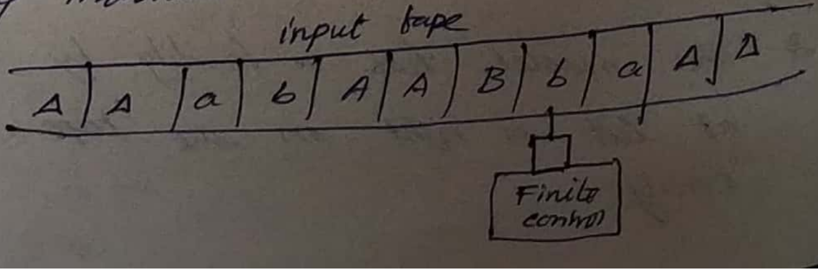
1. Q is a Finite set of States
2. Γ is Finite set of external symbols
3. Σ is a Finite set of input symbols
4. Δ or b or $B \in \Gamma$ blank symbol.
5. δ is a transition or mapping function
6. q_0 be the initial State.
7. F is a set of Final State.

Basic Model

1. The i/p tape having infinite number of cells, each cell containing one i/p symbol. The empty tape is filled by blank character.

2. The Finite control and the tape head which is responsible for reading the current i/p symbol. The will move left or right.

3. Finite set of symbols called external symbols which are used in building the logic of turing machine.



Techniques of Turing Machine construction

- Turing machine is a process of writing out the complete set of states and next move function.

- This is a totally conceptual phenonmenon.

- The turing machine can be designed with the help of some conceptual tools.

Storage in Finite control

The Finite control can be used to hold some amout of information.

The Finite automata stores the information in pair of elements such as the current state and the current symbol pointed by the tape head.

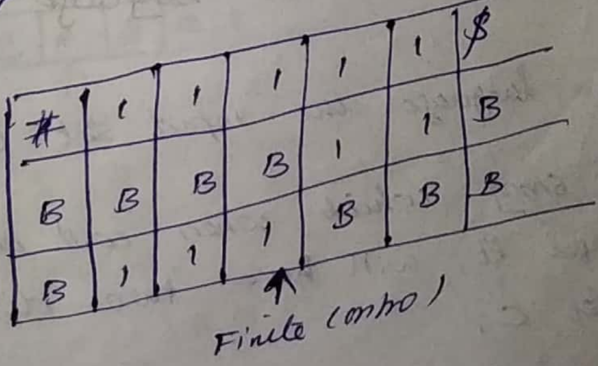
$$\delta([q_0, 0], 1) \rightarrow ([q_1, 1], \Delta, R)$$

→ initial state q_0 and stores the current symbol

◦ if it reads the symbol 1 then the machine goes to next state q_1 replace that 1 by Δ moves to right.

Multiple Tracks

If the input tape is divided into multiple tracks.



#	1	1	1	1	1	$
B	B	B	B	1	1	B
B	1	1	1	B	B	B

- The input tape has multiple tracks on the First track The input which is placed is Surrounded by # and $.

- The unary number equivalent to 5 is Placed on the tape. on the First track.

- on the second track unary 2 is Placed.

- If we construct a TM Which subtracts 2 from 5 we get the answer on the third track and that is 3 in unary form.

checking off symbols

- It is an effective way of recognizing the language by TM.

- The symbols are to be placed on the input tape.
- The symbol which is read is marked by any special character.

- The tape head can be moved to the right or left.

① Construct a turing machine $M=(Q,\Sigma,\Gamma,\delta,q_0,B,F)$ which recognizes the language $L=\{WcW \mid W\in(a+b)^+\}$

- This language the input set is $\Sigma=\{a,b\}$.

- The string which when will be placed on the input tape it will have two distinct parts separated by letter c.

a	b	a	c	a	b	a	Δ	Δ	. . .

a	b	a	c	a	b	a	Δ
↑							

Read a and mark it

*	b	a	c	a	b	a	Δ
	↑						

*	b	a	c	a	b	a	Δ
		↑					

*	b	a	c	a	b	a	Δ
			↑				

*	b	a	c	a	b	a	Δ
				↑			

It is same as marked one

*	b	a	c	*	b	a	Δ
			↑				

*	b	a	c	*	b	a	Δ
		↑					

*	b	a	c	*	b	a	Δ
	↑						

*	b	a	c	*	b	a	Δ
↑							

*	*	a	c	*	b	a	Δ
	↑						

Mark it

*	*	a	c	*	b	a	Δ
		↑					

*	*	a	c	*	b	a	Δ
			↑				

*	*	a	c	*	b	a	Δ
				↑			

*	*	a	c	*	b	a	Δ
					↑		

It is same as marked one

*	*	a	c	*	*	a	Δ
					↑		

*	*	a	c	*	*	a	Δ
				↑			

*	*	a	c	*	*	a	Δ
			↑				

*	*	a	c	*	*	a	Δ
		↑					

*	*	a	c	*	*	a	Δ
	↑						

*	*	a	c	*	*	a	Δ
↑							

*	*	a	c	*	*	a	Δ
	↑						

*	*	a	c	*	*	a	Δ
		↑					

Mark it

*	*	*	c	*	*	a	Δ
		↑					

*	*	*	c	*	*	a	Δ
			↑				

*	*	*	c	*	*	a	Δ
				↑			

*	*	*	c	*	*	a	Δ
					↑		

*	*	*	c	*	*	a	Δ
						↑	

Same as marked one

*	*	*	c	*	*	*	Δ
						↑	

*	*	*	c	*	*	*	Δ
					↑		

*	*	*	c	*	*	*	Δ
				↑			

*	*	*	c	*	*	*	Δ
			↑				

*	*	*	c	*	*	*	Δ
		↑					

Now machine goes accept state.

Subroutine

In the high level languages use of subroutines built the modularity in the program development process.

The same type of concept can be introduced in construction of TM.

Construct a TM for the subroutine $f(a,b)=a*b$ Where a and b are unary numbers.

The unary number are represented by 1's.

$a=2$ & $b=3$

1	1	0	1	1	1	0	B	B

```
1 1 0 1 1 1 0 B
↑

X 1 0 1 1 1 0 B      Moving to right upto 0
  ↑

X 1 0 1 1 1 0 B
    ↑

X 1 0 1 1 1 0 B      copy this 1 after B by
      ↑              marking it as Y

X 1 0 Y 1 1 0 1 B    Now move to left upto Y
              ↑

X 1 0 Y 1 1 0 1 B
      ↑

X 1 0 Y 1 1 0 1 B    Mark this 1 as Y and
        ↑            copy this 1 at the right most
                     end.

X 1 0 Y Y 1 0 1 1 B  Now move to left upto Y
                ↑

X 1 0 Y Y 1 0 1 1 B
        ↑
```

$x\;1\;0\;y\;y\;\underset{\uparrow}{1}\;B\;1\;1\;B$ Mark this 1 as y and copy this 1 at the right end

$x\;1\;0\;y\;y\;y\;B\;1\;1\;1\;B$ Now move to left upto y

$x\;1\;0\;y\;y\;\underset{\uparrow}{y}\;B\;1\;1\;1\;B$

Now head is pointing to y next is 0 that means all the 1's are over we will convert these y's to 1's and repeat the same procedure.

$x\;\underset{\uparrow}{1}\;0\;1\;1\;1\;0\;1\;1\;1$

$x\;\underset{\uparrow}{x}\;0\;1\;1\;1\;0\;1\;1\;1$ convert this 1 to x nomove to right upto after the 0 symbol

$x\;x\;0\;\underset{\uparrow}{1}\;1\;1\;0\;1\;1\;1$ Mark it y and copy this 1 to rightmost end.

$x\;x\;0\;y\;1\;1\;0\;1\;1\;1\;\underset{\uparrow}{1}$ move to left upto y

$x\;x\;0\;y\;\underset{\uparrow}{1}\;1\;0\;1\;1\;1\;1$

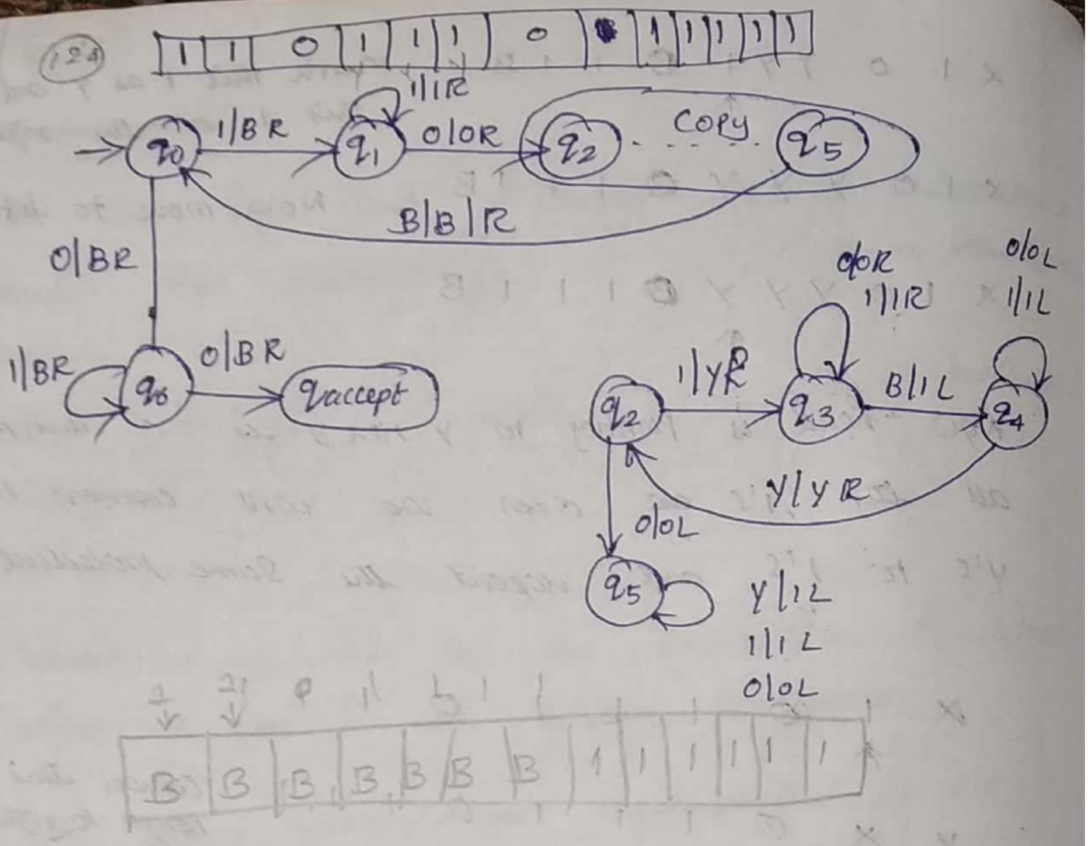
$x\;x\;0\;y\;y\;1\;0\;1\;1\;1\;1\;\underset{\uparrow}{1}$

