CS8501 THEORY OF COMPUTATION

UNIT- I

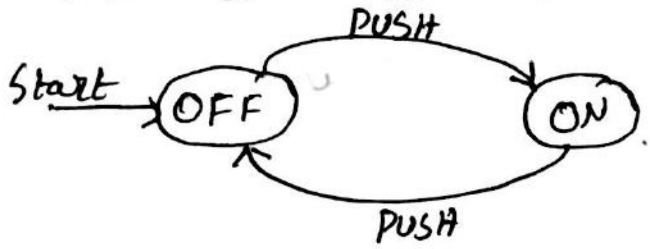
AUTOMATA FUNDAMENTALS

Introduction to formal proofs - Additional forms of proof - Inductive proofs - Finite Automata - Deterministic Finite Automata - Non-deterministic Finite Automata - Finite Automata with Epsilon Trsitions.

I Introduction:

Finite Automata is a mathematical model that accepts a set of inputs, process through a set of states, generates on outputs.

Example: Finite State System - switch operation:



II BASIC MATHEMATICAL NOTATION AND TECHNIQUES:

1. Basic Mathematical objects:

SETS [A]:

A set is collection of elements of finite number.

Example:

$A = \{10, 01, 00, 11\}$

$B = \{ w / w$ is a set of prime numbers less than $100 \}$

$\Rightarrow w \in B$

SUBSET [$\subseteq$]

Let A_1 and A_2 are two sets, then A_1 is a subset of A_2, If every element of A_1 is in A_2.

Example:

$A_1 = \{1, 2, 3, 4\}$ $A_2 = \{1, 2, 3, 4, 5\}$

$\Rightarrow A_1 \subseteq A_2$

COMPLEMENT OF A SET [A']:

Let A be a set of finite number of elements Then A' is a set of elements that are not the elements of set A [$A' \neq A$].

Example:

$A_1 = \{a, e, i, o, u\}$

$A_1' = \{$all alphabets except vowels$\}$

OPERATIONS ON SETS:

Union [$\cup$]:

The union of two sets results in a set containing the elements of both the sets.

Example:

$A_1 = \{1, 3, 5, 7\}$ $A_2 = \{1, 2, 3, 4\}$

$A_1 \cup A_2 = \{1, 2, 3, 4, 5, 7\}$

Intersection [$\cap$]:

The intersection of two sets containing a set of elements that are available in both the sets.

Example:

$A_1 = \{1, 3, 5, 7\}$ $A_2 = \{1, 2, 3, 4\}$

$A_1 \cap A_2 = \{1, 3\}$

LAWS ON SETS:

Commutative law:

$$A \cup B = B \cup A$$
$$A \cap B = B \cap A$$

Associative law:

$$A \cup (B \cup C) = (A \cup B) \cup C$$
$$A \cap (B \cap C) = (A \cap B) \cap C$$

Distributive law:

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$
$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

RELATIONSHIP OF SETS:

Relation of two sets is the association of one set with other.

Reflexive: Every elements of the set is associated with itself.

Symmetric: Let $a, b \in A$, then a is related to b, and b is related to a [$aRb = bRa$]

Transitive: If $a \to b$ and $b \to c$, then $a \to c$. It is denoted as: aRb, bRc then aRc.

2. PROOFS:

A proof is single line / multiline derivation that provide a convincing arguments to make a statement true.

Forms of proofs:

There are basically two forms of proofs

1) Deductive proofs

2) Inductive proofs

Deductive Proofs:

These are sequence of Statements which are derived from any assumption [hypothesis] or a given initial Statement to a conclusion statement.

Example: If $x \geq 4$, $2^x \geq x^2$

The hypothesis statement $\Rightarrow x \geq 4$

Conclusion statement $\Rightarrow 2^x \geq x^2$

This can be provided as subsition statement as:

when $x = 4$: $2^4 \geq 4^2 \Rightarrow 16 \geq 16 \Rightarrow$ True

when $x = 5$: $2^5 \geq 5^2 \Rightarrow 32 \geq 25 \Rightarrow$ True

when $x = 3$: $2^3 \geq 3^2 \Rightarrow 8 \geq 9 \Rightarrow$ False

Hence proved.

Inductive Proof:

It has a sequence of recursive / Parametrical Statements that handle the Statements with lower value of its Parameters.

There are two types of inductions,

* Mathematical Induction
* Structural Induction

Mathematical Induction:

This follows induction principle that follows two steps:

* Basis of Induction: we start with lowest possible value.
  Ex: To prove f(n). we take n=0 or 1 initially.
* Inductive Step: Here we prove if f(n) is true, f(n+1) is also true.

Structural Induction:

Structural induction follows the mathematical induction concept but applies for trees and expressions.

Additional Forms of Proofs:

1. Proofs about sets
2. Proofs about contradiction
3. Proofs by counter example
4. Direct proofs.

Sample Problems : Inductive Proofs:

1) Prove that : $1+2+3+\cdots+n = (n+1)/2$ using method of induction for $(n>0)$.

Proof:

Basis of Induction:

let $n=1$

$L.H.S = 1$ —— (1)

$R.H.S = \frac{1+1}{2}$

$= 2/2$

$= 1$ —— (2)

since $L.H.S = R.H.S$

(1) = (2)

Inductive step:

we have $1+2+3+\cdots+n = \frac{n(n+1)}{2}$

let $n = n+1$

$L.H.S = \frac{n(n+1)}{2} + (n+1)$

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

$$= \frac{n^2+n+2n+2}{2}$$

$$= \frac{n^2+3n+2}{2} \longrightarrow (1)$$

R.H.S $= \frac{(n+1)((n+1)+1)}{2}$

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

$$= \frac{n^2+2n+n+2}{2}$$

$$= \frac{n^2+3n+2}{2} \longrightarrow (2)$$

from (1) & (2) $\Rightarrow$ $\boxed{L.H.S = R.H.S}$

The hypothesis is proved.

—x—

2) For all $n \geq 0$ $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

Proof:

Let $n=1$

$$L.H.S = \sum_{i=1}^{1} i^2 = 1^2 = 1$$

$$R.H.S = \frac{1(1+1)(2+1)}{6} = \frac{1(2)(3)}{6}$$

$$= 6/6 = 1$$

L.H.S = R.H.S $\Rightarrow$ proved.

Inductive step:

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

L.H.S : Sub $n = n+1$

$$= \sum_{i=1}^{n+1} i^2$$

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

$$= \frac{n(n+1)(2n+1)}{6} + (n+1)^2$$

$$= \frac{n(n+1)(2n+1) + 6(n+1)^2}{6}$$

$$= \frac{n(2n^2+n+2n+1) + 6(n^2+2n+1)}{6}$$

$$= \frac{2n^3+n^2+2n^2+n+6n^2+12n+6}{6}$$

$$= \frac{2n^3+9n^2+13n+6}{6} \longrightarrow (1)$$

R.H.S : Sub $n = n+1$

$$= \frac{(n+1)((n+1)+1)(2(n+1)+1)}{6}$$

$$= \frac{(n+1)(n+2)(2n+2+1)}{6}$$

$$= \frac{(n^2+2n+n+2)(2n+3)}{6}$$

$$= \frac{2n^3+3n^2+4n^2+6n+2n^2+3n+4n+6}{6}$$

$$= \frac{2n^3+9n^2+13n+6}{6} \longrightarrow (2)$$

from (1) & (2) $\boxed{L.H.S = R.H.S}$

The hypothesis is proved.

III FINITE AUTOMATA:

Basic Definitions:

1) Alphabet (Σ):

An alphabet is a finite, non empty set of symbols

Ex: $\Sigma = \{0,1\}$ → Binary alphabet

$\Sigma = \{a,b,c, \dots z\}$ → lower case letters set

2) String (ω):

A string is a finite sequence of symbols chosen from some alphabet.

Ex:

01101 is a string from $\Sigma = \{0,1\}$

aa, bb, ab, ba are strings from $\Sigma = \{a,b\}$

3) Empty string (ε) | NULL string (λ | Λ):

An empty string is the string with zero occurrences of symbols.

4) Reverse string (ωR):

The reverse of the string is obtained by writing the string in reverse order.

Ex: $\omega = \{abc\}$ $\quad \omega R = \{cba\}$

5) Kleene closure (Σ^*):

Let Σ be an alphabet. Then the kleene closure, Σ^* denotes the set of all strings over the alphabet, Σ

$$\boxed{\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots}$$

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

$\Sigma^* = \{\varepsilon, a, b, aa, bb, ab, ba, aaa, \dots\}$

6) Positive closure / Kleene plus (Σ^+):

Let Σ be an alphabet. Then the positive closure, Σ^+ denotes the set of all strings over the alphabet Σ except null string (ϵ).

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

Ex:

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

$\Sigma^+ = \{1, 0, 11, 00, 01, 10, 111, 000, \ldots\}$

7) Palindrome:

A palindrome is a string which is same when read in backward or forward direction.

$$\boxed{w = w^R \Rightarrow w \text{ is a palindrome}}$$

Ex: $w = \{1001\}$

$\therefore$ $w^R = \{1001\}$ equal $\Rightarrow$ w is a Palindrome

8) language (L):

Alphabet $\rightarrow$ finite set of symbols

strings $\rightarrow$ collection of alphabets

language $\rightarrow$ collection of appropriate strings.

A set of strings taken from an alphabet is called a language

Ex: 1) $\Sigma = \{a, b\}$

$L = \{a^n b \; n \geq 0\} \Rightarrow \{b, ab, aab, aaab, \ldots\}$

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

$L = \{$set of strings ending with 11$\}$

$\Rightarrow \{11, 011, 0011, 1011, 0111, \ldots\}$

Definition of FINITE AUTOMATA:

A finite Automata is a quintuple (5 tuple)

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

where,

Q - Finite set of States

Σ - Finite set of symbols called i/p alphabet

$\delta : Q \times \Sigma \rightarrow Q$ - Transition function

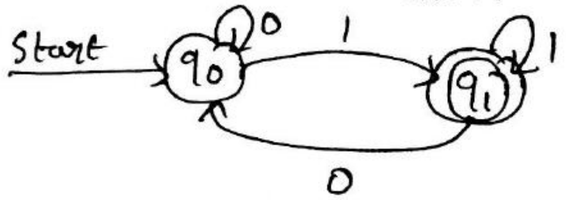
$q_0 \in Q$ - Initial state

$F \subseteq Q$ - set of final state

Transition Diagram:

A transition diagram is a directed graph associated with the vertices of the graph corresponding to the state of finite automata.

Ex:



Transition Table:

A transition table is a conventional, tabular representation of function like δ, that takes 2 arguments and returns a value.

Ex:

	δ	0	1
$\rightarrow$	q_0	q_2	q_0
	q_1	q_1	q_1
*	q_2	q_2	q_1

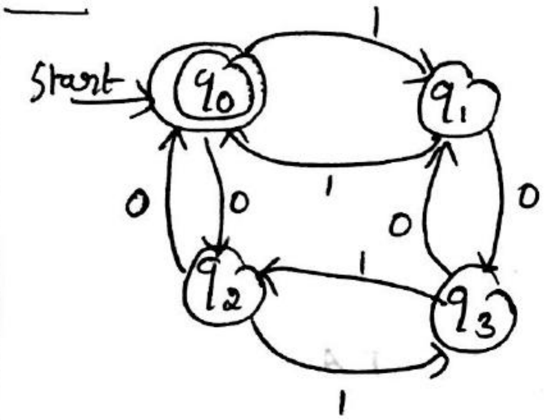
Language Acceptance by FA's

A string 'x' is accepted by finite automata $M = (Q, \Sigma, \delta, q_0, F)$ only if $\delta(q_0, x) = p$ for some p in F

The language accepted by M which is denoted by $L(M)$.

$$\boxed{L(M) = \{x \mid \delta(q_0, x) \text{ is in } F\}}$$

Ex 1:



Check whether the i/p string 110101 is accepted by FA or not.

soln:

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

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

δ:

state | i/p	0	1
$\rightarrow * q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

$\delta[q_0, 110101) = \delta(\delta(q_0, 1), 10101)$

$= \delta(\delta(q_1, 1), 0101)$

$= \delta(\delta(q_0, 0), 101)$

$= \delta(\delta(q_2, 1), 01)$

$= \delta(\delta(q_3, 0), 1)$

$= \delta(q_1, 1)$

$= q_0 \implies$ accepted state.

$\therefore$ 110101 is in $L(M)$

Types of Finite Automata (FA):

There are two types of FA

1. Deterministic Finite Automata (DFA)
2. Non Deterministic Finite Automata (NFA or NDFA)

Deterministic Finite Automata (DFA):

The term deterministic refers to the fact that on each i/p there is one and only state to which the automaton can have transition from its current state.

Definition:

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

Where,

Q - Finite set of states

Σ - Finite set of symbols

$\delta: Q \times \Sigma \to Q$ - Transition function

$q_0 \in Q$ - Initial state

$F \subseteq Q$ - set of final state

Examples:

1) Design a DFA for accepting all strings of
$L = \{0^m 1^n \mid m \geq 0 \text{ and } n \geq 1\}$.

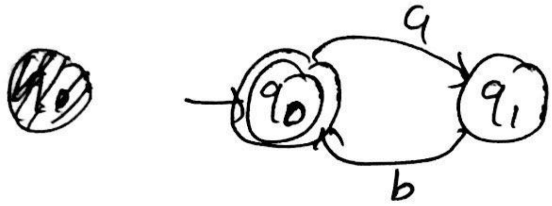
Soln:

DFA:

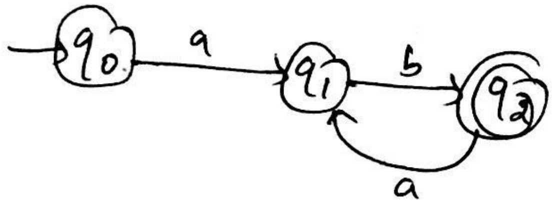
2) Design DFA over $\Sigma = \{a, b\}$ for
i) $(ab)^n$ with $n \geq 0$
ii) $(ab)^n$ with $n \geq 1$

Soln:

i)



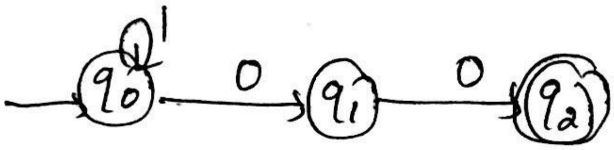
ii)



3) Give DFA's accepting the following languages over $\Sigma = \{0, 1\}$

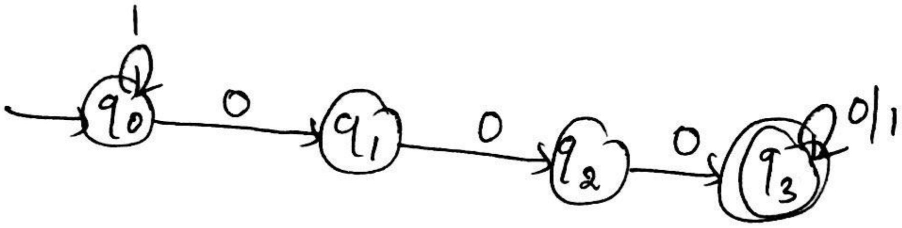
a) The set of all strings ending in 00

Soln:



b) The set of all strings with 3 consective 0's (not necessarily at the end)

Soln:



Non Deterministic Finite Automata (NFA / NDFA):

A NFA has power to be in several states at once.