$x\;x\;0\;y\;y\;\underset{\uparrow}{1}\;0\;1\;1\;1\;1\;1$

$\boxed{x\;x\;0\;y\;y\;y\;0\;1\;1\;1\;1\;1\;\underset{\uparrow}{1}}$

Now we will move to left by converting all y's and x's to 1's.

$$\underbrace{1\;1}_{a}\;0\;\underbrace{1\;1\;1}_{b}\;0\;\underbrace{1\;1\;1\;1\;1\;1}_{answer.}\;B$$



TM is accepted.

Computational Languages and Functions

- The turing machine accepts all the languages even through there are recursively enumerable

- Recursive means repeating the same set of rules for any number of times.

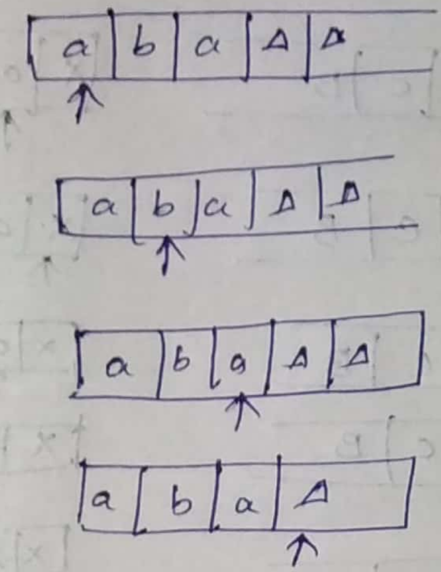
- Enumerable means a list of elements.

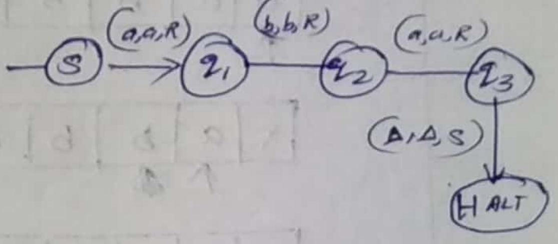
- The TM also accepts the computable function.
    - Addition
    - multiplication
    - Subtraction
    - division
    - power of function
    - square function

① construct a turing machine which accepts the language of aba over $\Sigma = \{a, b\}$.

solution:

TM is only for $L = \{aba\}$



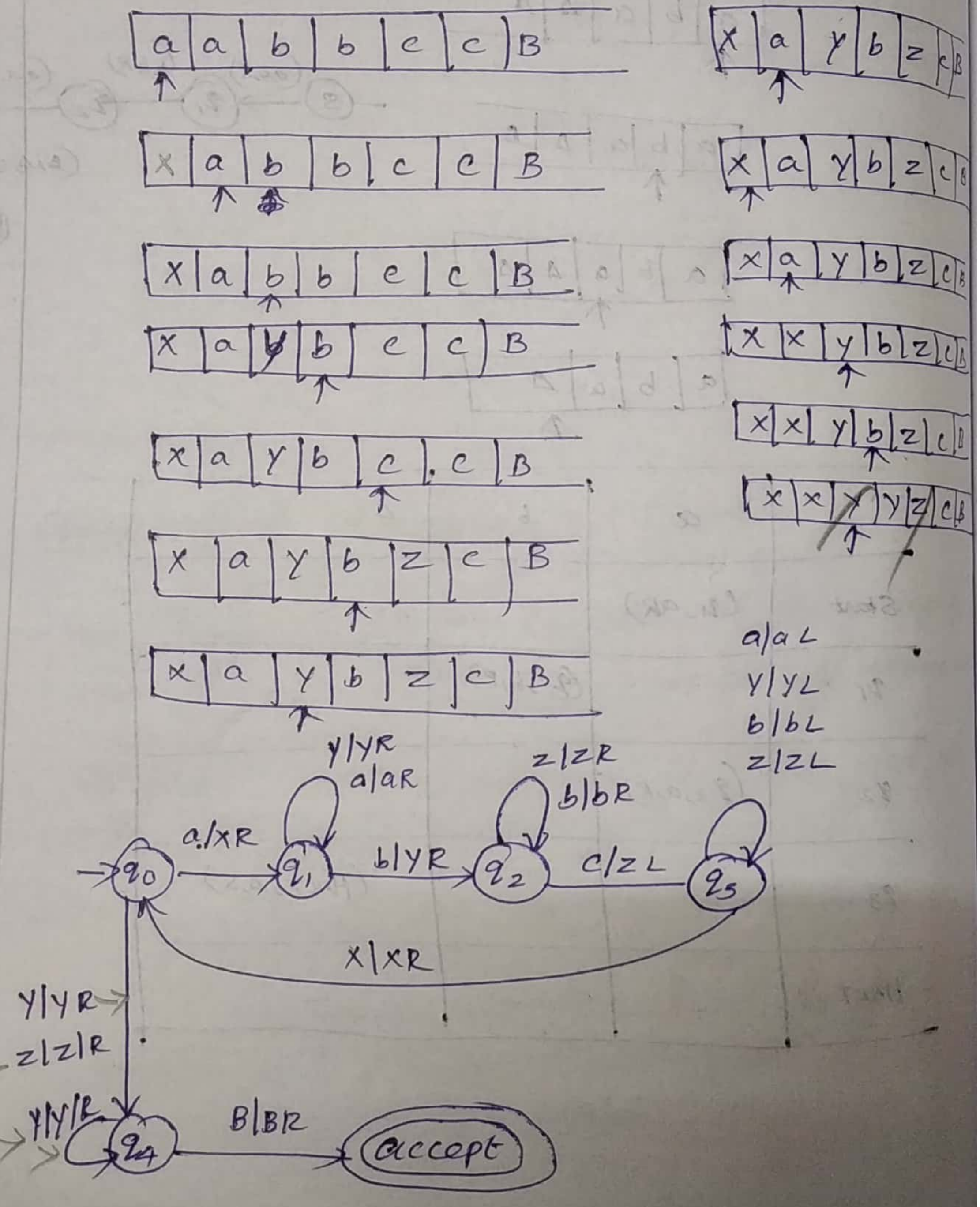


	a	b	Δ
Start	(q_1, aR)		
q_1		(q_2, b, R)	
q_2	(q_3, a, R)		
q_3			(HALT, Δ, S)
HALT	–	–	–

② construct TM for the language $L=\{a^n b^n c^n \mid n \geq 1\}$

∴ $n \geq 1$ ⇒ abc, aabb,cc, aaa,bbb,ccc

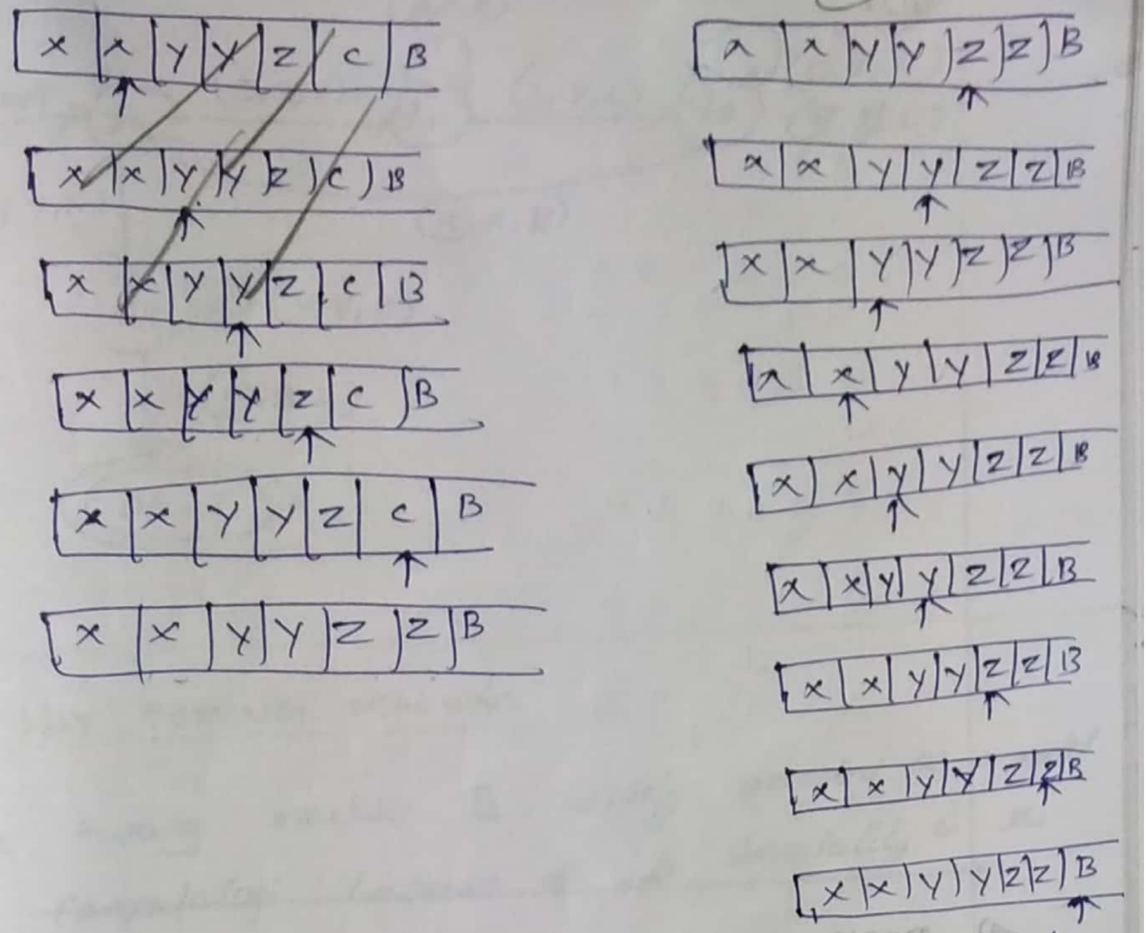
$n=2$



$\delta(q_4, b) = (q_{reject}, b, R)$

$\delta(q_4, c) = (q_{reject}, c, R)$

$\delta(q_1, z) =$
$\delta(q_1, c) =$
$\delta(q_1, B)$
} q reject.

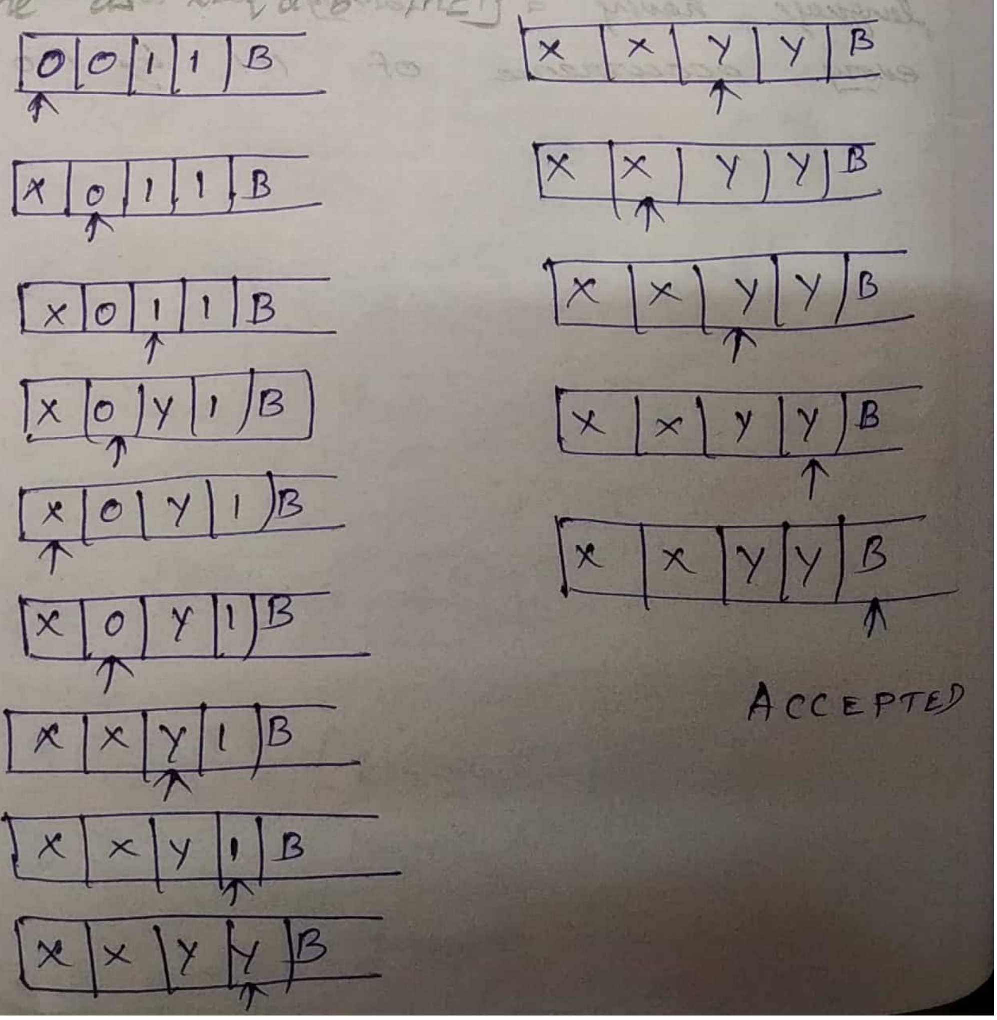

x | x | Y | Y | Z | c | B
x | x | Y | Y | Z | C | B
x | x | Y | Y | Z | c | B
x | x | Y | Y | Z | C | B
x | x | Y | Y | Z | c | B
x | x | Y | Y | Z | Z | B
x | x | Y | Y | Z | Z | B
x | x | Y | Y | Z | Z | B
x | x | Y | Y | Z | Z | B
x | x | y | y | Z | Z | B
x | x | y | Y | Z | Z | B
x | x | Y | Y | Z | Z | B
x | x | Y | Y | Z | Z | B
x | x | Y | Y | Z | Z | B
x | x | Y | Y | Z | Z | B

Accepted.

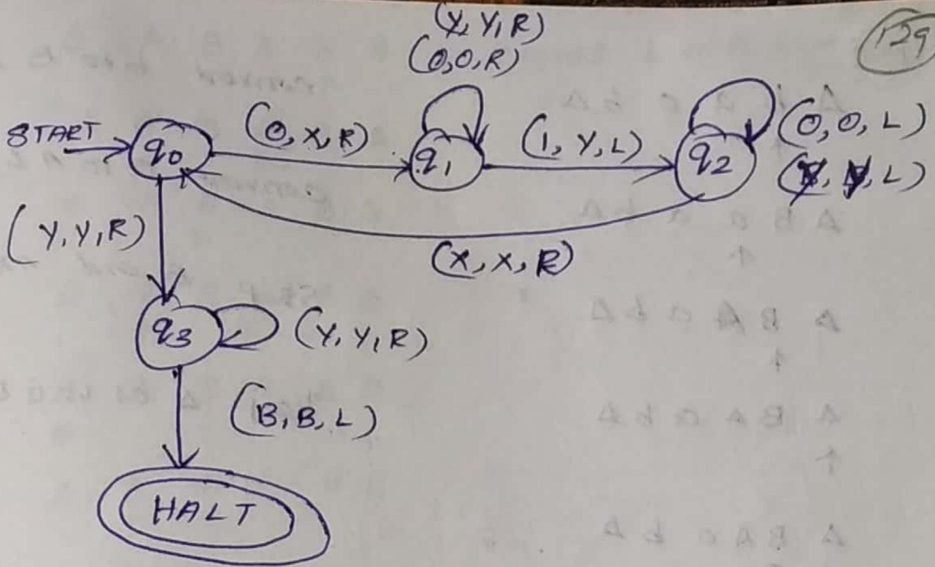
③ Design a turing machine which recognizes the input language having a substring as 101 and replaces every occurrence of 101 by 110

4) Design a Turing Machine to accept the language $L = \{0^n 1^n \mid n \geq 1\}$ and simulate it's action on the input 0011

Same as $L = \{a^n b^n \mid n \geq 1\}$



ACCEPTED

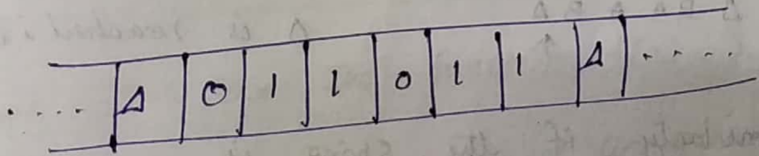


TWO WAY TURING MACHINE

- The turing machine is widely accepted as a model of Computation because of it's Versatality in the Power of Computation.

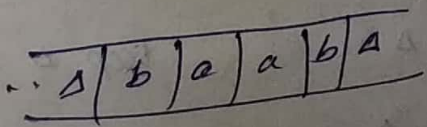
- The TM can perform string Operations, arithmetic computations, recognizing languages either regular or non regular language.

- The TWO way TM is more power full.