Definition:

A NFA is defined by a 5 tuple

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

where,

Q - Finite set of states

Σ - Finite set of symbols

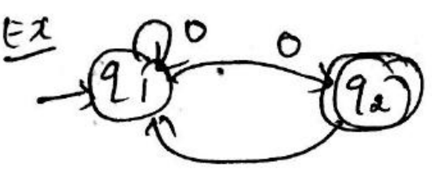
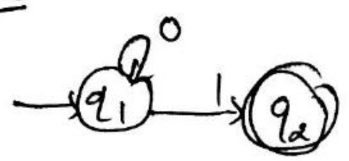
$q_0 \in Q$ - Start state

$F \subseteq Q$ - Finite of final state

δ :- transition function mapping

$Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$

Difference b/w NFA & DFA:

NFA	DFA
* δ is a transition function that takes a state and i/p symbol as arguments but returns a zero, one or more state. $Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$	* δ is a transition function that takes a state and i/p as arguments but returns exactly one state. $Q \times \Sigma \rightarrow Q$
* Ex  $\delta(q_1, 0) = \{q_1, q_2\}$	* Ex  $\delta(q_1, 0) = \{q_2\}$

Extended Transition function [$\hat{\delta}$]:

This is used to represent transition functions with a string of i/p symbol 'w' and returns a set of state. It is represented by $\hat{\delta}$. suppose $w = xa$

$$\delta[q, x] = \{p_1, p_2, \dots p_k\}$$

Then

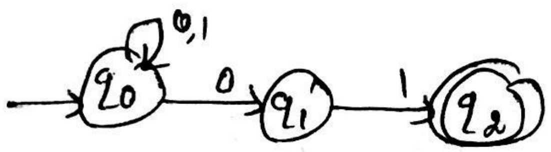
$$\bigcup_{i=1}^{k} \hat{\delta}(p_i, a) = \{r_1, r_2, \dots r_m\}$$

$$\therefore \hat{\delta}[q, w] = \{r_1, r_2, \dots r_m\}$$

(Or)

$$\hat{\delta}(q, w) = \hat{\delta}(\delta(q, x), a)$$

Ex:



Process the i/p 001.

Soln:

$$\hat{\delta}(q_0, 001) = \hat{\delta}(\delta(q_0, 0), 01)$$

$$= \hat{\delta}(\delta(\{q_0, q_1\}), 01)$$

$$= \delta((\delta(q_0, 0) \cup \delta(q_1, 0)), 1)$$

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

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

$$= \{q_0\} \cup \{q_2\}$$

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

Language of an NFA:

If $A = (Q, \Sigma, \delta, q_0, F)$ is an NFA then

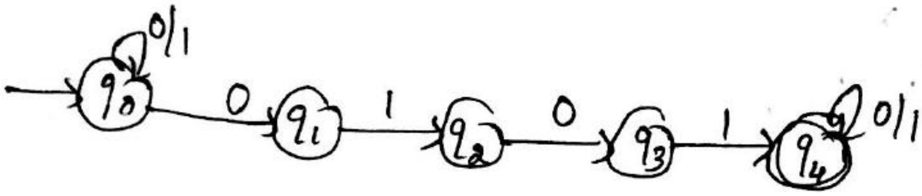
$$\boxed{L(A) = \{ w \mid \hat{\delta}(q_0, w) \cap F \neq \phi \}}$$

$L(A)$ is the set of strings w in Σ^* such that $\hat{\delta}(q_0, w)$ contains atleast 1 accepting state.

Problems:

1) Design a NFA to accept strings containing the substring "0101".

Soln:



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

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

$\Sigma = \{0, 1\}$, δ:

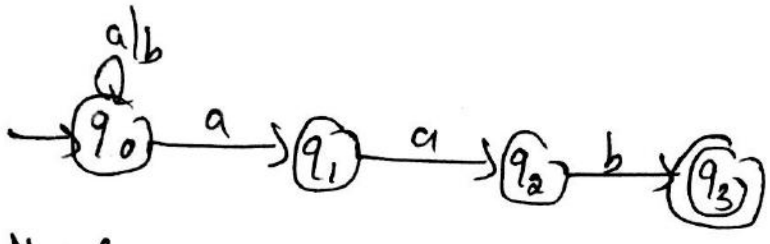
$q_0 = \{q_0\}$

$F = \{q_4\}$

Q / Σ	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	ϕ	$\{q_2\}$
q_2	$\{q_3\}$	ϕ
q_3	ϕ	$\{q_4\}$
$*q_4$	$\{q_4\}$	$\{q_4\}$

2) Construct a NFA that accepts $L = \{x \in \{a,b\} \mid x$ ends with 'aab$\}$

Soln:



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

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

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

$q_0 = \{q_0\}$

$F = \{q_3\}$

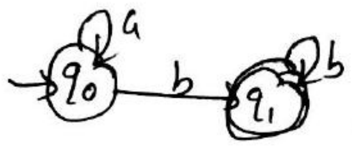
δ is

Q / Σ	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	ϕ
q_2	ϕ	$\{q_3\}$
$*q_3$	ϕ	ϕ

—x—

3) Design a NFA for $L = \{x \in \{a,b\}^* \mid x$ contains any number of a's followed by atleast one b$\}$

Soln:



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

$Q = \{q_0, q_1\}$

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

$q_0 = \{q_0\}$

$F = \{q_1\}$

δ is:

Q / Σ	a	b
$\rightarrow q_0$	$\{q_0\}$	$\{q_1\}$
$*q_1$	ϕ	$\{q_1\}$

Equivalence of NFA and DFA:

Theorem: Let L be a set accepted by NFA. Then there exists a DFA that accept L.

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 states of M' are all the subset of M'. The $Q' = 2^Q$. F' be the set of all the final states in M.

The elements in Q' will be denoted by $[q_1, q_2, \ldots q_i]$ and the elements in Q are denoted by $\{q_1, q_2, \ldots q_n\}$. The $[q_1, q_2, \ldots q_i]$ will be assumed as one state in Q' if in the NFA q_0 is initial state it is denoted in DFA as $q_0' = [q_0]$. we define,

$$\delta'([q_1, q_2, \ldots q_i], a) = [p_1, p_2, \ldots p_j]$$

if and only if,

$$\delta(\{q_1, q_2, \ldots q_i\}, a) = \{p_1, p_2, \ldots p_j\}$$

This means that whenever in NFA, at the current states $\{q_1, q_2, \ldots q_i\}$ if we get i/p 'a' and it goes to the next states $\{p_1, p_2, \ldots p_j\}$ then while constructing DFA for it the current state is assumed to be $[q_1, q_2, \ldots q_i]$ at this state, the i/p is 'a' and the next is assumed to be $[p_1, p_2, \ldots p_j]$.

The theorem can be proved 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 (a) $|x| = 0$, that means x is ε then $q_0' = [q_0]$.

Induction: If we assume that the hypothesis is true for the string of length m or less than m. Then if x is a string of length $m+1$. Then function δ' could be written as

$$\delta'(q_0', xa) = \delta'(\delta'(q_0, x), a)$$

By the induction hypothesis,

$$\delta'(q_0', x) = [p_1, p_2, \cdots p_j]$$

if and only if,

$$\delta(q_0, x) = \{p_1, p_2, \cdots p_j\}$$

By defn. of δ'

$$\delta'([p_1, p_2, \cdots p_j], a) = [r_1, r_2, \cdots r_k]$$

if and only if,

$$\delta(\{p_1, p_2, \cdots p_j\}, a) = \{r_1, r_2, \cdots r_k\}$$

Thus

$$\delta'(q_0', xa) = [r_1, r_2, \cdots r_k]$$

If and only if

$$\delta(q_0, xa) = \{r_1, r_2, \cdots r_k\}$$

is shown by inductive hypothesis.

Thus $\boxed{L(M) = L(M')}$. Hence proved.

Ex:

1) 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.

Soln:

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

Q': $Q' = 2^Q$ states $\Rightarrow 2^2 \Rightarrow 4$

$Q' = \{[q_0], [q_1], [q_0 q_1], \phi\}$, $\Sigma = \{0,1\}$

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

δ':

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

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

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

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

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

$= [q_0 q_1] \cup \phi$

$= [q_0 q_1]$

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

$= [q_1] \cup [q_0 q_1]$

$= [q_0 q_1]$

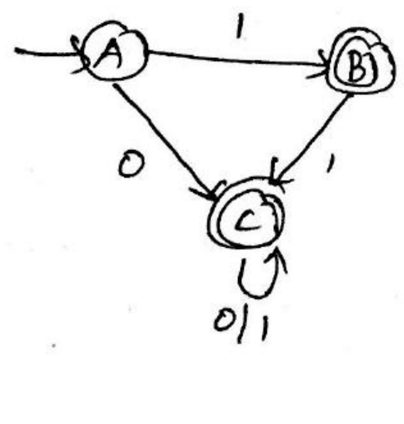
$\therefore A = q_0$, $B = q_1$, $C = q_0 q_1$

Table:

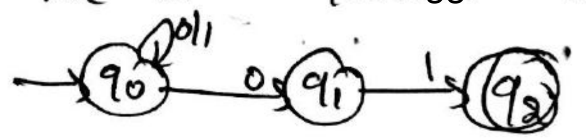
δ	0	1
→A	C	B
*B	ϕ	C
*C	C	C

δ	0	1
→A	C	B
*B	A	C
*C	C	C

Transition diagram:



2) Obtain the DFA equivalent to the following NFA



Soln:

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

$Q' = \{[q_0], [q_1], [q_2], [q_0 q_1], [q_0 q_2], [q_1 q_2], [q_0 q_1 q_2], \phi\}$

$\Sigma = \{0, 1\}$, $q_0' = [q_0]$, $F' = \{[q_2], [q_0 q_2], [q_1 q_2], [q_0 q_1 q_2]\}$

δ':

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

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

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

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

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

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

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

$\delta'([q_0 q_1], 1) = \delta(q_0, 1) \cup \delta(q_1, 1)$
$= [q_0] \cup [q_2]$
$= [q_0 q_2]$

$\delta'([q_0 q_2], 0) = \delta(q_0, 0) \cup \delta(q_2, 0)$
$= [q_0 q_1] \cup \phi$
$= [q_0 q_1]$

$\delta'([q_0 q_2], 1) = \delta(q_0, 1) \cup \delta(q_2, 1)$
$= [q_0] \cup \phi$
$= [q_0]$

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

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

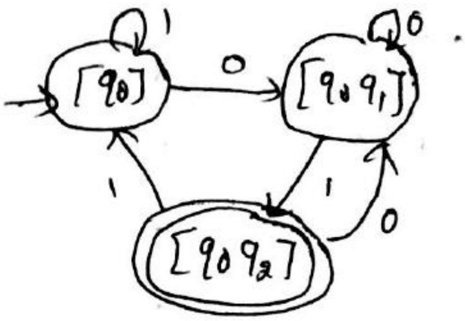
$\delta'([q_0 q_1 q_2], 0) = \delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)$
$= [q_0 q_1] \cup \phi \cup \phi$
$= [q_0 q_1]$

$\delta'([q_0 q_1 q_2], 1) = \delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)$
$= [q_0] \cup [q_2] \cup \phi$
$= [q_0 q_2]$

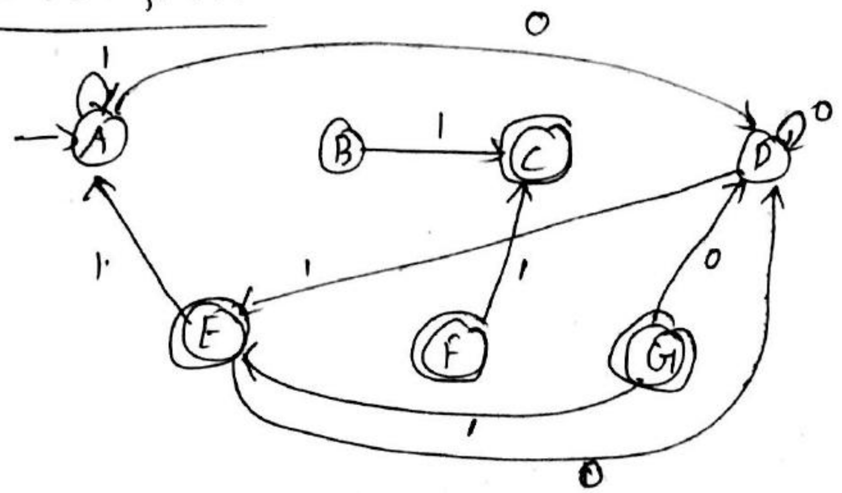
Transition Table:

δ	0	1
→[q_0] A	[$q_0 q_1$] D	[q_0] A
[q_1] B	φ	[q_2] C
* [q_2] C	φ	φ
[$q_0 q_1$] D	[$q_0 q_1$] D	[$q_0 q_2$] E
* [$q_0 q_2$] E	[$q_0 q_1$] D	[q_0] A
* [$q_1 q_2$] F	φ	[q_2] C
* [$q_0 q_1 q_2$] G	[$q_0 q_1$] D	[$q_0 q_2$] E

δ	0	1
→[q_0]	[$q_0 q_1$]	[q_0]
[$q_0 q_1$]	[$q_0 q_1$]	[$q_0 q_2$]
*[$q_0 q_2$]	[$q_0 q_1$]	[q_0]

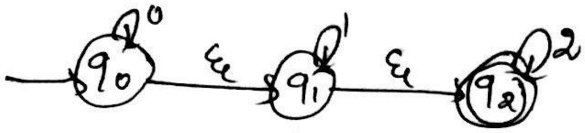


Transition diagram:



FINITE AUTOMATA WITH ϵ MOVES:

It is possible in NFA that an NFA is allowed to make transition spontaneously, without receiving an i/p symbol. This move is called ϵ-moves. This ϵ represents "any number of times".



States	input			
	ϵ	0	1	2
$\rightarrow q_0$	q_1	q_0	ϕ	ϕ
q_1	q_2	ϕ	q_1	ϕ
$* q_2$	ϕ	ϕ	ϕ	q_2

EPslion(ϵ) closure:

It state p is in ϵ-closure(q), and there is a transition from state p to state r labled ϵ, then r is in ϵ-closure(q). More precisely, If δ is the function of the ϵ-NFA involved, and p is in ϵ closure(q), then ϵ-closure(q) also contains all the states in $\delta(p,\epsilon)$.

Naturally let ϵ-closure, where P is a set of states, then

$$\boxed{\bigcup_{q \text{ in } P} \epsilon\text{-closure}(q)}$$

Ex: Find ϵ-closure

Find $\hat{\delta}(q_0, 01)$.