① Construct a TM for a language having equal number of a's and b's in it over the input set Σ={a,b}

Solution:

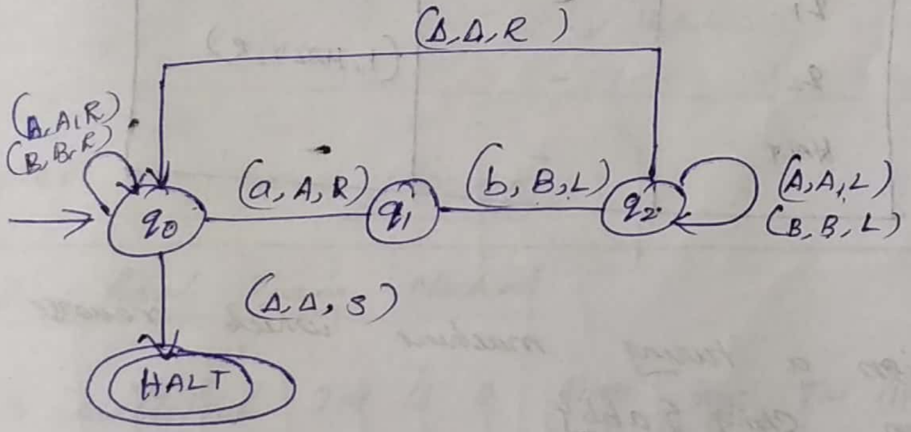


```
Δ b a a b Δ          convert b to B & move right.
  ↑

Δ B a a b Δ          convert a to A & move left
    ↑

Δ B A a b Δ          Strip B and move left
  ↑

Δ B A a b Δ          keep Δ as it is & move right
↑

Δ B A a b Δ
  ↑

Δ B A a b Δ
    ↑
                     convert a to A & move left
Δ B A a b Δ
      ↑

Δ B A A b Δ
    ↑

Δ B A A b Δ
  ↑

Δ B A A b Δ          keep Δ as it is & move right
↑

Δ B A A b Δ
  ↑

Δ B A A b Δ
    ↑

Δ B A A b Δ
      ↑
                     Convert b to B & move right
Δ B A A b Δ
        ↑

Δ B A A B Δ          Δ is reached, goto HALT state.
          ↑
```

Similarly if the string is

```
Δ a b a b Δ
  ↑                  Convert a to A move right

Δ A b a b Δ          Convert b to B move left
    ↑

Δ A B a b Δ
  ↑

Δ A B a b Δ          keep Δ as it is & move right
↑

Δ A B a b Δ
  ↑

Δ A B a b Δ
    ↑                convert a to A & move right

Δ A B a b Δ
      ↑
```

```
Δ A B A b Δ        convert b to B & move left
        ↑

Δ A B A B Δ
      ↑

Δ A B A B Δ
    ↑

Δ A B A B Δ
  ↑

Δ A B A B Δ
↑

Δ A B A B Δ
  ↑

Δ A B A B Δ
    ↑

Δ A B A B Δ
      ↑

Δ A B A B Δ
        ↑

Δ A B A B Δ        Δ is reached goto Halt State.
          ↑
```



② Construct TM for the function $f(x) = x+3$

we will represent x using unary number.

The input set $\Sigma = \{1\}$. If we assume $x=4$

1	1	1	1	Δ	Δ	Δ . . .

```
↑

1 1 1 1 Δ Δ Δ        move right
  ↑

1 1 1 1 Δ Δ Δ
    ↑

1 1 1 1 Δ Δ Δ
      ↑
```

```
1  1  1  1  Δ  Δ  Δ .
            ↑
```

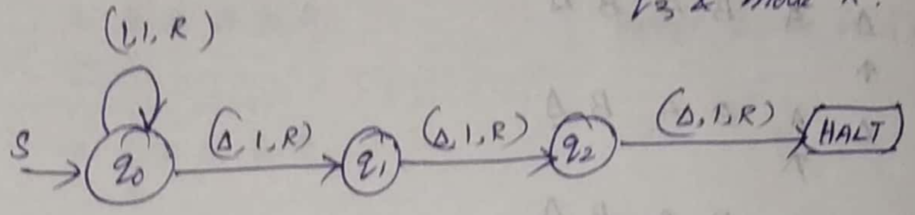
Replace Δ by 1 go to state q1 & move right

```
1  1  1  1  1  Δ  Δ
               ↑
```

Replace Δ by 1 go to state q2 & move right

```
1  1  1  1  1  1  Δ
                  ↑
```

RePlace Δ by 1 goto state q3 & move R.



State \ I/P	1	Δ
q0	(1,q0,R)	(1,q1,R)
q1	-	(1,q2,R)
q2	-	(1,HALT,R)
HALT	-	-

③ Design a turing machine which reverse the given string {abb}.

Solution:

The i/p head will move right at the end of the string. Then copy each letter from left to right at the end after Δ.

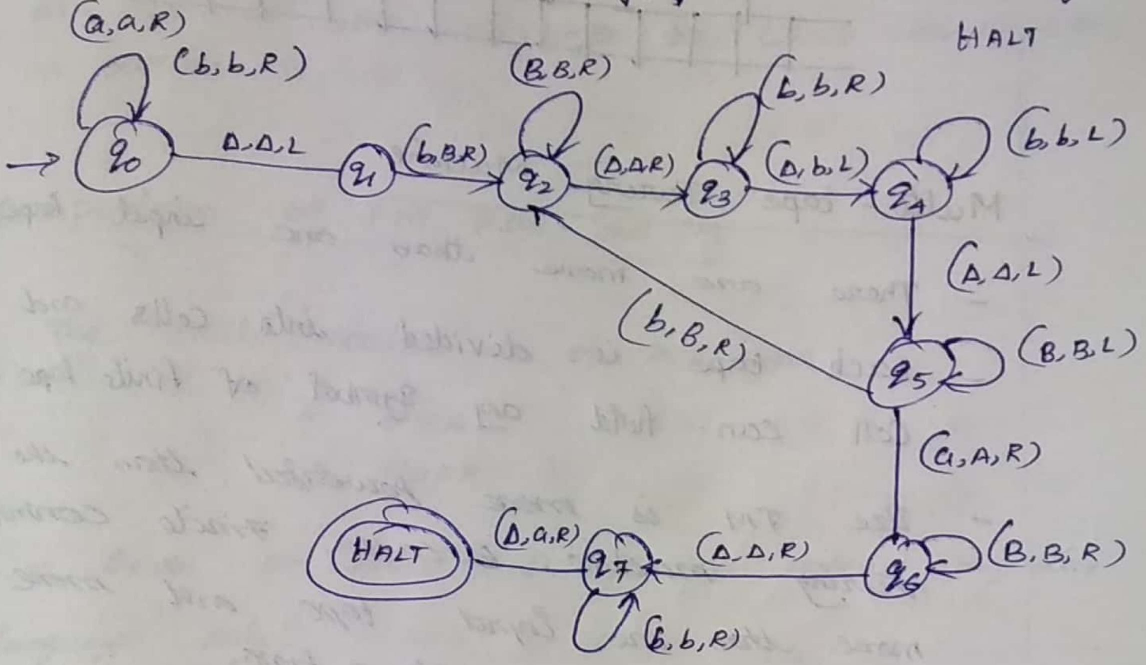
```
Δ a b b Δ  →  Δ a b b Δ  →  a b b Δ  →  a b b Δ →
  ↑               ↑             ↑             ↑

Δ a b b Δ  →  Δ a b B Δ  →  Δ a b B Δ bΔ  →  Δ a b B Δ b
    ↑                 ↑                 ↑                ↑

Δ a b B Δ bΔ  →  Δ a b B Δ bΔ  →  Δ a B B Δ bΔ  →  Δ a B B Δ b
      ↑                ↑                  ↑                ↑

Δ a B B Δ b Δ  →  Δ a B B Δ b bΔ  →  Δ a B B Δ b bΔ  →  a B B Δ b
          ↑                   ↑              ↑                ↑
```

$\Delta a B \underset{\uparrow}{B} \Delta b b \Delta \rightarrow \Delta a \underset{\uparrow}{B} B \Delta b b \Delta \rightarrow \Delta \underset{\uparrow}{a} B B \Delta b b \Delta$

$\Delta A \underset{\uparrow}{B} B \Delta b b \Delta \rightarrow \Delta A B \underset{\uparrow}{B} \Delta b b \Delta \rightarrow \Delta A B B \underset{\uparrow}{\Delta} b b \Delta \rightarrow \Delta A B B \Delta \underset{\uparrow}{b} b \Delta$

$\Delta A B B \Delta b \underset{\uparrow}{b} \Delta \rightarrow \Delta A B B \Delta b b \underset{\uparrow}{a} \underset{\uparrow}{\Delta} \rightarrow \Delta A B B \Delta b b a \underset{\uparrow}{\Delta} \downarrow$

HALT



TYPES OF TURING MACHINE

Multi head Turing Machine

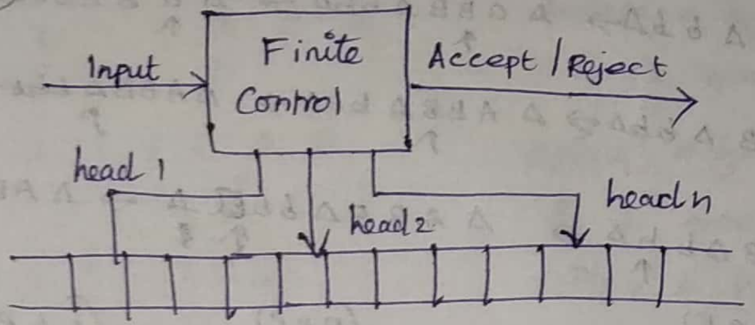
- A multihead TM is a single tape TM having n heads reading symbols on the same tape.

- In one step all the heads sense the scanned symbols and move or write independently.

- The heads are numbered 1 through n and move of TM depends upon the state and the symbol scanned by each head.

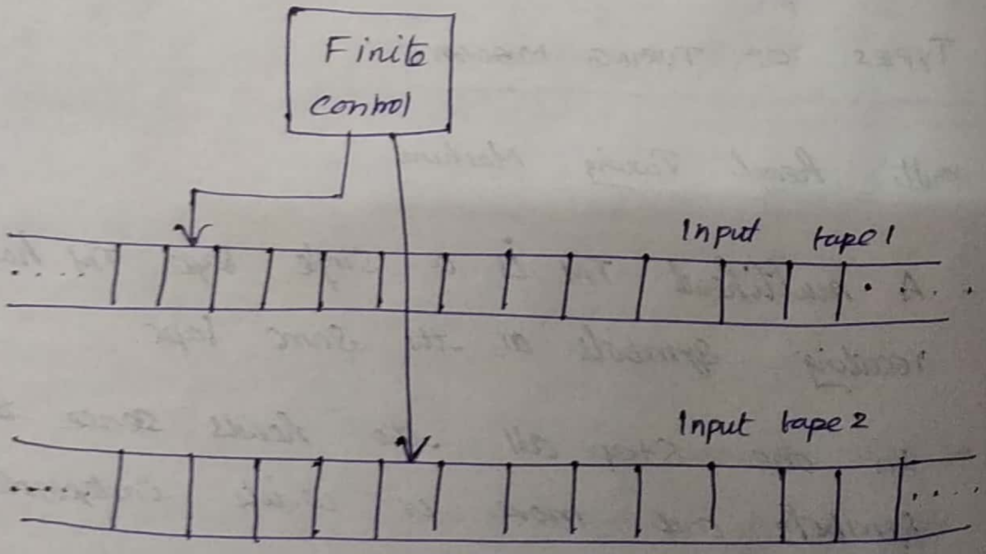
- one move the heads may move left, right or remain stationary.

- This type of Turing machine is as powerful as one tape Turing Machine.



Multi - tape Turing Machine

- There are more than one input tapes.
- Each tape is divided into cells and each cell can hold any symbol of finite tape alphabet
- The TM is more powerful than the basic turing machine. Because finite control reads more than one input tape and more symbols can be scanned at a time.



Power of Turing Machine

- The turing machine has a great computational capabilities.
- It can be used as a general mathematical model for modern computers.

- Turing machine can model even recursive enumerable languages.

- The advantage of Turing machine is that it can model all the computable functions as well as the languages for which the algorithm is possible.

Comparison of FM, PDA and TM

1. The Finite machine is of two types -

   * DFA
   * NDFA

- Both of these DFA & NFA accept regular language only.

- Both of the machines have equal power.

  DFA = NFA

2. The Push down automata consists of two types

   * Deterministic PDA
   * Non deterministic PDA

- The advantage of PDA over FA is that PDA has a memory and hence PDA accepts large class of languages than FA hence PDA has more power than FA.

3. The class of two stack or n- stack PDA has more power than one stack DPDA or NPDA. Hence two - stack / n- stack PDAs are more Powerful.

Scanned by CamScanner

4. Turing machines can be Programmed. The TM accepts very very large class of languages. therefore is the most Powerful computational model.

TM > PDA > FM.

TM accepts regular and non-regular languages.

Halting Problem

The halting problem we will consider the given configuration of a turing machine. The output of TM can be

(i) Halt : The machine starting at this configuration will halt after a finite number of states.

(ii) No Halt : The machine starting at this configuration never reaches a halt state, no matter how long it runs.

- Given any Functional matrix, input data tape and initial configuration, then is it possible to determine whether the process will ever halt?

That means we are asking for a procedure which enable us to solve the halting problem for every pair (machine, tape).

The answer is "no". That is the halting problem is unsolvable.

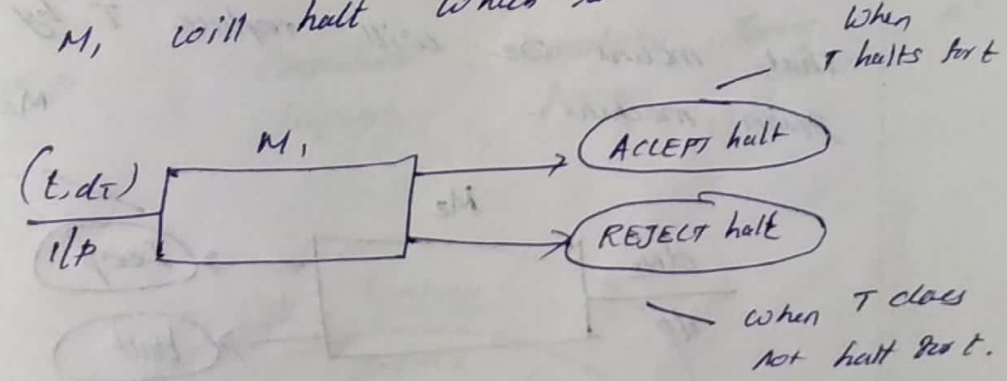
Scanned by CamScanner

We will prove why it is unsolvable.

Let, there exists a TM M_1 which decides whether or not any computation by a TM T will ever halt when a description d_T of T and tape t of T is given [that means the input to machine M_1 will be pair].

Then for every input (t, d_T) to M_1 if T halt for i/p t, M_1 also halts which is called accept halt.

Similarly if T does not halt for i/p t then then M_1 will halt which is called reject halt.



Consider another example.

- Turing Machine M_2 which takes an i/p d_T.

It first copies d_T and duplicates d_T on its tape and then this duplicated tape information is given as i/p to machine M_1.

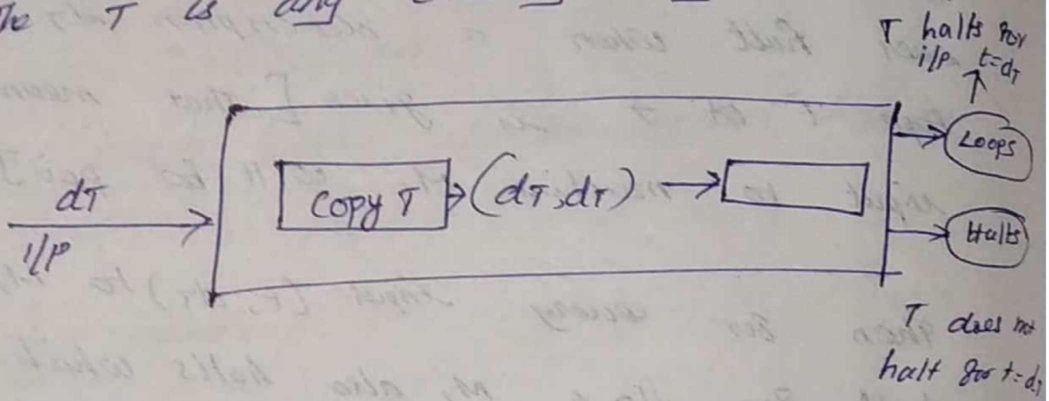
- But machine M_1 is a modified machine with the modification that whenever M_1 is supposed to reach an accept halt. M_2 loops forever.

- ~~M_2 loops~~

Scanned by CamScanner

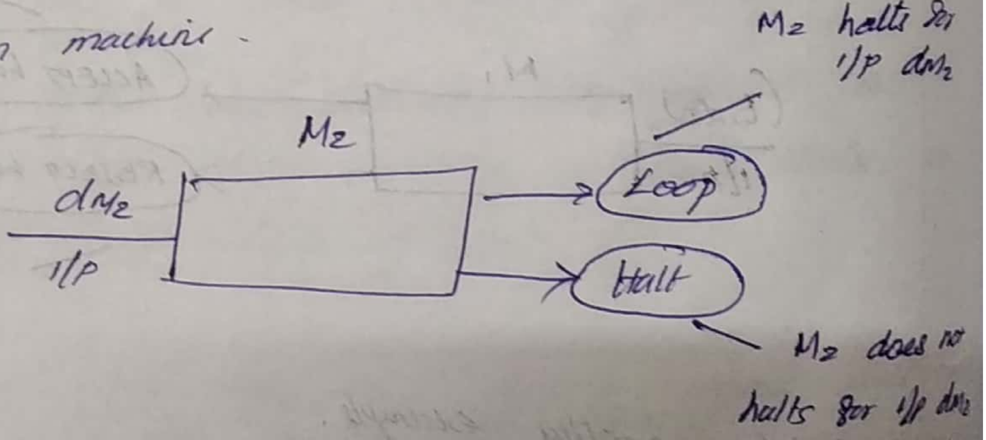
It loops if T halts for input $t = d_T$ and halts if T does not halt for $T = d_T$.

The T is any orbitary turing machine.



As M_2 itself is one turing machine we will take $M_2 = T$.

That means we will replace T by M_2 from given machine.



→ Thus machine M_2 halts for i/p d_{M_2} if M_2 does not halt for input d_{M_2}.

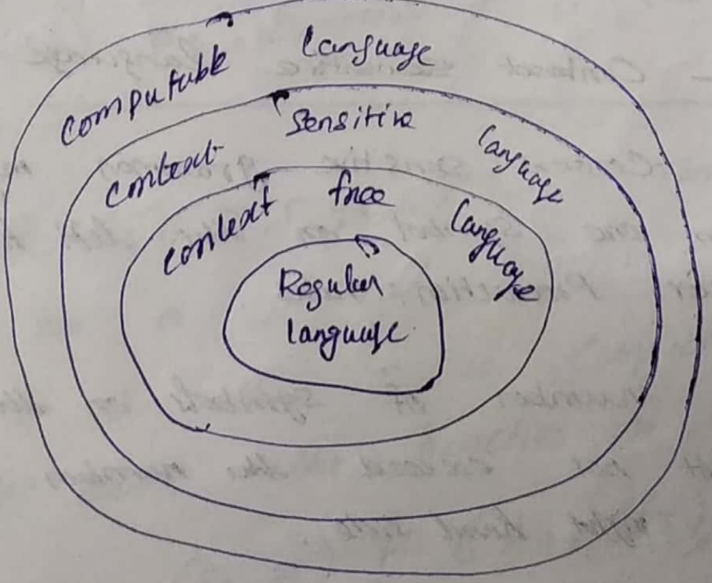
- This is a contradiction. That means a machine M_1 which can tell whether any other TM will halt on particular i/p does not exists.