Soln:

ϵ-closure $(q_0) = \{q_0, q_1, q_2\}$

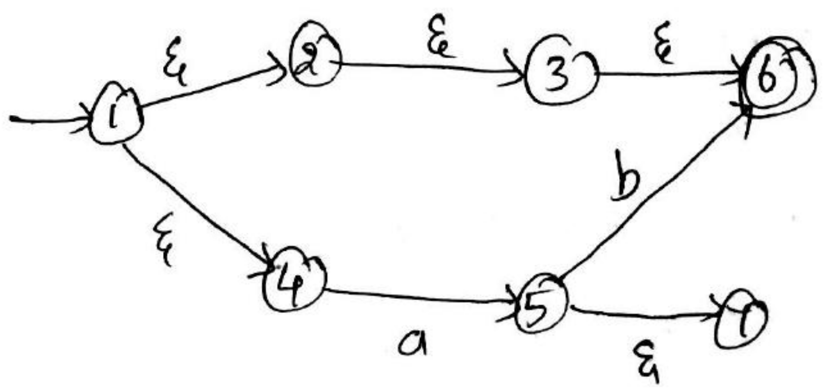
ϵ-closure $(q_1) = \{q_1, q_2\}$

ϵ-closure $(q_2) = \{q_2\}$

$$\hat{\delta}(q_0, 01) = \epsilon\text{-closure}(\delta(\hat{\delta}(q_0, 0), 1))$$
$$= \epsilon\text{-closure}(\delta(\{q_0, q_1, q_2\}, 0), 1)$$
$$= \epsilon\text{-closure}(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)), 1)$$
$$= \epsilon\text{-closure}(\delta(\{q_0 \cup \phi \cup \phi\}), 1)$$
$$= \epsilon\text{-closure}(\delta(q_0, 1))$$
$$= \epsilon\text{-closure}(\phi)$$
$$= \phi$$

— x —

Consider the NFA given below and find $\delta(1, ab)$



Language of ϵ-NFA:

The language of an ϵ-NFA, $M = (Q, \Sigma, \delta, q_0, F)$ is $\boxed{L(M) = \{ w \mid \hat{\delta}(q_0, w) \cap F \neq \phi \}}$

(ie) the language of M is the set of strings w that take the start state to atleast one accepting stat

Equivalence of NFA's with and without ϵ-Moves:

Theorem:

If L is accepted by NFA with ϵ-transitions, then L is accepted by an NFA without ϵ-transitions.

Proofs:

Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA with ϵ-transitions construct M' which is NFA without ϵ-transitions

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

where

$$F' = \begin{cases} F \cup \{q_0\} & \text{if } \epsilon\text{-closure}(q_0) \text{ contains a State of } F \\ F & \text{otherwise} \end{cases}$$

By induction: δ' and $\hat{\delta}$ are same

δ' and $\hat{\delta}$ are different

Let x be any string

$$\delta'(q_0, x) = \hat{\delta}(q_0, x)$$

This statement is not true if $x = \epsilon$ because $\delta'(q_0, \epsilon) = \{q_0\}$ and $\hat{\delta}(q_0, \epsilon) = \epsilon$-closure$(q_0)$.

Basic:

$|x| = 1$ x is a symbol whose value is a

$$\delta'(q_0, a) = \hat{\delta}(q_0, a)$$

Induction:

Let $x = wa$ where a is in Σ

$$\delta'(q_0, wa) = \delta'(\delta'(q_0, w), a)$$

$$= \delta'(\hat{\delta}(q_0, w), a)$$

$$= \delta'(p, a)$$ [because by inductive hypothesis $\delta(q_0, w) = \hat{\delta}(q_0, w) = p$]

Now we must show that

$$\delta'(p, a) = \hat{\delta}(q_0, wa)$$

But

$$\delta'(p, a) = \bigcup_{p \text{ in } q} \delta'(q, a) = \bigcup_{q \text{ in } p} \hat{\delta}(q, a)$$

$$= \hat{\delta}(\hat{\delta}(q_0, w), a)$$

$$= \hat{\delta}(q_0, wa)$$

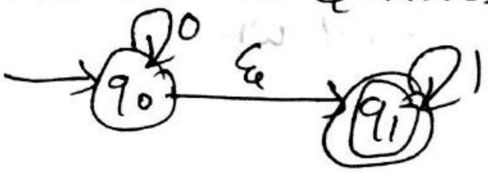
$$= \hat{\delta}(q_0, x)$$

Hence

$$\boxed{\delta'(q_0, x) = \hat{\delta}(q_0, x)}$$ Proved

Ex1:

Construct NFA without ϵ-moves from NFA with ϵ-move



Soln:

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

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

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

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

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

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

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

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

Transition Table

State	input	
	0	1
$\to * q_0$	$\{q_0, q_1\}$	$\{q_1\}$
$* q_1$	ϕ	$\{q_1\}$

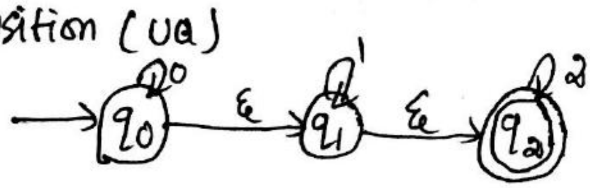
Transition diagram

— x —

2) Convert the ϵNFA to NFA (UQ)

δ	ϵ	a	b
$\to p$	$\{r\}$	$\{q\}$	$\{p, r\}$
q	ϕ	$\{p\}$	ϕ
$* r$	$\{p, q\}$	$\{r\}$	$\{p\}$

3) Obtain an NFA without ϵ-transition to the following NFA with ϵ-transition (UQ)



— x —

REGULAR EXPRESSIONS and REGULAR LANGUAGE:

The language accepted by finite automata are easily described by simple expressions called regular expressions.

OPERATIONS OF RE:

There are 3 operations on languages that the operations of RE represent.

i) UNION
ii) CONCATENATION
iii) CLOSURE

i) UNION:

The union of 2 languages L_1 and L_2 is $L_1 \cup L_2$ is the set of strings that are in either L_1 or L_2 or both.

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

Ex:

$L_1 = \{0, 11, 110\}$

$L_2 = \{\epsilon, 0, 01\}$

$L_1 \cup L_2 = \{\epsilon, 0, 110, 1101, 1100, 011, 0110, \ldots\}$

ii) CONCATENATION:

The concatenation of 2 languages L_1 and L_2 is $L_1 \cdot L_2$, which is formed by choosing a string L_1 and following it by a string in L_2, in all possible combinati

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

$L_1 = \{10, 1\}$ $L_2 = \{011, 11\}$

$L_1 \cdot L_2 = \{10011, 111, 1011\}$

iii) CLOSURE:

* Kleene closure:

The kleene closure of a language L is denoted L^* is defined as the set of strings that can be formed by taking any number of string from L, defined as,

$$\boxed{L^* = \bigcup_{i=0}^{\infty} L^i}$$

* Positive closure:

The Positive closure of L, denoted by L^+, is the set of string that can be formed by taking any number of string from L excluding ε. defined as

$$\boxed{L^+ = \bigcup_{i=1}^{\infty} L^i = L^* - L^0 = L^* - \{ε\}}$$

Ex1: $L_1 = \{10, 1\}$

$L^* = \{ε, 10, 1, 1010, 1010, 110, 11, \ldots\}$

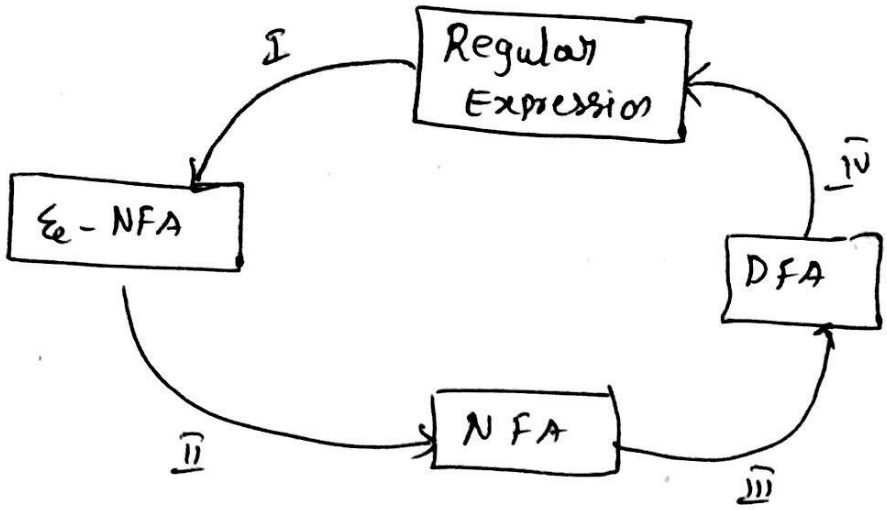
$L^+ = \{10, 1, 1010, 101, 110, 11, \ldots\}$

Ex2:

$0^* = \{ε, 0, 00, 000, \ldots\}$

$0^+ = \{0, 00, 000, \ldots\}$

FINITE AUTOMATA and REGULAR EXPRESSION:



Converting Regular Expressions to Automata:

Theorem:

Let L be a regular expression, then there exists an NFA with ϵ - transitions that accepts $L(r)$.

or

Every language defined by a regular expression is also defined by a finite automaton.

Proof:

To prove $L(M) = L(r)$ for some ϵ-NFA M.

Basis

i) Exactly one accepting state

$r = \epsilon$

ii) No arcs in to the initial state

$r = \phi$

iii) No arcs out of the accepting state

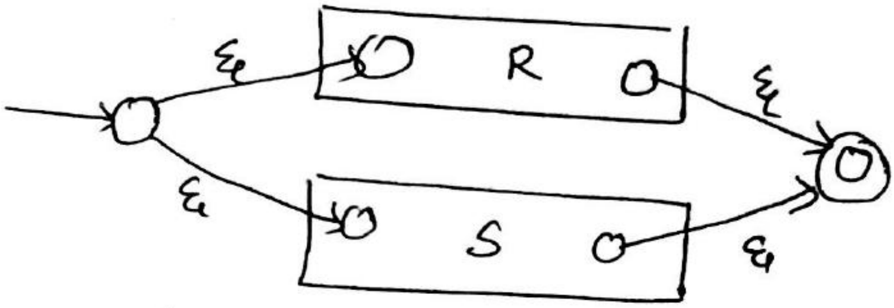
$r = a$

From the above figure it is clear that the expression r must be ε, ϕ or a for some a in ε.

Induction:

Assume that the theorem is true for the immediate sub expressions of a given regular expressions.

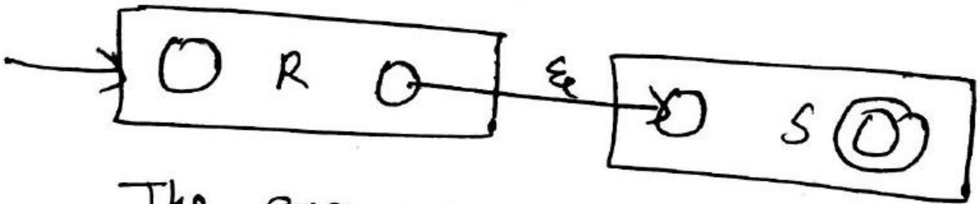
case 1:



$r = R + S$

The expression is $R+S$ for some smaller expressions R and S. Right from the start state, we can go to the start state of either the automaton for R or automaton for S. Finally reach the accepting state of automata. Thus the language of automaton is $L(R) \cup L(S)$

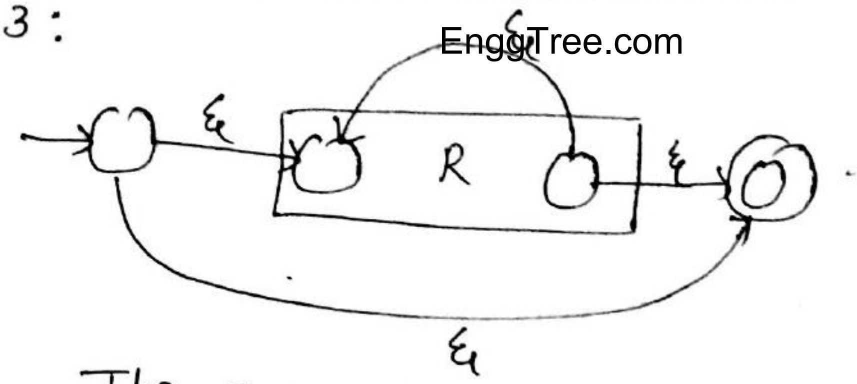
case 2:



$r = R \cdot S$

The expression RS, the start state of the first automaton becomes the start state. and the accepting state of the second automata becomes the accepting state of the whole. Thus the language of automaton is $L(R) \cdot L(S)$

Case 3 :



The expression is given by R^*, directly from the start state to accepting state along a path labled ε. That ε also belongs to R^*.

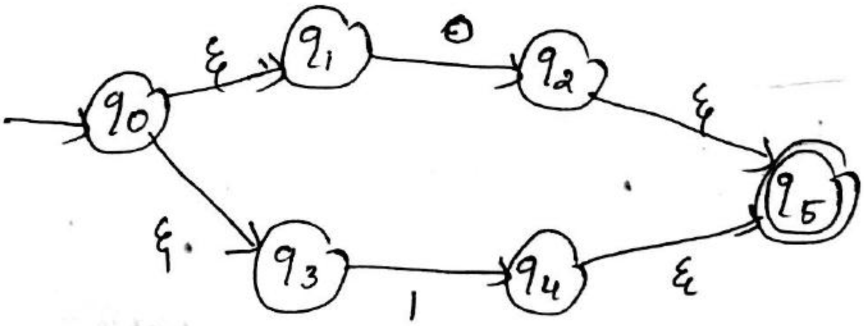
Thus the automaton satisfy the three conditions given in the inductive hypothesis - one accepting state, with no arcs in to the initial state or out of the accepting state.

—x—

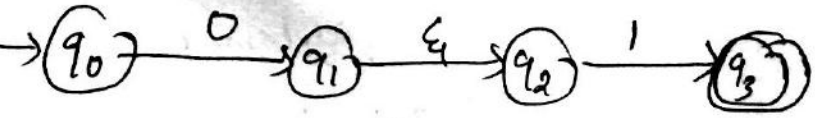
Ex:

Convert the R.E to ε NFA

1) 0+1

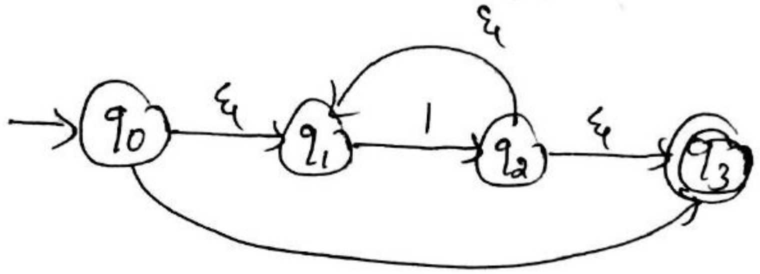


2) 01

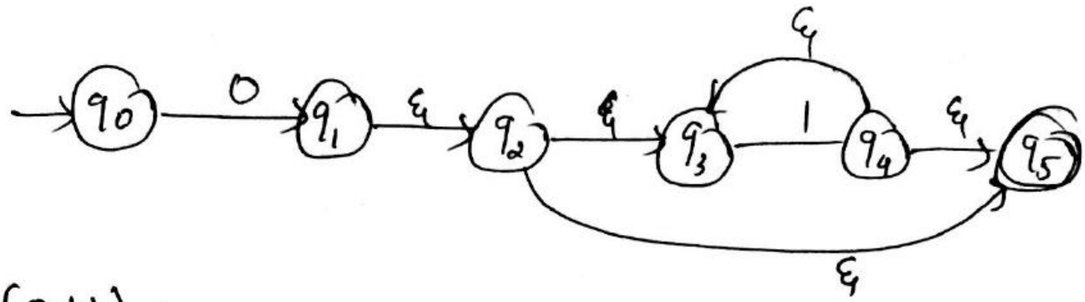


(or)

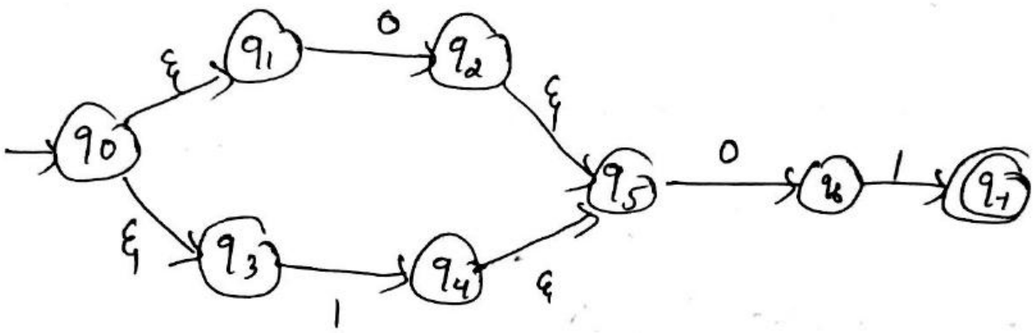
3) 1^*



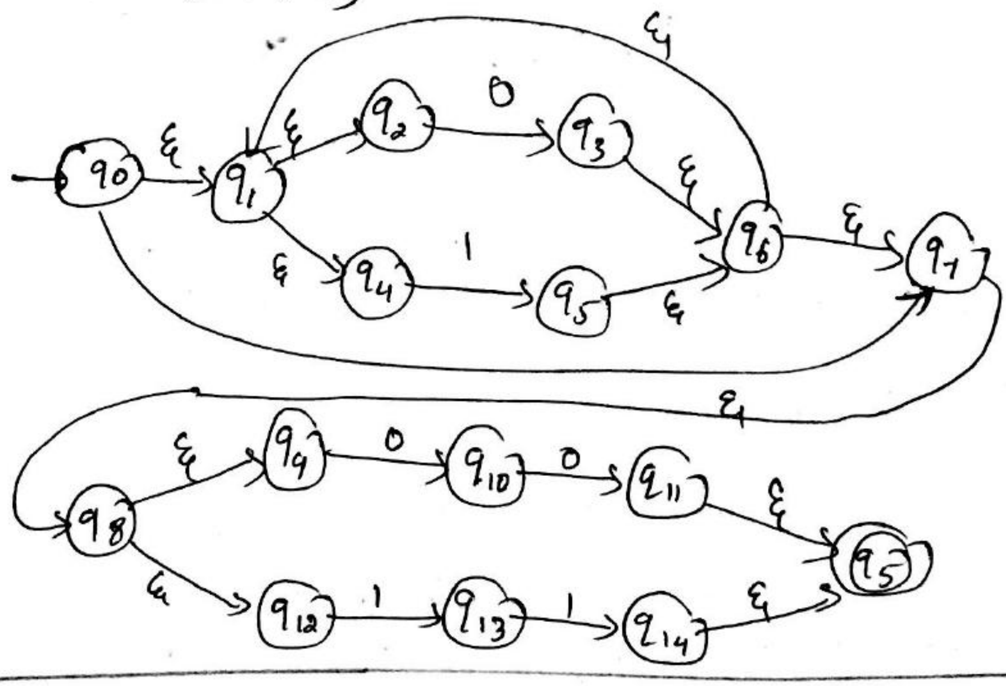
4) 01^*



5) $(0+1)\,01$



6) $(0+1)^*\,(00+11)$



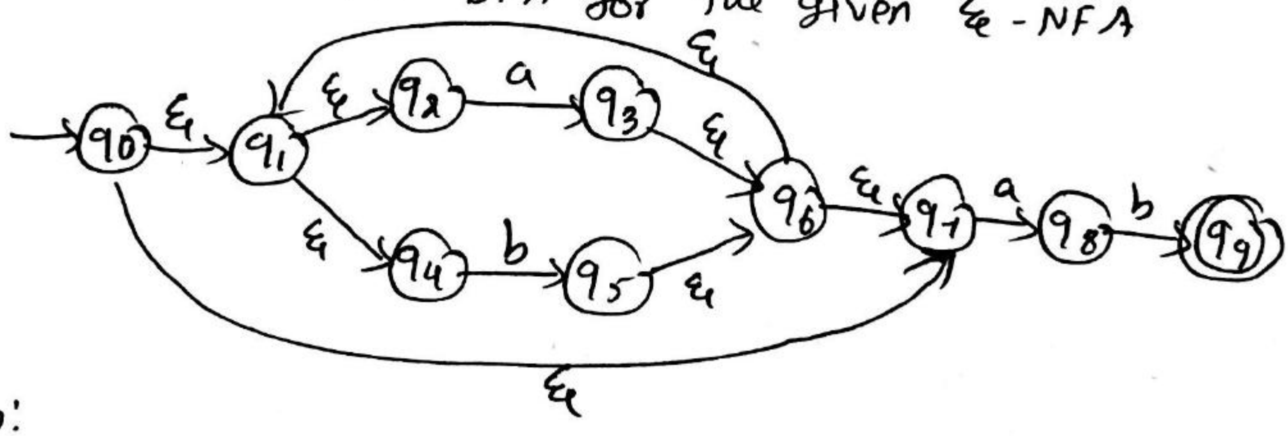
Conversion of ϵ-NFA into DFA:

Method:

1) Find the ϵ-closure of the state q_0 from the constructed ϵ-NFA from state q_0, ϵ-transition to other states are identified as well as ϵ-transitions from other states are also identified and combined as one set.

2) Perform the following steps until there are no more new states as been constructed.
   a) Find the transition of the given RE symbols over Σ from the new state.
   b) Find the ϵ-closure of Move(new state, symbol).

Ex1: Construct the DFA for the given ϵ-NFA



Soln:

ϵ-closure$(q_0) = \{q_0, q_1, q_2, q_4, q_7\} \rightarrow A$

Mov(A, a) = ϵ-closure $(\{q_3, q_8\})$

$= \epsilon$-closure$(q_3) \cup \epsilon$-closure(q_8)

$= \{q_3, q_6, q_7, q_1, q_2, q_4\} \cup \{q_8\}$

$= \{q_1, q_2, q_3, q_4, q_6, q_7, q_8\} \longrightarrow B$

$\varepsilon(Mov(A, b)) = \varepsilon\text{-closure}(q_5)$

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

$\varepsilon[Mov(B, a)] = \varepsilon\text{-closure}(q_3, q_8)$

$= \varepsilon\text{-closure}(q_3) \cup \varepsilon\text{-closure}(q_8)$

$= \{q_3, q_6, q_7, q_1, q_2, q_4\} \cup \{q_8\}$

$= \{q_1, q_2, q_3, q_4, q_6, q_7, q_8\} \rightarrow B$

$\varepsilon[Mov(B, b)] = \varepsilon\text{-closure}(q_5, q_9)$

$= \varepsilon\text{-closure}(q_5) \cup \varepsilon\text{-closure}(q_9)$

$= \{q_5, q_6, q_7, q_1, q_2, q_4\} \cup \{q_9\}$

$= \{q_1, q_2, q_4, q_5, q_6, q_7, q_9\} \rightarrow D$

$Mov(C, a) = \{q_3, q_8\}$

$\varepsilon\text{-closure}(Mov(C, a)) = \varepsilon\text{-closure}(\{q_3, q_8\})$

$= \varepsilon\text{-closure}(q_3) \cup \varepsilon\text{-closure}(q_8)$

$= \{q_1, q_2, q_3, q_4, q_6, q_7, q_8\} \rightarrow B$

$Mov(C, b) = \{q_5\}$

$\varepsilon\text{-closure}(Mov(C, b)) = \varepsilon\text{-closure}(q_5)$

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

$Mov(D, a) = \{q_3, q_8\}$

$\varepsilon\text{-closure}(Mov(D, a)) = \varepsilon\text{-closure}(q_3) \cup \varepsilon\text{-closure}(q_8)$

$= \{q_1, q_2, q_3, q_4, q_6, q_7, q_8\} \rightarrow B$

$Mov(D, b) = \{q_5\}$

$\epsilon\text{-closure}(Mov(D, b)) = \epsilon\text{-closure}(q_5)$

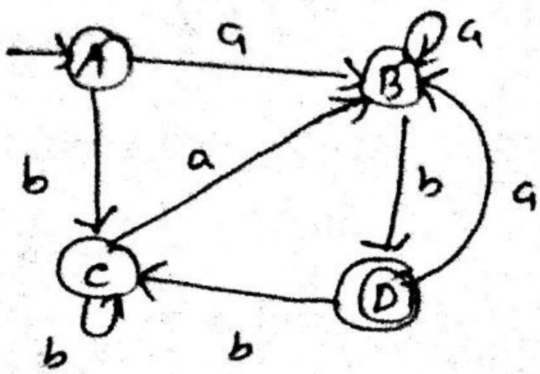
$= \{q_1, q_2, q_4, q_5, q_6, q_7\} \rightarrow C$

$F' = \{D\}$

Transition Table:

δ	input	
	a	b
$\rightarrow$A	B	C
B	B	D
C	B	C
*D	B	C

Transition diagram



—x—

1) Construct a DFA with reduced state equivalent to the regular expression $10 + (0+11)0^*$,

2) Construct a NFA for the given RE $((0+1)(0+1))^*$

3) Construct an NFA equivalent to $(0+1)^*(00+11)$

THEORY OF COMPUTATION

43.

2 MARK Questions and Answers

UNIT-I

1) What is Finite automata?

FA consist of set of states and transitions from occur on i/p symbols chosen from alphabet Σ. FA is denoted by a 5 tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is the finite set of states, Σ - finite i/p alphabet, δ - transition, q_0 - initial state, F - Final state. TYPES: NFA, DFA, ϵ NFA

2) Define Deterministic Finite Automata (DFA).

A DFA consist of finite set of states and a finite set of i/p symbols. In DFA, only a single transition occur from one state to another state. $M = (Q, \Sigma, \delta, q_0, F)$. $(\delta : Q \times \Sigma \rightarrow Q)$

3) Define Non Deterministic Finite Automata (NFA or NDFA):

A NFA consist of finite set of states and a set of i/p symbols. In NFA, zero or more transition occur from one state to another state. $M = (Q, \Sigma, \delta, q_0, F)$. $(\delta : Q \times \Sigma \rightarrow 2^Q)$.

4) What is Transition diagram and Table?

Transition diagram: FA can be represented by a directed graph namely "transition diagram". $\rightarrow q_0 \xrightarrow{a} q_1$

Transition Table: FA can be represented by a ~~directed~~ tabular representation of "δ" transition function

δ	0	1
$\rightarrow q_0$	q_0	q_0
$* q_1$	q_1	q_1

5) Define language accepted by NFA and DFA.

DFA: The language of DFA. $M = (Q, \Sigma, \delta, q_0, F)$ is denoted by $L(M)$ and it's define as $\boxed{L(M) = \{w \mid \delta'(q_0, w) \text{ is in } F\}}$

NFA: the language of DFA, $M = (Q, \Sigma, \delta, q_0, F)$ is denoted by $L(M)$ and it is define as $\boxed{L(M) = \{w \mid \delta'(q_0, w) \cap F \neq \phi\}}$

6) Define ϵ-NFA [ϵ-Transition].

It is define as, $M = (Q, \Sigma, \delta, q_0, F)$ where
Q - Set of State, Σ - Set of i/p symbol $(\Sigma \cup \{\epsilon\})$.
δ - Transition function $(Q \times \Sigma \cup \{\epsilon\} \rightarrow 2^Q)$, q_0 - Initial state
F - set of final state.

7) Define Regular Expression (RE).

RE'S are the type of language defining Notation, used to describe the regular language.

Ex: $(0+1)^* 0$

8) Define kleen closure and ~~positi~~ positive closure.

Kleen closure: Regular language L includes zero or more instance of the occurrence. $\boxed{L^* = \bigcup_{i=0}^{\infty} L^i}$

Positive closure: Regular language L includes one or more instance of the occurrence. $\boxed{L^+ = \bigcup_{i=0}^{\infty} L^i}$

9) State the pumping lemma for Regular language.

Let L be a Regular language then there exist a constant n such that w in L, $|w| \geq n$ we can break w into 3 strings: $w = xyz$

such that

i) $y \neq \epsilon$

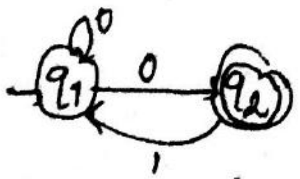
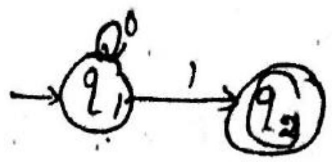
ii) $|xy| \leq n$

iii) For all $k \geq 0$, the string $xy^k z$ is also L.

10) Mention the closure properties of regular language.

- 1. Union 2. Difference 3. Concatenation 4. Intersection
  5. Complementation 6. Transpose 7. Kleene Star
  8. Homomorphism 9. Inverse homomorphism.

11) Enumerate the difference b/w NFA and DFA.

NFA	DFA
→ δ is a transition function that takes a state and i/p symbol as argument but returns a zero or more state	→ δ is a transition function that takes a state and i/p symbol as arguments but returns exactly one state
→ $M = (Q, \Sigma, \delta, q_0, F)$ Q - set of states Σ - set of i/p symbol δ - transition $Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$ q_0 - Initial State F - Final State.	→ $(M = Q, \Sigma, \delta, q_0, F)$ Q - set of states Σ - set of i/p symbol δ - transition $Q \times \Sigma \to Q$ q_0 - Initial state. F - Final State
 $\delta(q_1, 0) = \{q_1, q_2\}$	 $\delta(q_1, 0) = \{q_1\}$

— * —

1) What are the applications of finite automata?

1) Compiler construction 2) Switch circuit, 3) Pattern searching
4) To verify the correctness of a program
5) Design and analysis of complex s/w and H/w systems

12) Define Proof.

A Proof is single line/ multi line derivation that provide a convincing argument to make a statement true.

<u>Two Forms of proofs:</u> 1. Deductive proofs 2. Inductive proofs.

→ Mathematical Induction

→ Structural Induction

12) What is Deductive proof?

These are sequence of statements which are derived from any assumption (hypothesis) or a given initial statement to a conclusion statement.

Ex: If $x \geq 4$, $2^2 \geq x^2$

Hypothesis statement $\Rightarrow x \geq 4$

conclusion statement $\Rightarrow 2^2 \geq x^2$

13) What is Inductive proof?