Hence halting problem is unsolvable.

Chomsky Hierarchy of Languages

- The chomsky's hierarchy represents the class of languages that are accepted by different machine.

Language class	Language	Grammar	Machine	One example
Type 3	Regular	Regular Grammar	FSM ie NFA(or)DFA	a^*b^*
Type 2	context free	Context free grammar	PDA	a^nb^n
Type 1	Decidable language	context sensitive grammar	Linear bounded automata	$a^nb^nc^n$
Type 0	Computable language	unrestricted grammar	Turing machine	$n!$



Chomsky hierarchy

Type 3: Regular language

These language which can be described using regular expressions. theses languages can be modelled by NFA or DFA.

Scanned by CamScanner

Type 2 - context free languages

- It can be represented by context Free Grammar (CFG). The production rule is of the form

$$A \to \alpha$$

where A is any single non-terminal

α is any combination of terminals and non-terminal.

A NFA or DFA cannot recognize strings of this language because these automata are not having "Stack" to memorize.

- Instead of it the Push Down Automata can be used to represent these languages.

Type 1 - Context Sensitive language

1. The context sensitive grammar may have more than one symbol on the left hand side of their Production rules.

2. The number of symbols on the left hand side must not exceed the number of symbols on the right hand side.

3. The rule of the form $A \to \epsilon$ is not allowed unless A is a start symbol. It does not occur on the right hand side of any rule.

- The automation which recognizes context sensitive language is called linear bounded automata while deriving using context sensitive grammar the sentential form must always increase in length every time a production rule is applied. thus the size of a sentential form is bounded by a length of the sentence we are deriving.

Type 0 - Unrestricted languages

- There is no restriction on the grammar rules of these type of language. These language can be effectively modeled by turing machines.

PARTIAL SOLVABILITY

A partial function f on Σ^* may be undefined at certain points (ie the points not in the domain of f)

If the function is defined everywhere on Σ^*, then the function f is called as a total function.

Let $T = (Q, \Sigma, \Gamma, q_0, \delta)$ be a Turing machine. let f be a partial function on Σ^* with value in Γ^*, then T is computable if for every $x \in \Sigma^*$ and the function is defined as

$$(q_0, x) \vdash_T^* (h_a, f(x))$$

and no other $x \in \Sigma^*$ is accepted by T.

If f is a partial function defined on $(\Sigma^*)^k$ with values in Γ^*. T computes f if for every k - tuple $(x_1, x_2 .. x_k)$ at which f is defined.

$$(q_0, x_1, x_2, \ldots x_k) \overset{*}{\underset{T}{\vdash}} (h_a, f(x_1, x_2 \ldots x_k))$$

and no other input which is k-tuple of strings which is accepted by T.

A partial function $f:(\Sigma_1^*)^k \to \Sigma_2^*$ is Turing Computable, for alphabets Σ_1 and Σ_2 and a positi integer k, if there is a Turing machin computing f.

It can be written as

a function $f(x_1, x_2, \ldots x_m)$ is Turing Computable for arguments $a_1, a_2 \ldots a_m$ if there is exists a Turing machine for which

$$(q_0, x, x_2 \ldots x_m) \overset{*}{\vdash} (q_n, f(x_1, .. x_2 .. x_m))$$

Construct a Turing machine to compute zero.

$f(x) = 0$ for all $x \geq 0$

Solution:

Let $x = 3$, $W = 000$

B	0	0	0	B	...

⇒

B	B	B	B	B	...

(i) Whenever T reads 0, change it into blank symbol and move right

(ii) if T reads the end blank symbol, it halts and accepts.

	0	B
→ q_0	(q_0, B, R)	$(q_1, B, Halt)$
q_1		

UNIT-5

Unsolvable Problems and Computational Functions

UNSOLVABILITY

- The TOC we often come across such problems that are answered either yes or NO
- The class of problems which can be answered as yes are is called solvable
- Otherwise the class of problems is said to be unsolvable (or) undecidable.

Primitive Recursive Functions

- Recursive function is class of functions.
- The Recursive function theory begins with some very elementary functions that are intuitively effective.
- Then it provides a few methods for building more complicatev functions from Simpler function.
- That means the computability of given function can be proved using the initial function.

Initial function

The initial functions are the elementary functions whose values are independent of their smaller arguments.

- The zero function : $z(x) = 0$
- The successor function: $s(x) =$ successor of x (roughly, "$x+1$")
- The identity function ; $id(x) = x$
- The zero functions returns zero regardless of its arguments.

The zero and successor functions take only one argument each.

* When it takes one argument it return its argument as its value.

* When it takes more than one argument, it returns on of them.

*

Building Operations: We will build more complex functions from the initial set by using only three methods

(i) Composition

(ii) Primitive recursion

Scanned by CamScanner

(i) composition

We will start with the successor functions

$s(x) = x+1$

Then we may replace its argument, x, with a function. if we replace the argument, x with the zero function.

$z(x)$

then the result is the successor of zero

$s(z(x)) = 1$

$s(s(z(x))) = 2$ and so on.

(ii) primitive recursion

- The second building operation is called primitive recursion.
- Primitive recursion is a method of defining new functions from old function.
- The function h is defined through functions f and g by primitive recursion.

When $h(x,0) = f(x)$

$h(x, s(y)) = g(x, h(x,y))$

- When f and g are known computable functions.
- When h's second argument is zero, the first equation applies.
- When it is not zero, we use the second.

$n! = n*(n-1)$

consider $n=5$

$f(5) = 5*f(4)$

$f(4) = 4*f(3)$

$f(3) = 3*f(2)$

$f(2) = 2*f(1)$

By putting the value of equation for calculating f(2) we will get.

$f(2) = 2*1 = 2$

$f(3) = 3*f(2)$

$= 3*2$

$= 6$

$f(4) = 4*f(3)$

$= 4*6$

$= 24$

$f(5) = 5*f(4)$

$= 5*24$

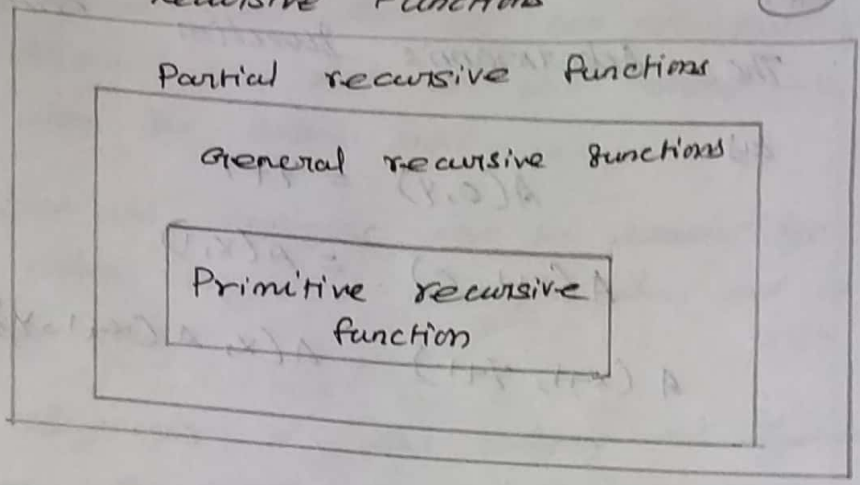
$= 120$

(iii) Minimization

- If g(x) is a function that computes to least x such that f(x)=0, then we know that g is computable.
- Then we can say that g is produced from f by minimization.
- We can build g by minimization only if f(x) is already known to be computable

Scanned by CamScanner

class of Recursive Functions


Partial recursive functions
General recursive functions
Primitive recursive function

1. Partial Recursive function

The function is called partial recursive function if it can be obtained by applying composition, primitive recursion and minimizati. as building operation.

2. General Recursive Function

- It is obtained by applying composition primitive recursive and an unbounded minimization that happen to terminate.

- The general recursive function is a larger class than partial recursive functions.

3. Primitive Recursive Function

- It is obtain by applying composition, primitive recursion and unbounded minimization that does not terminate.

Ex:

- General recursive function is an Ackermann's function.

The Ackermann's function can be defined as

$$A(0,y) = y+1$$

$$A(x+1,0) = A(x,1)$$

$$A(x+1,y+1) = A(x, A(x+1,y))$$

Recursive and Recursively Enumerable Language

- It has two categories
- The first categories consists of all the language that has some algorithm and an algorithm for language L can be modeled by a Turing machine.
- Naturally such turing machine always halts on valid input and enters in accept state.
- But on invalid input not accepted & It halts without entering in halt state.
- Thus languages of this category possess a property of definiteness.
- The languages of this category are particularly called as recursive language.

* Second category

The languages can be modeled by Turing machines but there is no guarantee that the TM will eventually halt.

Scanned by CamScanner

- In the sense that the we can not predict that Turing machine will halt or will enter is an infinite loop for certain input.

- Such type of languages can be denoted by pair (M, w) where M is a Turing machine and w is an input string.