It has a sequence of recursive/ parametrical statement that handle the statements ~~which~~ with lower ~~...~~ value of its parameters.

<u>Types:</u> 1. Mathematical Induction 2. Structual Induction

14) What is Mathematical Induction?

This follows induction principle that follows two steps:

* <u>Basis of Induction</u>: we start with lowest possible value

Ex: To prove $f(n)$. we take $n=0$ or 1 initially

* <u>Inductive step</u>: Here we prove if $f(n)$ is true, $f(n+1)$ is also true.

15). What is Structural Induction?

Structural Induction follows the mathematical induction concept but applies for trees and expressions.

16 Define Epsilon closure (ϵ-closure).

If state p is ϵ-closure(q), and there is a transition from state p to state r labled ϵ then r is in ϵ-closure (q). $\boxed{\bigcup_{q \in P} \epsilon\text{-closure}(q)}$.

UNIT - II

1) Define Grammar. (or) Context Free Grammar (CFG)

A grammar is a set of production rules for generating string in a language. $\boxed{G = (V, T, P, S)}$ where V - variable, T - Terminal, P - Production, S - starting symbol.

2) **TYPES** of Grammar.

1. TYPE 0 - Recursive Grammar, 2. TYPE 1 - Context sensitive
3. TYPE 2 - Context free grammar 4. TYPE 4 - Regular Grammar

3) Define derivation.

set of terminal strings derived from the start symbol.

Ex: $S \to 0S1 / \epsilon$

$S \Rightarrow 0S1$ [$S \to \epsilon$

$\Rightarrow 01$

TYPES:
1. Left Most Derivation (LMD)
2. Right Most Derivation (RMD)

4) Define LMD and RMD.

LMD: we replace the left Most non terminal by one of its production in the grammar it's represented by $\boxed{\underset{lm}{\Rightarrow}}$

Ex: $E \to E+E \mid E*E \mid (E) \mid a$

$E \underset{lm}{\Rightarrow} (E)$

$\underset{lm}{\Rightarrow} (E+E)$

$\underset{lm}{\Rightarrow} (a+E) \underset{lm}{\Rightarrow} (a+a)$

UNIT-II

REGULAR EXPRESSIONS AND LANGUAGES

Regular Expressions - FA and Regular Expressions - Proving languages not to be regular - closure properties of Regular languages - Equivalence and Minimization of Automata.

REGULAR EXPRESSIONS and REGULAR LANGUAGE:

The language accepted by finite automata are easily described by simple expressions called regular expressions

OPERATIONS OF RE:

There are 3 operations on languages that the operations of RE represent.

i) UNION

ii) CONCATENATION

iii) CLOSURE

i) UNION:

The union of 2 languages L_1 and L_2 is $L_1 \cup L_2$ is the set of strings that are in either L_1 or L_2 or both.

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

Ex:

$L_1 = \{0, 11, 110\}$

$L_2 = \{\varepsilon, 0, 01\}$

$L_1 \cup L_2 = \{\varepsilon, 0, 110, 1101, 1100, 011, 0110, \ldots\}$

ii) CONCATENATION:

The concatenation of 2 languages L_1 and L_2 is $L_1 \cdot L_2$, which is formed by choosing a string L_1 and following it by a string in L_2, in all possible combinatio

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

$L_1 = \{10, 1\}$ $L_2 = \{011, 1\}$

$L_1 \cdot L_2 = \{10011, 111, 1011\}$

iii) CLOSURE:

* Kleene closure:

The Kleene closure of a language L is denoted L^* is defined as the set of strings that can be formed by taking any number of string from L, defined as,

$$\boxed{L^* = \bigcup_{i=0}^{\infty} L^i}$$

* Positive closure:

The Positive closure of L, denoted by L^+, is the set of string that can be formed by taking any number of string from L excluding ε. defined as

$$\boxed{L^+ = \bigcup_{i=1}^{\infty} L^i = L^* - L^0 = L^* - \{\varepsilon\}}$$

Ex1: $L_1 = \{10, 1\}$

$L^* = \{\varepsilon, 10, 1, 1010, 101, 110, 11, \ldots\}$

$L^+ = \{10, 1, 1010, 101, 110, 11, \ldots\}$

Ex2:

$0^* = \{\varepsilon, 0, 00, 000, \ldots\}$

$0^+ = \{0, 00, 000, \ldots\}$

Construct the regular Expression for the following:

1) String with either a single 0 followed by any number of 1's or a single 1 followed by any number of 0's.

Ans:

$$01^* + 10^*$$

2) String consisting of zero or more occurrence of 01

Ans:

$$(01)^*$$

3) The set of all strings {0,1} starting and ending with the symbol zero.

Ans:

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

4) The set of all strings {a,b} should end with aba

Ans:

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

5) The string with any number of a's followed by any number of b's and any number of c's.

Ans:

$$a^*b^*c^*$$

6) The set of all string over alphabet {a,b,c} containing atleast one a and one b.

Ans:

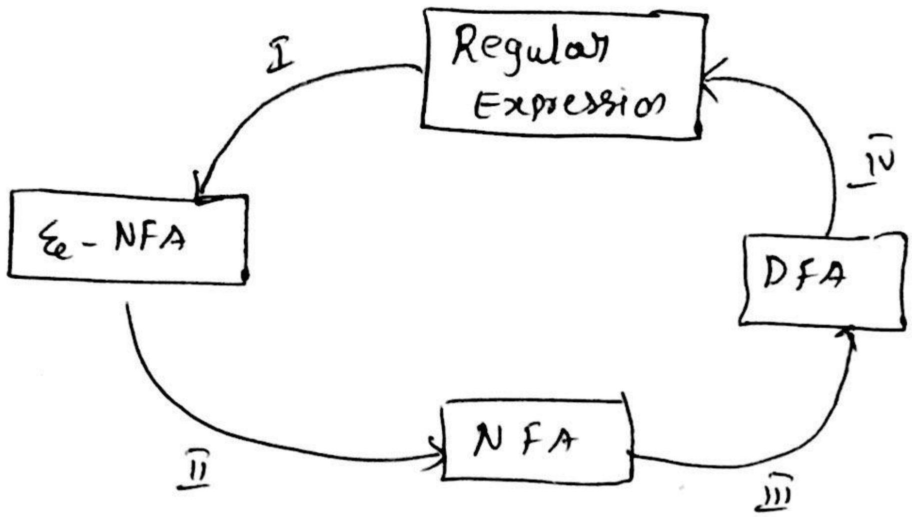
$$(a+b+c)^*ab$$

7) The set of strings of 0's and 1's whose number of 0's is divisible by five.

Ans:

$$(00000+1)^*$$

FINITE AUTOMATA and REGULAR EXPRESSION:



Converting Regular Expressions to Automata:

Theorem:

Let L be a regular expression, then there exists an NFA with ε - transitions that accepts $L(r)$.

or

Every language defined by a regular expression is also defined by a finite automaton.

Proof:

To prove $L(M) = L(r)$ for some ε-NFA M.

Basis

i) Exactly one accepting state

$r = \epsilon$

ii) No arcs in to the initial state

$r = \phi$

iii) No arcs out of the accepting state

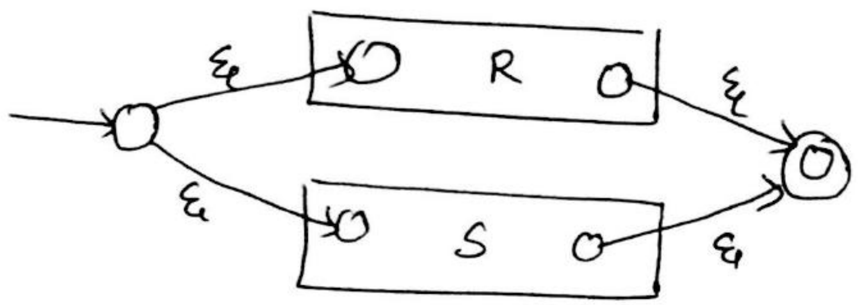
$r = a$

From the above figure it is clear that the expression r must be ϵ, ϕ or a for some a in ϵ.

Induction:

Assume that the theorem is true for the immediate sub expressions of a given regular expressions.

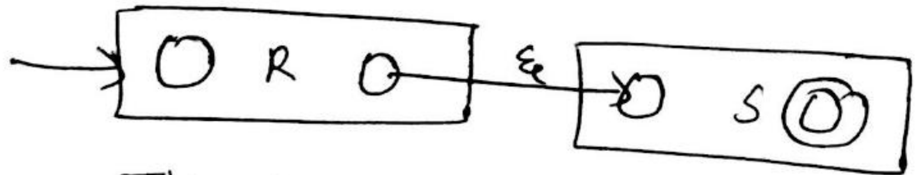
case 1:



$r = R + S$

The expression is $R+S$ for some smaller expressions R and S. Right from the start state, we can go to the start state of either the automaton for R or automaton for S. Finally reach the accepting state of automata. Thus the language of automaton is $L(R) \cup L(S)$.

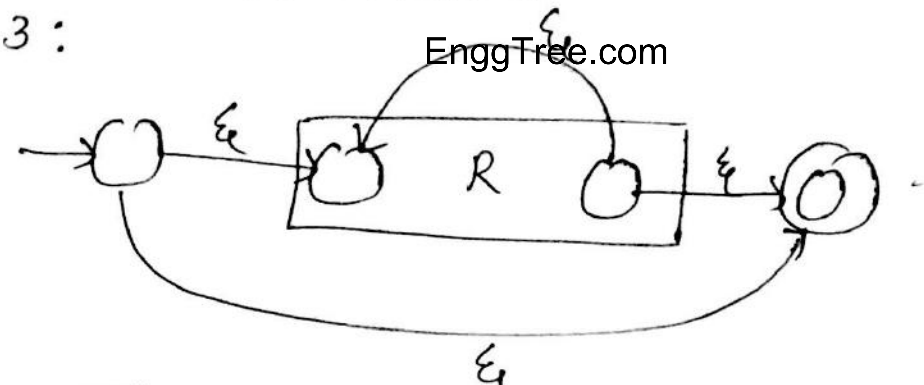
case 2:



$r = R \cdot S$

The expression RS, the start state of the first automaton becomes the start state and the accepting state of the second automata becomes the accepting state of the whole. Thus the language of automaton is $L(R) \cdot L(S)$

Case 3:



The expression is given by R^*, directly from the start state to accepting state along a path labled ε. That ε also belongs to R^*.

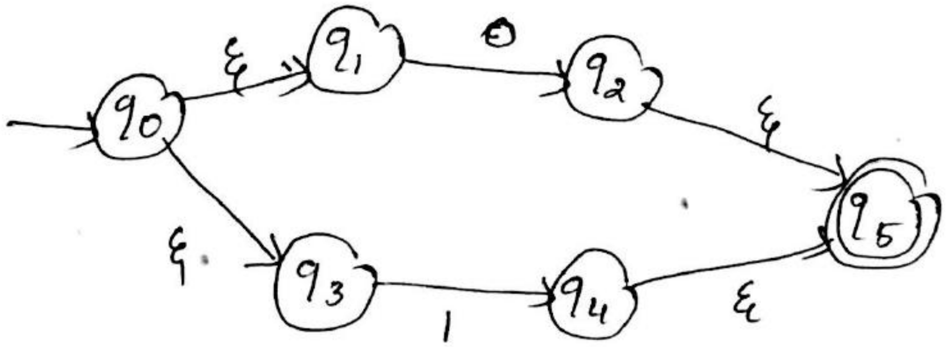
Thus the automaton satisfy the three conditions given in the inductive hypothesis - one accepting state, with no arcs in to the initial state or out of the accepting state.

— x —

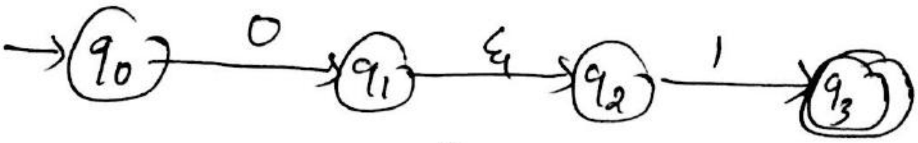
<u>Ex:</u>

Convert the R.E to ε NFA

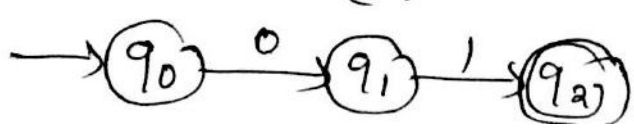
1) 0+1



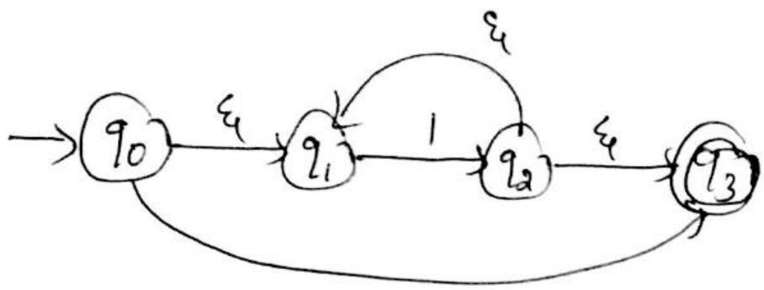
2) 01



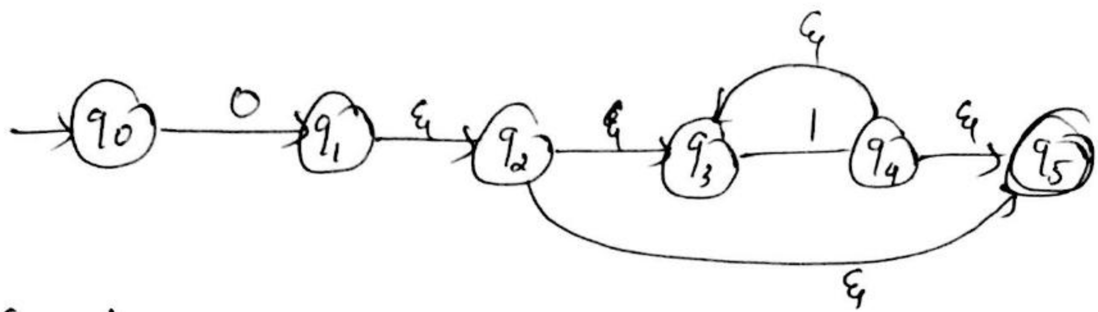
(or)