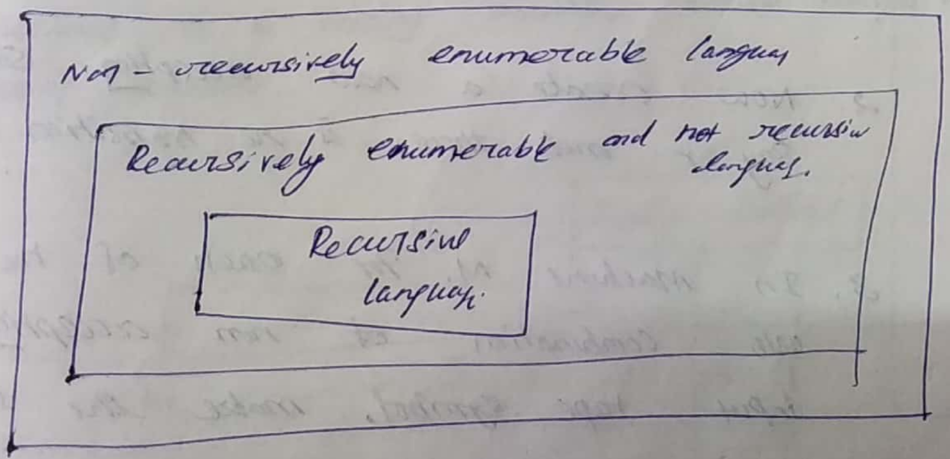
- These languages of this category are typically called as Recursively Enumerable languages.

There are three categories of the languages.

1. Recursive language for which the algorithm exists

2. Recursively enumerable language for which it is not sure that on which input the TM will ever halt.

3. The non recursively enumerable languages for which there is no Turing Machine at all.


Non - recursively enumerable langu
Recursively enumerable and not recursive languages.
Recursive language.

Properties of Recursive and Recursively Enumerable Language.

Decidable & undecidable Languages

- If a language is recursive then it is called decidable language.

- If the language is not recursive then such a language is called undecidable language.

Scanned by CamScanner

Theorem:

If L is recursive language the L' is also a recursive language.

Proof: Let there will be some L that can be accepted by Turing machine M. Hence we can denote language L by L(M). as acceptance of L(M) the machine M always halts.

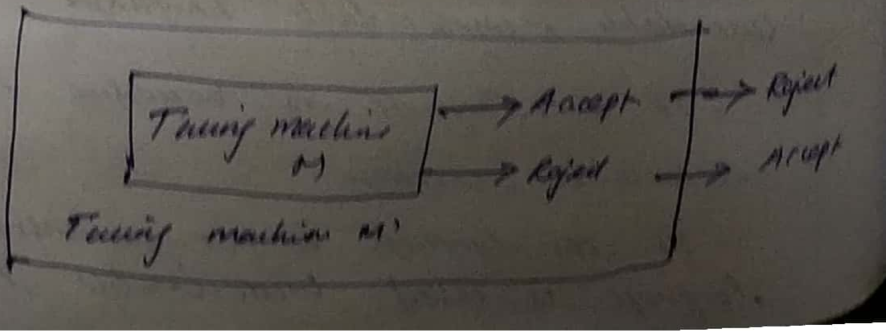
- We construct a TM M' such that L' = (M'). for construct of M' following steps.

1. The accepting steps of M are made non-accepting states of M' and there is no transition from M'.

   That means we have created the states such that M' will halt without accepting.

2. Now create a new accepting state for M' say r and there is no transition from r.

3. In machine M, for each of the transition with combination of non accepting state and input tape symbol, make the same transition having the combination of accepting state and input tape symbol for machine M'.



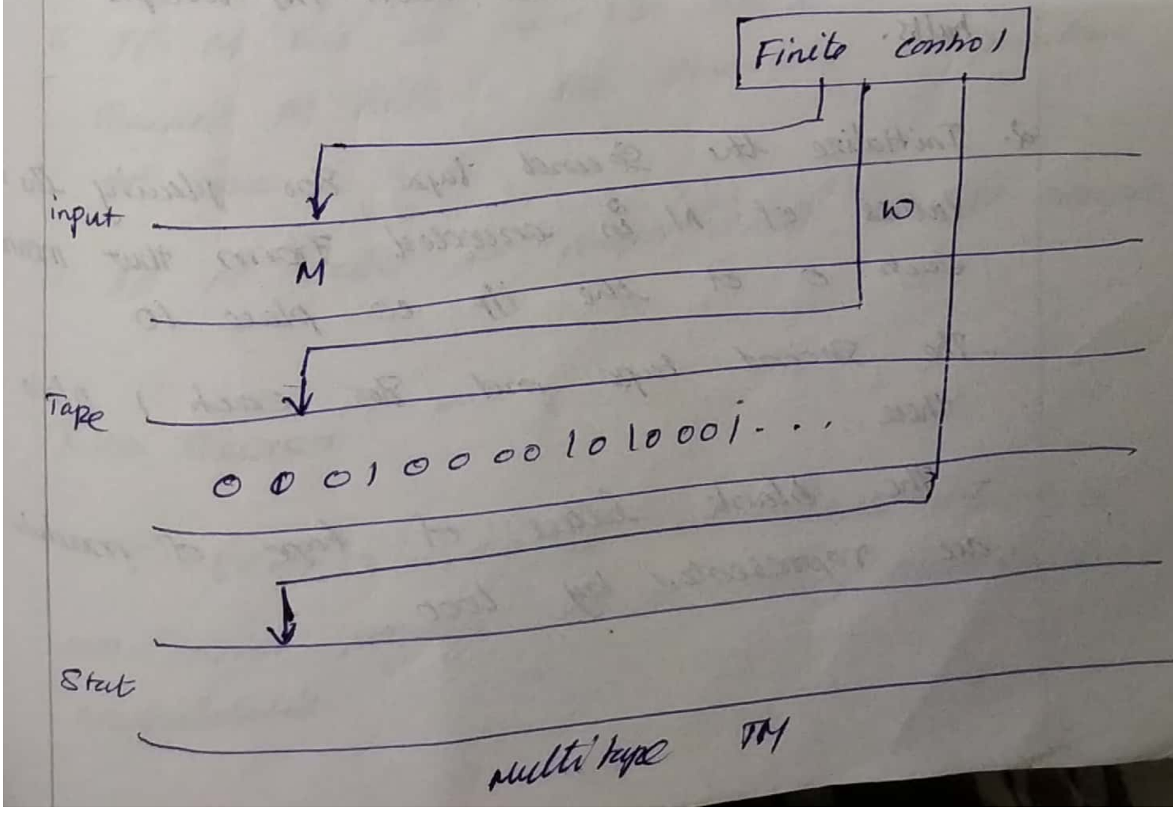
Since M is guaranteed to halt M' is also guaranteed to halt.

In fact, M' accept exactly those strings that M does not accept.

Thus we can say that M' accepts L'.

UNIVERSAL TURING MACHINE

- The universal language Lu is a set of binary strings which can be modeled by a Turing machine.
- The universal language can be represented by a Pair (M, W) where M is a TM that accepts this language and W is a binary string (0+1)* such that W belongs to ~~FM~~ L(M).
- This binary string represents various codes of many Turing machines. Thus the universal Turing machine is a turing machine which accepts many Turing machine.



- The universal Turing machine U accept the TM M

- The transitions of M are Stored initially on First tape along with the String w

- On the second tape the Simulated tape of M is placed.

  Hence symbol x_i of M will be represented by 0^i and tape symbols are sepreted by Single 1's

- On the third tape various States of machine M are Stored. The State q_i is represented by i 0's.

Operations on Turing Machine

1. The First tape is observed carefully whether the code for M is valid or not.

   If the code is not valid than U halt without accepting.

   As invalid codes are assumed to represent the TM with no moves. such TM accepts no i/p's and halts.

2. Initialize the Second tape for placing the tape values of M in encoded Form that means At each 0 of the i/p co place 10

- The Second tape and for each 1 place 100 there.

- The blank letter of tape of machine M are represented by 1000.

Scanned by CamScanner

3. As the thirell tape stone states of Machi M, Place 0 then as the start state of machine M.

The head of third tape will now position at start state and the head for Hard tape 2 will position at the first simulator cell of second tape.

4. Now the move of M can be simulated. As on the first tape transitions are stored U first searches the tape 1 to know the transitions which are always given in the form $0^i 10^j 10^k 10^l 10^m$.

0^i means current state.

0^j means current input tape symbol.

0^k means changed state.

0^l means changed input tape symbol.

0^m means direction in which the head is to be moved.

5. If M has no transition then no transition is performed. Therefore M halts in the simulated configuration and hence U halts

6. If M enters in accepts state, the U accepts.

Rice Theorem:

$L = \{<M> \mid L(M) \in P\}$ is undecidable when P, a non-trivial property of the Turing machine. is undecidable.

If the following two properties hold, it is proved as undecidable.

Property 1: If M_1 and M_2 recognize the same language, then either $\langle M_1\rangle\langle M_2\rangle \in L$ or $\langle M_1\rangle\langle M_2\rangle \notin L$

Property 2: For some M_1 and M_2 such that $\langle M_1\rangle \in L$ and $\langle M_2\rangle \notin L$

Proof :-

Let there are two Turing machine X_1 and X_2

Let us assume $\langle X_1\rangle \in L$ such that

$L(X_1) = \psi$ and $\langle X_2\rangle \notin L$.

For ~~all~~ input 'w' in a particular instant, perform the following steps :-

- If X accepts w, then simulate X_2 on X.
- Run Z on input $\langle w\rangle$.
- If Z accepts $\langle w\rangle$, Reject it; and if Z rejects $\langle w\rangle$ accept it.

if X accepts w, then

$L(w) = L(X_2)$ and $\langle w\rangle \notin P$

if M does not accept w, then

$L(w) = L(X_1) = \psi$ and $\langle w\rangle \in P$

Here the contradiction arises, Hence it is undecidable.