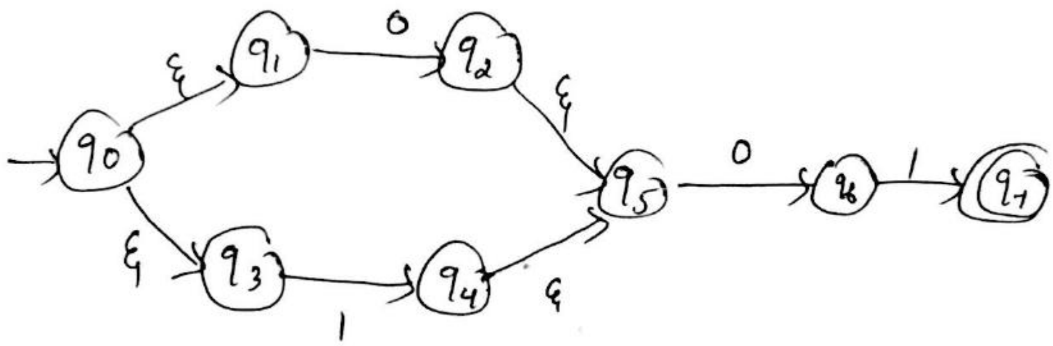
3) 1^*



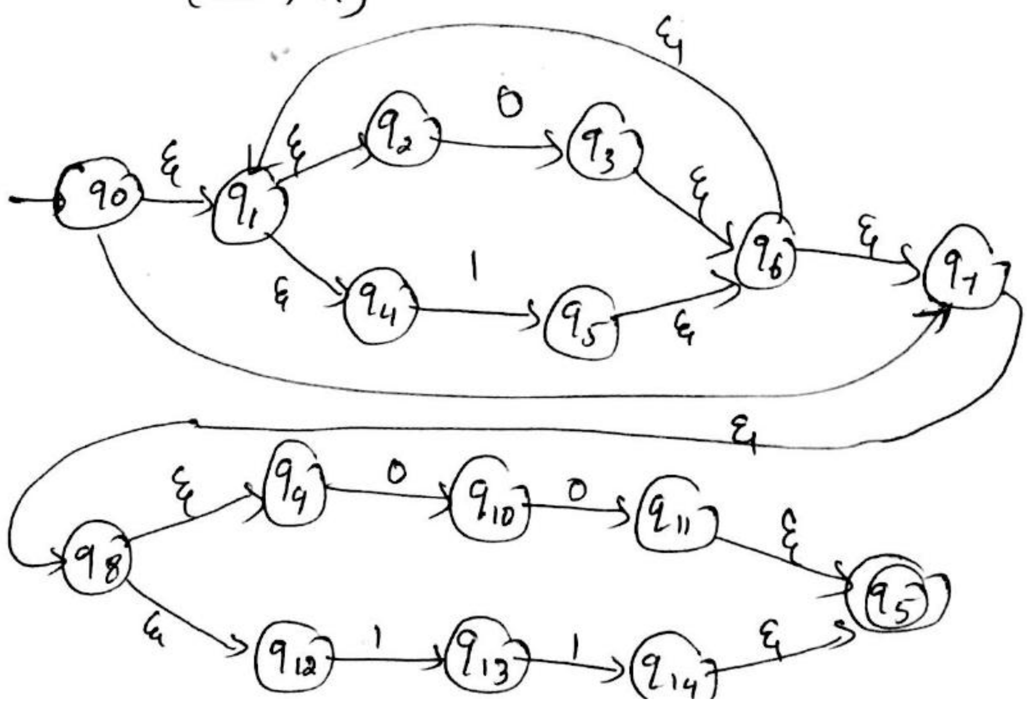
4) 01^*



5) $(0+1)\,01$



6) $(0+1)^*\ (00+11)$



Conversion of ε-NFA into DFA:

Method:

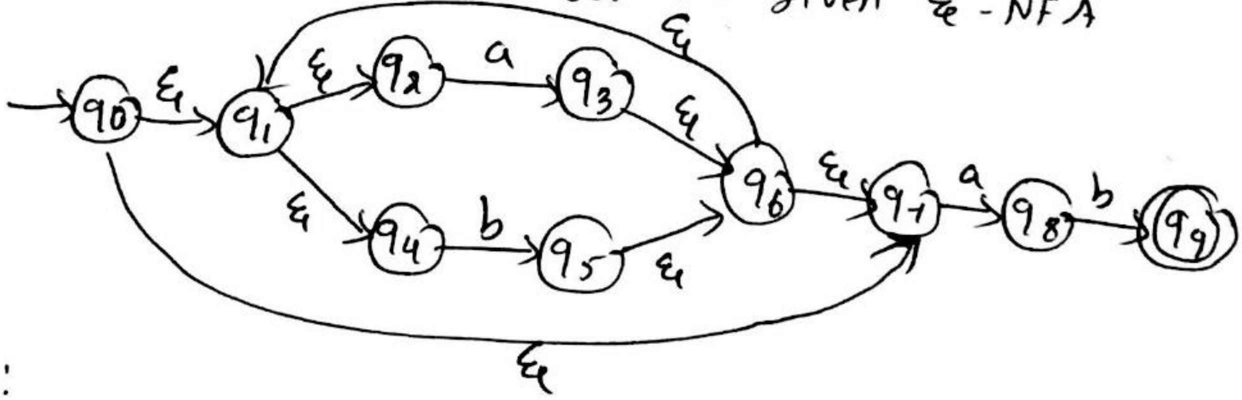
1) Find the ε-closure of the state q_0 from the constructed ε-NFA from state q_0, ε transition to other states are identified as well as ε-transitions from other states are also identified and combined as one set.

2) Perform the following steps until there are no more new states as been constructed.

   a) Find the transition of the given RE symbols over Σ from the new state.

   b) Find the ε-closure of Move(new state, symbol).

Ex 1: Construct the DFA for the given ε-NFA



Soln:

$$\varepsilon\text{-closure}(q_0) = \{q_0, q_1, q_2, q_4, q_7\} \rightarrow A$$

$$\begin{aligned}
Mov(A, a) &= \varepsilon\text{-closure}(\{q_3, q_8\}) \\
&= \varepsilon\text{-closure}(q_3) \cup \varepsilon\text{-closure}(q_8) \\
&= \{q_3, q_6, q_7, q_1, q_2, q_4\} \cup \{q_8\} \\
&= \{q_1, q_2, q_3, q_4, q_6, q_7, q_8\} \longrightarrow B
\end{aligned}$$

$\varepsilon(MOV(A,b)) = \varepsilon\text{-closure}(q_5)$

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

$\varepsilon(MOV(B,a)) = \varepsilon\text{-closure}(q_3, q_8)$

$= \varepsilon\text{-closure}(q_3) \cup \varepsilon\text{-closure}(q_8)$

$= \{q_3, q_6, q_7, q_1, q_2, q_4\} \cup \{q_8\}$

$= \{q_1, q_2, q_3, q_4, q_6, q_7, q_8\} \rightarrow B$

$\varepsilon(MOV(B,b)) = \varepsilon\text{-closure}(q_5, q_9)$

$= \varepsilon\text{-closure}(q_5) \cup \varepsilon\text{-closure}(q_9)$

$= \{q_5, q_6, q_7, q_1, q_2, q_4\} \cup \{q_9\}$

$= \{q_1, q_2, q_4, q_5, q_6, q_7, q_9\} \rightarrow D$

$MOV(C,a) = \{q_3, q_8\}$

$\varepsilon\text{-closure}(MOV(C,a)) = \varepsilon\text{-closure}(\{q_3, q_8\})$

$= \varepsilon\text{-closure}(q_3) \cup \varepsilon\text{-closure}(q_8)$

$= \{q_1, q_2, q_3, q_4, q_6, q_7, q_8\} \rightarrow B$

$MOV(C,b) = \{q_5\}$

$\varepsilon\text{-closure}(MOV(C,b)) = \varepsilon\text{-closure}(q_5)$

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

$MOV(D,a) = \{q_3, q_8\}$

$\varepsilon\text{-closure}(MOV(D,a)) = \varepsilon\text{-closure}(q_3) \cup \varepsilon\text{-closure}(q_8)$

$= \{q_1, q_2, q_3, q_4, q_6, q_7, q_8\} \rightarrow B$

$Mov(D, b) = \{q_5\}$

$\epsilon\text{-closure}(Mov(D, b)) = \epsilon\text{-closure}(q_5)$

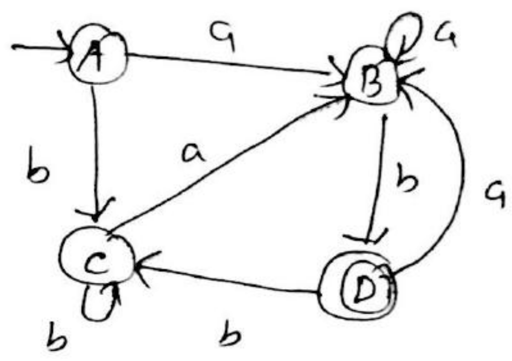
$= \{q_1, q_2, q_4, q_5, q_6, q_7\} \rightarrow C$

$F' = \{D\}$

Transition Table:

δ	input	
	a	b
$\rightarrow A$	B	C
B	B	D
C	B	C
*D	B	C

Transition diagram



— x —

1) Construct a DFA with reduced state equivalent to the regular expression $10 + (0 + 11)0^*1$

2) Construct a NFA for the given RE $((0+1)(0+1))^*$

3) Construct an NFA equivalent to $(0+1)^*(00+11)$

Conversion of DFA to RE:

Theorem: For every DFA $A = (Q, \Sigma, \delta, q_0, F)$, there is a regular expression R, such that $L(R) = L(A)$.

Proof:

Let L be the set accepted by the DFA $A = [\{q_1, q_2, \cdots q_n\}; \Sigma, \delta, q_0, F]$ with q_1 being the start state.

Let $R_{ij}^{(K)}$ be the regular expression describing the set of all string x such that $\delta(q_i, x) = q_j$ going through intermediate states $\{q_1, q_2, \cdots q_K\}$ only

$R_{ij}^{(K)}$ will be defined inductively. Note thate

$$L\left(\bigcup_{j \in F} R_{1j}^{(n)}\right) = L(A)$$

Basis:

$K = 0$, (ie) no intermediate states.

$R_{ij}^{(0)}$ denotes a set of strings which is either ε or single symbol.

Case 1: $i \neq j$

$R_{ij}^{(0)} = \{a \mid \delta(q_i, a) = q_j\}$ denotes set symbols a such that $\delta(q_i, a) = q_j$.

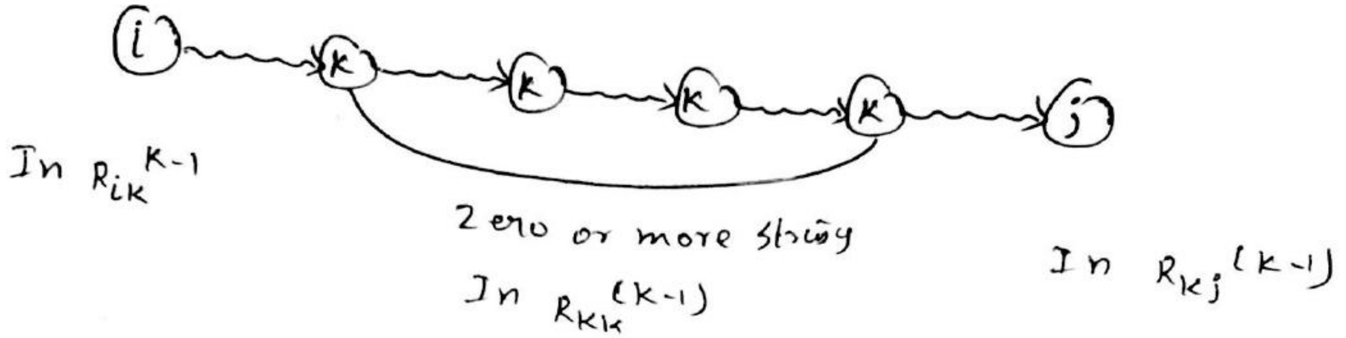
Case 2: $i = j$

$R_{ij}^{(0)} = R_{ii}^{(0)} = (\{a \mid \delta(q_i, a) = q_j\} \cup \{\varepsilon\})$ denotes set of all symbols a such that a or ε. $R_{ii}^{(0)} = a + \varepsilon$

Induction:

It involves regular expression operations union, concatenation and closure.

$$R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} \left(R_{kk}^{(k-1)}\right)^* R_{kj}^{(k-1)}$$



The observation of this proof is that regular expression.

$$L(A) = \bigcup_{q_j \text{ in } F} R_{1j}^{(n)}$$

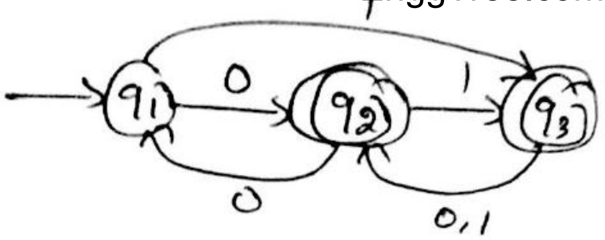
where $R_{1j}^{(n)}$ denotes the labels of all paths from q_1 to q_j.

where $F = \{q_{j_1}, q_{j_2}, \cdots q_{j_p}\}$

So

$$L(A) = R_{1j_1}^{(n)} + R_{1j_2}^{(n)} + \cdots + R_{1j_n}^{(n)}$$

1) Find R for finite automaton given below:



Soln:

We will find $R_{ij}^{(0)}$ where $K=0$

$R_{11}^{(0)} = \epsilon$, $R_{12}^{0} = 0$, $R_{13}^{0} = 1$

$R_{21}^{(0)} = 0$, $R_{22}^{0} = \epsilon$, $R_{23}^{0} = 1$

$R_{31}^{0} = \phi$, $R_{32}^{0} = 0+$, $R_{33}^{0} = \epsilon$.