Scanned by CamScanner

Tractable and Intractable problems

- The class of solvable problems is known as decidable problems.

- That means decidable problems can be solved in measurable amount of time or space.

- The tractable problems are the class of problems that can be solved within reasonable time and space.

- The intractable problems are the class of problems that can be solved within polynomial time.

This has lead to two classes of solving problems - P and NP class problems.

TRACTABLE AND POSSIBLY INTRACTABLE PROBLEMS P and NP

- There are Two group of problems

- The First group consists of the problems that can be solved in polynomial time.

ex: - Searching of an element from the list $O(\log n)$

- Sorting of elements $O(\log n)$

- The second group consists of problems that can be solved in non deterministic polynomial time.

ex: knapsack problem $O(2^{n/2})$.

Trabelling Salesperson problem $(O(n^2 2^n))$

- Any problem for which answer is either Yes (or) no is called as decision problem.

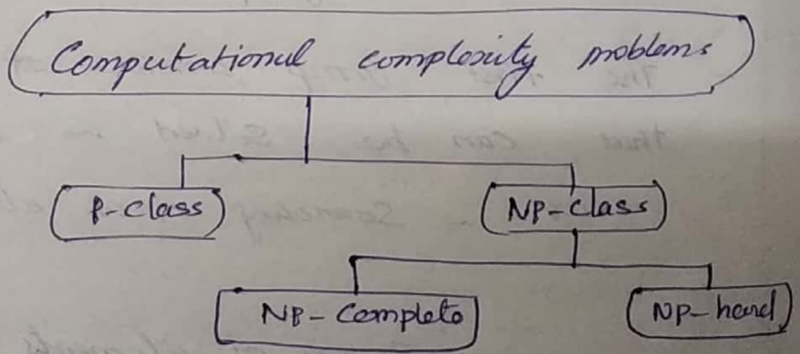
- Any problem that involves the identification of optimal cost (minimum or maximum) is called Optimization problem.

- P – problem that can be solved in polynominal time.

  * searching of key element, Sorting of elements *, All pair shortest path.

- NP – It stands for non-deterministic polynominal time".

  * Travelling Salesperson problem.
  * Graph coloring problem
  * knapsack problem.



Example of P class problem

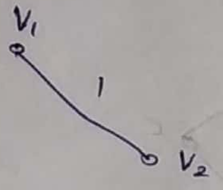
- Kruskal's Algorithm :

In Kruskal's algorithm the minimum weight is obtained. Each time the edge of minimum weight has to be selected, from the graph. It is not necessary in this algorithm to have edges of minimum weight

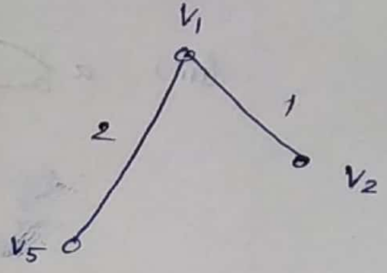
Scanned by CamScanner

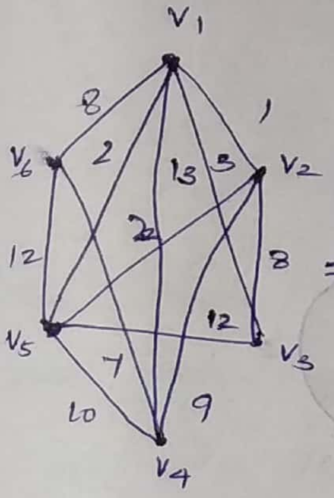
to be adjacent.

ex: (1)

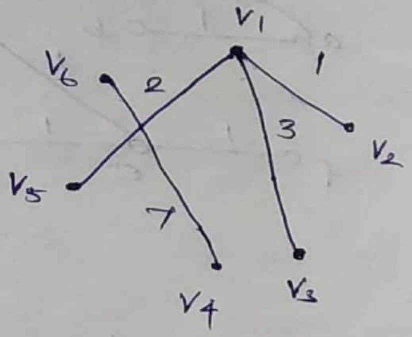


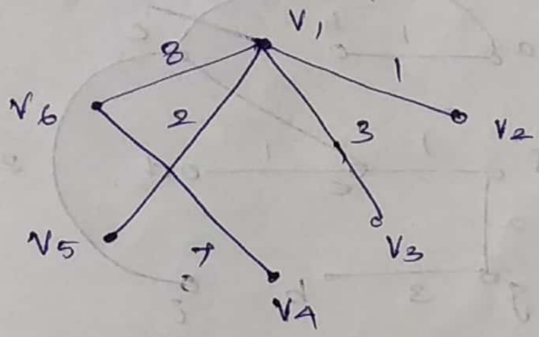
(2)





⇒ (3)

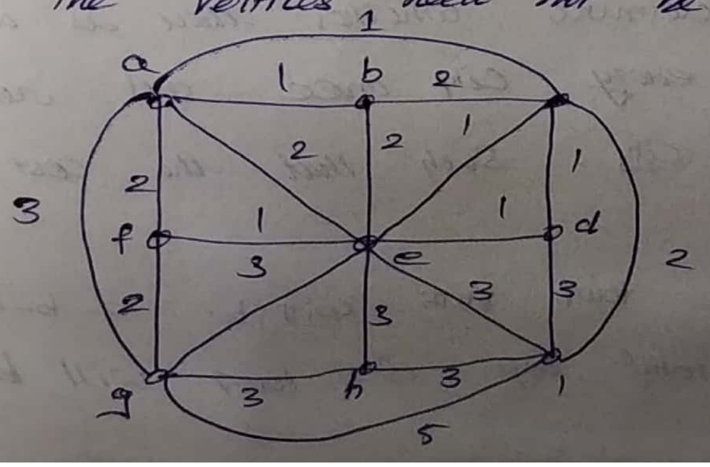




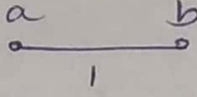
weight = 21

ex: 2 Find the minimum spanning tree to the following

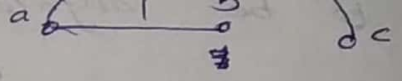
In kruskal's algoritm, we will start with some Vertex and will cover all the vertics with minimum weight. The vertices need not be adjacent.



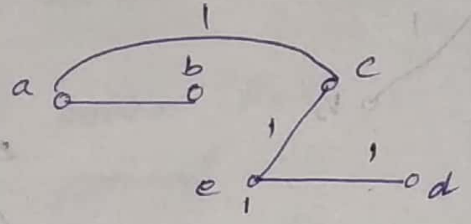
(i)



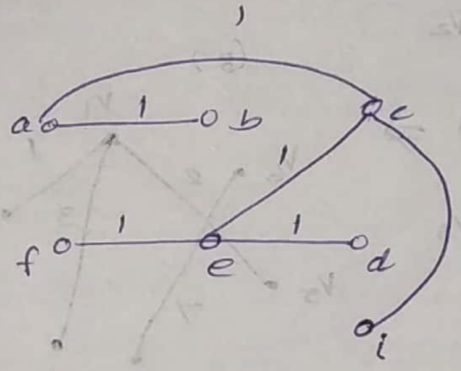
(ii)



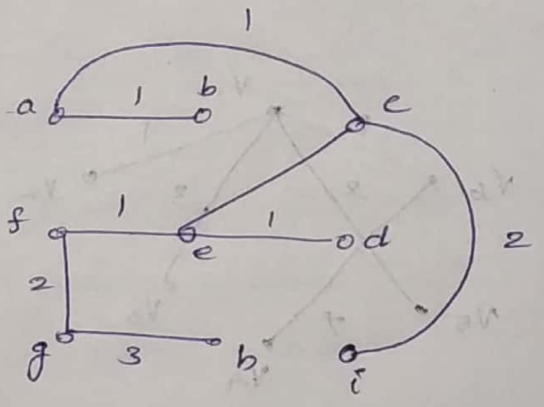
(iii)



(iv)



(v)

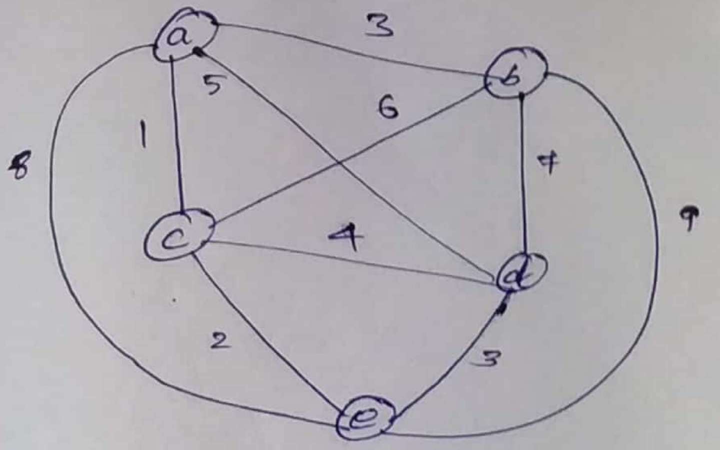


NP class problem

Travelling Salesman's problem (TSP): This problem can be stated as " Given a set of cities and cost to traval between each pair of cities.

determine whether there is a path that Visits every city once and return to the First city. Such that the cost travelled is less,

The tour path will be a-b-d-e-c-a and total cost of tour will be 16,



This problem is NP problem as there may exist some path with shortest distance b/w the cities.

If you get the solution by applying certain algorithm then travelling salesman problem is NP complete problem.

If we get not solution at all by applying an algorithm then the travelling Salesman problem belongs to NP hard class.

Scanned by CamScanner