$K=1$

$$R_{ij}^{(K)} = R_{ij}^{(K-1)} + R_{ik}^{(K-1)} (R_{KK}^{(K-1)})^* R_{Kj}^{(K-1)}$$

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

$$= \epsilon + \epsilon (\epsilon)^* \epsilon.$$

$$R_{12}^{1} = R_{12}^{0} + R_{11}^{0} (R_{11}^{0})^* R_{12}^{0}$$

$$= 0 + \epsilon (\epsilon)^* 0 \quad [\because \epsilon R = R]$$

$$= 0 + 0$$

$$= 0$$

$$R_{13}^{1} = R_{13}^{0} + R_{11}^{0} (R_{11}^{0})^* R_{13}^{0}$$

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

$$= 1 + \epsilon 1$$

$$= 1 + 1$$

$$= 1$$

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

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

$= 0 + 0$

$= 0$

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

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

$= \epsilon + 00$

$R_{23}^{1} = R_{23}^{0} + R_{21}^{0}(R_{11}^{0})^{*} R_{13}^{0}$

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

$= 1 + 01$

$R_{31}^{1} = R_{31}^{0} + R_{31}^{0}(R_{11}^{0})^{*} R_{11}^{0}$

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

$= \phi + \phi$ $\quad [\because \phi R = \phi]$

$= \phi$

$R_{32}^{1} = R_{32}^{0} + R_{31}^{0}(R_{11}^{0})^{*} R_{12}^{0}$

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

$= 0+1 + \phi$

$= 0 + 1$ $\quad [\because \phi + R = R]$

$R_{33}^{1} = R_{33}^{0} + R_{31}^{0}(R_{11}^{0})^{*} R_{13}^{0}$

$= \epsilon + \phi(\epsilon)^{*} 1$

$= \epsilon + \phi$

$= \epsilon$

$\underline{\underline{K=2:}}$

$R_{11}^{2} = R_{11}^{1} + R_{12}^{1}(R_{22}^{1})^{*} R_{21}^{1}$

$= \varepsilon + 0(\varepsilon + 00)^{*} 0$

$= \varepsilon + 0(00)^{*} 0$

$= (00)^{*}$

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

$= 0 + 0(\varepsilon + 00)^{*}(\varepsilon + 00)$

$= 0 + 0(00)^{*}$ $\quad [\because R + RS^{*} = RS^{*}]$

$= 0(00)^{*}$

$R_{13}^{2} = R_{13}^{1} + R_{12}^{1}(R_{22}^{1})^{*} R_{23}^{1}$

$= 1 + 0(\varepsilon + 00)^{*}(1 + 01)$

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

$= 1 + 00^{*}1$ $\quad [R + RS^{*} = RS^{*}]$

$= 00^{*}1$

$= 0^{*}1$

$R_{21}^{2} = R_{21}^{1} + R_{22}^{1}(R_{22}^{1})^{*} R_{21}^{1}$

$= 0 + (\varepsilon + 00)(\varepsilon + 00)^{*} 0$

$= 0 + 00(00)^{*} 0$

$= 0 + (00)^{*} 0$

$= 0(00)^{*}$

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

$= (\epsilon + 00) + (\epsilon + 00)(\epsilon + 00)^{*}(\epsilon + 00)$

$= 00 + 00(00)^{*}00$

$= 00 + (00)^{*}$

$= (00)^{*}$

$R_{23}^{2} = R_{23}^{1} + R_{22}^{1} (R_{22}^{1})^{*} R_{23}^{1}$

$= (1+01) + (\epsilon + 00)(\epsilon + 00)^{*}(1+01)$

$= (1+01) + 00(00)^{*}(1+01)$

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

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

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

$= 0^{*}1$

$R_{31}^{2} = R_{31}^{1} + R_{32}^{1} (R_{22}^{1})^{*} R_{21}^{1}$

$= \phi + (0+1)(\epsilon + 00)^{*}0$

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

$= (0+1)(00)^{*}0$

$R_{32}^{2} = R_{32}^{1} + R_{32}^{1} (R_{22}^{1})^{*} R_{22}^{1}$

$= (0+1) + (0+1)(\epsilon + 00)(\epsilon + 00)$

$= (0+1) + (0+1)(00)^{*}$ $[R + RS^{*} = RS^{*}]$

$= (0+1)(00)^{*}$

$$R_{33}^{2} = R_{33}^{1} + R_{32}^{1}(R_{22}^{1})^{*} R_{23}^{1}$$

$$= \epsilon + (0+1)(\epsilon + 00)^{*}(1+01)$$

$$= \epsilon + (0+1)(00)^{*}(1+01)$$

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

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

$$L(M) = R_{12}^{3} + R_{13}^{3}$$

$$R_{12}^{3} = R_{12}^{2} + R_{13}^{2}(R_{33}^{2})^{*} R_{32}^{2}$$

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

$$= 0(00)^{*} + 0^{*}1((0+1)0^{*}1)^{*}(0+1)(00)^{*}$$

$$R_{13}^{3} = R_{13}^{2} + R_{13}^{2}(R_{33}^{2})^{*} R_{33}^{2}$$

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

$$= 0^{*}1[\epsilon + (0+1)0^{*}1)^{*}(0+1)0^{*}1]$$

$$= 0^{*}1((0+1)0^{*}1)^{*} \quad [\epsilon + R^{*}R = R^{*}]$$

$$R_{12}^{3} + R_{13}^{3} = 0(00)^{*} + 0^{*}1((0+1)0^{*}1)^{*}(0+1)(00)^{*} + 0^{*}1[(0+1)0^{*}1)^{*}$$

$$[R + RS^{*} = RS^{*}]$$

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

Pumping Lemma for Regular sets: [Applications]

Pumping lemma is used to check whether certain sets are regular or not.

The steps to prove that certain sets are not regular are as follows:

1. Assume L is regular. There exists a constant n be the number of states in the corresponding FA.
2. Choose a string z such that $|z| \geq n$. use the pumping lemma to write $z = uvw$ with $|uv| \leq n$ and $|v| > 0$
3. Find a suitable integer i such that $uv^iw \notin L$. In some cases we prove $uv^iw \notin L$ by considering $|uv^iw|$. In some cases we have to use the structure of string in L.

Ex 1:

Show that $L = \{a^p \mid p \text{ is a prime}\}$ is not regular

Soln:

let $z = a^k \in L$

representing $z = uvw$

such that $|uv| \leq k$, $|v| \geq 1$

let

$uv = a^m$ where $m < k$

$v = a^j$ where $j < m$

$w = a^{k-m}$

$$uv^i w = uvv^{i-1}w$$
$$= a^m a^{j(i-1)} a^{k-m}$$
$$= a^{m+j(i-1)+k-m}$$
$$= a^{j(i-1)+k}$$

for $i=0$,

$$uv^i w = a^{j(0-1)+k}$$
$$= a^{k-j} \notin L \text{ since } k-j \text{ is not prime}$$

Hence the given language is not regular.

Proving languages Not to be Regular:

Pumping Lemma for Regular languages is used as a tool to prove that certain languages are not regular. The principle behind this Lemma "Pigeon Hole Principle".

PIGEON HOLE PRINCIPLE:

If 'n' objects are put in to 'm' containers, where $n > m$, then at least one container must hold more than one objects.

2.7 Closure Properties of Regular Languages

To prove that certain languages are regular languages. The language L is formed by certain operations then L is regular. The main key objective is that one language is regular then certain related languages are also regular.

Example: Let L is a regular language and M is a language which is related to L then $L \cup M$ is also a regular language.

Properties:

Union: $L \cup M$

Concatenation: $L \cdot M$

Closure: L^*

Intersection: $L \cap M$

Complement: $\overline{L}$

Difference: $L - M$

Reversal: LR

Homomorphism: h (L)

Inverse Homomorphism: h − 1(L)

1. **Union**

   Let L and M are languages over alphabet Σ. Then $L \cup M$ is the language that contains all strings that are in either L or M.

   Theorem:

   If L and M are regular languages, so is $L \cup M$.

   Proof: Let L and M be the languages can be represented by the regular expressions R and S, respectively.

$$L = L(R), M = L(S)$$

   Then $L \cup M = L(R) + L(S)$

   $L \cup M = L(R + S)$

   Then R + S are a regular expression whose language is $L \cup M$.

2. **Concatenation**

   Let L and M are languages over alphabet Σ. Then L.M is the language that contains all strings that are in L and M

Theorem:

If L and M are regular languages, so is L.M or LM.

Proof:

Let L and M be the languages can be represented by the regular expressions R and S, respectively.

$$L = L(R), M = L(S)$$

$$L . M = L (R) . L (S)$$

$$L.M = L (R.S)$$

Then R.S is a regular expression whose language is L.M.

3. **Closure**

Let L be the language over alphabet Σ. Then L* is the language that contains set of all strings from L.

Theorem:

If L is regular language, so is L*.

Proof:

Let L is a language can be represented by the regular expressions R respectively.

$$L = L(R)$$

$$L^* = L(R)^*$$

R* is a regular expression whose language is L*.

4. **Intersection**

Let L and M are languages over alphabet Σ. Then $L \cap M$ is the language that contains all strings that are in both L and M.

If L and M are regular languages, then so is $L \cap M$.

Proof:

Let L and M regular languages represented by R_L and R_M accepted by finite automata A_L and A_M respectively.

Let L be the language of $A_L = (Q_L, \Sigma, \delta_L, q_L, F_L)$ accepts R_L and M be the language of $A_M = (Q_M, \Sigma, \delta_M, q_M, F_M)$ accepts R_M.

Design an automata A (as shown in Figure 2.4) that simulates A_L and A_M in parallel, and accepts if and only if both A_L and A_M accept.

If A_L goes from state p to state s on reading input symbol a, and A_M goes from state q to state t on reading input symbol a, then $A_{L\cap M}$ will go from state (p, q) to state (s, t) on reading a.

$$\delta(p, a) = s$$
$$\delta(q, a) = t$$

Then

$$\delta(\{p, q\}, a) = \{s, t\}$$

Formally $A_{L\cap M} = (Q_L \times Q_M, \Sigma, \delta_{L\cap M}, (q_L \times q_M), F_L \times F_M)$,

where $\delta_{L\cap M}$ (p, q), a) = (δ_L (p, a), δ_M (q, a)) = $\{s, t\}$

$$L(A) = L(A_L) \cap L(A_M)$$

The input string w is accepted by A, if and only if both A_L and A_M accepts the string w. The final states of A be the pairs consisting of final states of both A_L and A_M.

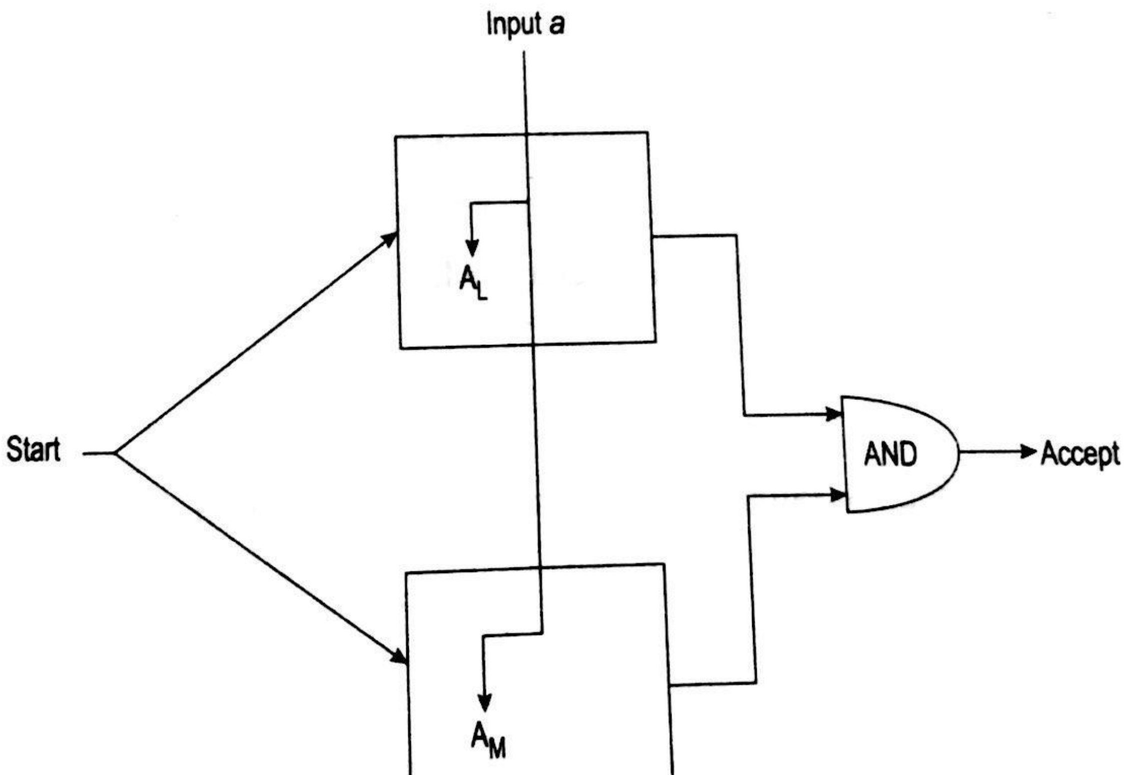


Figure 2.4 Intersection of Finite Automata

Intersection can be related by using union, complement and intersection

$$L \cup M = \overline{(\overline{L} \cap \overline{M})}$$

$$L \cap M = \overline{(\overline{L} \cup \overline{M})}$$

5. **Complementation:**

Theorem:

Let L is a regular language over alphabet Σ, and then complement of a language L is also a regular language.

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

Proof:

Let L = L (A) for some DFA A = (Q, Σ, δ, q, F). Then $\overline{L}$ = L (B) for some DFA B = (Q, Σ, δ, q, Q−F). Accepting states of A become non accepting states of B. Then w is in L (B) if and only if $\delta^\wedge(q_0, w)$ is in Q-F, which occurs if and only if w is not in L (A).

6. **Difference**

Let L and M are languages over alphabet Σ, and then L−M is the set of strings that are in language L but not in language M.

Theorem:

If L and M are regular languages, then so is L − M = {Strings in L but not in M}.

Proof: $$L - M = L \cap \overline{M}$$

Let A and B be Finite Automata's whose languages are L and M, respectively. Construct C, the product automaton of A and B. The final states of C be the pairs where A-state is final state but B-state is not a final state.

With reference to the theorem for complement of a regular language is also a regular language

i.e., $\overline{M}$ is regular language.

With reference to the theorem for intersection of a regular language is also a regular language.

i.e., $L \cap \overline{M}$ is also a regular language.

Therefore L-M is also a regular language.

7. **Reversal**

Let L is a regular language over alphabet Σ, then L^R is the language consisting of the reversals of all its strings.

$$w = a_1 a_2 \ldots a_n$$
$$w^R = a_n a_{n-1} \ldots a_2 a_1$$

Theorem:

Let L is a regular language, so is L^R.

Proof:

Let E be a regular expression for regular language L. We show how to reverse E, to provide a regular expression E^R for L^R

Given language L, L^R is the set of strings whose reversal is in L.

Basis:

If E is a symbol a, ε, or $\emptyset$, then $E^R = E$.

$$(a)^R = a$$
$$(\varepsilon)^R = \varepsilon$$
$$(\varepsilon)^R = \varepsilon$$

Induction:

Three cases are

Case 1:

If E is $E_1 + E_2$, then $E^R = E_{1R} + E_{2R}$

Case 2:

If E is $E_1 + E_2$, then $E^R = E_{2R} . E_{1R}$

Case 3:

If E is $E1^*$, then $E^R = (E_{1R})^*$

The string is in L(E) if and only if its reversal is in $(E_{1R})^*$

Example: $L = \{0, 01, 100\}$; $L^R = \{0, 10, 001\}$.

8. **Homomorphism**

A homomorphism is a function on strings that works substituting a particular string for each symbol in that alphabet.

Example: $h(0) = ab$; $h(1) = \varepsilon$.

Extend to strings by $h(a_1 \ldots a_n) = h(a_1) \ldots h(a_n)$.

$h(01010) = h(0)\, h(1)\, h(0)\, h(1)\, h(0) = ab\ \varepsilon\ ab\ \varepsilon\ ab = ababab$

Theorem:

If L is a regular language over alphabet Σ, and h is a homomorphism on its alphabet Σ, then h (L) is also a regular language (Refer Figure 2.5).

$$h(L) = \{h(w) | w \text{ is in } L\}$$

Proof:

Let $L = L(R)$ for some regular expression R. Let E be a regular expression with symbols in Σ. Apply h to each symbol in E. Language of resulting Regular expression is h(L).

$$L(h(E)) = h(L(E))$$

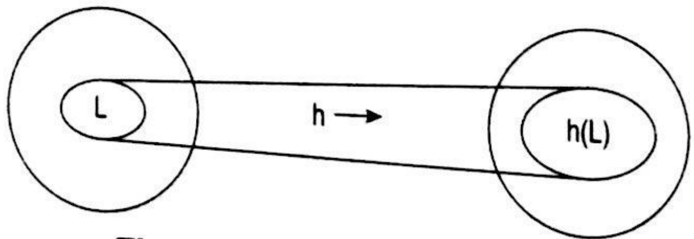


Figure 2.5 Homomorphism

Basis:

Let E is a symbol represented as ε, $\emptyset$ and a.

If E is ε, $\emptyset$ then h (E) same as E. L(E) contains either no strings or no symbols. Homomorphism h does not affect the string ε or $\emptyset$.

$$L(h(E)) = h(L(E)) = L(E)$$

If E is a, a in Σ

$L(E) = \{a\}$,

$$L\ (E) = \{h(a)\}$$

$$L(h(E)) = h(L(E))$$

Induction:

Three cases

Case 1: Union

$$E = F + G.$$

Apply homomorphism

$$h(E) = h(F.G) = h(F).h(G)$$

$$L(h(F\ .G)) = L(h(F).h(G))$$

$$L(h(E))= L(h(F).h(G))$$

$$= L(h(F)).L(h(G))$$

$$h(L(E)) = h((L(F)).h((L(G))$$

Case 3: Closure

$$E = E^*.$$

Apply homomorphism

$$h(E) = h((E))^*$$

$$L(h(E)) = L(h(E))^*$$

$$h(L(E)) = h((L(E))^*$$

Inverse Homomorphism:

Let h be a homomorphism and L a language whose alphabet is the output language of h. Homomorphism applied in backwards (Figure 2.6). It is the set of strings w in Σ^* such that h(w) is in L.

$$h^{-1}(L) = \{w|h(w) \text{ is in } L\}$$

Theorem:

If h is a homomorphism from alphabet Σ to alphabet T and L is a regular language over T, the h^{-1} (L) is also regular language.

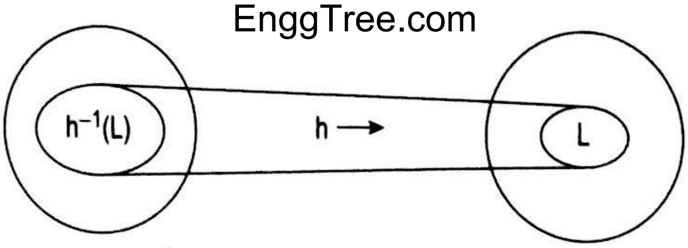


Figure 2.6 Inverse Homomorphism

Proof:

Construct Automaton A and h for DFA (Refer Figure 2.7). It defines $h^{-1}(L)$. DFA uses the states of A but translates the input symbol according to h before deciding on the next input symbol. Let L be L (A), A = (Q, T, δ, q, F). Define DFA B= (Q, Σ, γ, q, F). Transition function δ is constructed by the rule

$$\gamma(q, a) = \delta^\wedge(q, h(a))$$

Transition B makes on the string of symbols h(a), it could be ε , one symbol or many symbols.

$$\widehat{\gamma}(q_0, w) = \widehat{\delta}(q_0, h(w))$$

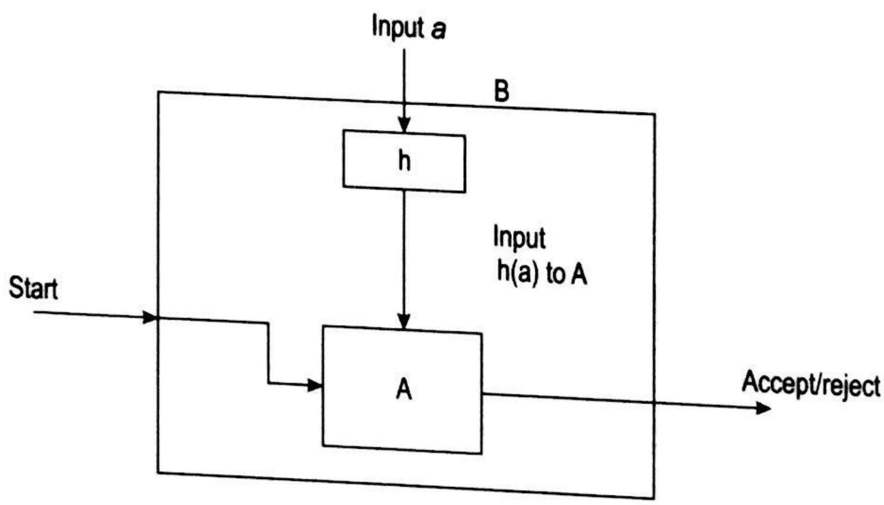


Figure 2.7 Homomorphism in FA

The accepting states of A and B are the same. B accepts w if and only if A accepts h (w) and B accepts exactly those strings w that are in h^{-1} (L).

Problems:

1. Construct the Finite Automata for the regular expression set of all strings 0, 1 that end in 01.

   Convert into DFA and find the complement of that FA

UNIT III CONTEXT FREE GRAMMAR AND LANGUAGES 9

CFG – Parse Trees – Ambiguity in Grammars and Languages – Definition of the Pushdown Automata – Languages of a Pushdown Automata – Equivalence of Pushdown Automata and CFG, Deterministic Pushdown Automata.

Grammar Introduction:

A grammar is a set of production rules for generating string in a language.

A grammar generates a language with the aid of 4 elements $G = (V, T, P, S)$ where,

V – is a finite nonempty set whose elements are called <u>variables</u> (Non terminals)

T – is a finite nonempty set whose elements are called <u>terminals</u>

P – is a finite set whose elements are $\alpha \rightarrow \beta$ where α and β are strings on $V \cup T$ elements of P are called <u>productions</u>

S – is a special variable called the <u>start symbol</u>

Ex1: construct a grammar for $L = \{a^n \mid n \text{ is odd}\}$.

Soln:

$S = a^1 = a \rightarrow [S \rightarrow a]$

$S = a^3 = aaa \rightarrow [S \rightarrow aSa]$

$\therefore$ Productions

$S \rightarrow a$

$S \rightarrow aSa$

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

$V = \{S\}$

$T = \{a\}$

$P = \{S \rightarrow a \mid aSa\}$

$S = \{S\}$

— x —

Ex2: Let $G = (\{S\}, \{a, b\}, P, S)$ with the production $P: S \rightarrow aSa, S \rightarrow bSb, S \rightarrow \epsilon$. Find the language generated by the grammar.

Soln:

1) $S \Rightarrow aSa$
$\Rightarrow aa \quad [S \rightarrow \epsilon]$

2) $S \Rightarrow bSb$
$\Rightarrow bb \quad [\because S \rightarrow \epsilon]$

3) $S \Rightarrow bSb$
$\Rightarrow baSab \quad [\because S \rightarrow aSa]$
$\Rightarrow baab \quad [\because S \rightarrow \epsilon]$

4) $S \Rightarrow aSa$
$\Rightarrow abSba \quad [S \rightarrow bSb]$
$\Rightarrow abba \quad (S \rightarrow \epsilon)$

$\therefore$ language can be defined as,

$$\boxed{L = \{ww^R \mid w \in (a, b)^*\}}$$

Types of Grammar:

Type	Grammar/Language	Form of Production in grammar	Example	Automata
0	Unrestricted grammar (or) Phase structure grammar	A grammar without any restriction with the following production form: $\phi A \psi \rightarrow \phi \alpha \psi$ where A — variable ϕ — left context ψ — right context $\phi \alpha \psi \Rightarrow$ replacement string	1) $ab\underline{A}cd \rightarrow ab\underline{AB}bcd$ where ab - left context bcd - right context α - AB 2) $A\underline{C} \rightarrow A$ where $A \rightarrow$ left context λ - right context α - λ	Turing Machine (TM)
1	Context-sensitive grammar (or) Context dependent grammar	A production of the form $\phi A \psi \rightarrow \phi \alpha \psi$ is called type 1 production if $\alpha \neq \lambda$. In type 1 productions erasing of A is not permitted	1) $a\underline{A}bcD \rightarrow a\underline{bCD}bcD$ where a - left context bcD - right context A is replaced by $bcD \neq \lambda$ 2) $AB \rightarrow AbBc$ Where A - left context λ - right context $\alpha = bBc \neq \lambda$	Linear Bounded Automata

Type	Grammar/language	Form of production in grammar	Example	Automata
2	Context Free Grammar (CFG)	A production of the form $A \rightarrow \alpha$, where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$ is called a type 2 grammar	$S \rightarrow Aa$ $A \rightarrow a$ $B \rightarrow abc$ $A \rightarrow \lambda$ $A = L.H.S \in V$ $\alpha = R.H.S \in (V \cup \Sigma)^*$	Push down Automata (PDA)
3	Regular Grammar	A production of the form $A \rightarrow a$ or $A \rightarrow aB$ where $A, B \in V$ and $a \in \Sigma$ is called a type 3 production. A production $S \rightarrow \lambda$ is allowed in type 3, but in this case S does not appear on the right	$S \rightarrow b/c$ $S \rightarrow bA$ $S \rightarrow a$ $A = L.H.S \in V$ $R.H.S \in (V \cup a)$	Finite Automata

Context - Free - Grammars and languages :

A grammar is means of representing a language.

A context - free Grammar (CFG) is described by four tuples as,

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

where,

V is a finite set of variables

T is a finite set of terminals

P is a finite set of productions

S is a start symbol.

The language of a grammar :

Every string of a language is generated by applying the production rules, a finite number of times.

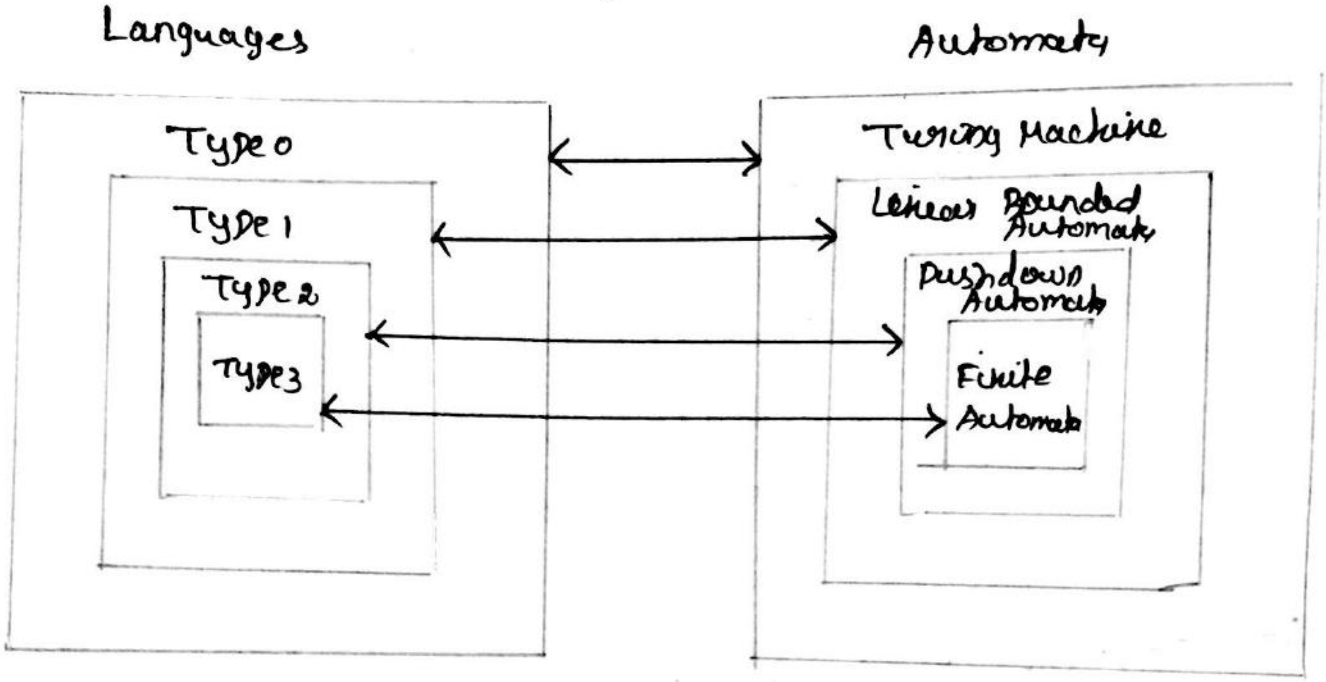
The language of a grammar, $G = (V, T, P, S)$ is denoted as $L(G)$.

$$\boxed{L(G) = \{ w \mid w \in T^* \text{ and } S \overset{*}{\underset{G}{\Rightarrow}} w \}}$$

The language generated from a context free grammar is called a context free language.

Language and Automata:

The following describes the relation between the four types of languages and automata.



Ex:

1) Find the highest number which can be applied to the following grammar.

a) $S \to Aa,\ A \to c \mid Bc,\ B \to abc$

b) $S \to ASB \mid d,\ A \to aA$

c) $S \to aS \mid ab$

Soln:

a) $S \to Aa,\ B \to Ba,\ B \to abc$ — type 2

[because productions are of the form $A \to \alpha$]

$A \to a$ — type 3 [form $A \to a$]

∴ highest type number is 2.

1) construct a CFG for the language, $L = \{wcw^R / w \in (a,b)^*\}$

Soln:

The possible strings generated from L can be $\{c, aca, bcb, aacaa, bbcbb, abcba, bacab, \ldots\}$

$\therefore$ Production rules are defined to be,

$S \rightarrow c$

$S \rightarrow aSa$

$S \rightarrow bSb$

Thus the grammar, $G = (V, T, P, S)$ where,

$V = \{S\}$

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

$P = \{S \rightarrow aSa | bSb | c\}$

$S = \{S\}$

—x—

2) construct a CFG for $L = \{a^n b^n / n \geq 0\}$

Soln:

The possible productions are $\{\epsilon, ab, aabb, a^4 b^4, \ldots\}$

$\therefore$ Productions are

$S \rightarrow \epsilon$

$S \rightarrow aSb$

Thus the CFG, $G = (V, T, P, S)$ is given by

$V = \{S\}$

$T = \{a, b\}$

$P = \{S \rightarrow aSb | \epsilon\}$

$S = \{S\}$

Derivations and languages:

Derivations:

Derivations are the set of strings that are derived from the start symbol, after applying the production rules, a finite number of times.

$$\boxed{S \overset{*}{\underset{G}{\Rightarrow}} w \,/\, w \in T^*}$$

Representation of derivations:

Derivations are represented in either of the two forms:

* Sentential Form
* Parse tree Form

Types of Derivations:

There are two types of derivations,

* Leftmost derivation [LMD]
* Rightmost derivation [RMD]

1. Sentential Form:

Sentential form is the derivation, derived from the start symbol, by applying the rules.

Let $G = (V, T, P, S)$ be a CFG then

$$S \overset{*}{\Rightarrow} a$$

is in sentential form, where $a \in (T \cup V)^*$

Example:

Let $G = (V, T, P, S)$ be given by $G = (\{S, A, B\}, \{0,1\}, P, \{S\})$

$P: S \rightarrow A1B, \ A \rightarrow 0A \mid \epsilon, \ B \rightarrow 0B \mid 1B \mid \epsilon.$

Soln:

Consider the string 00101, which is to be generated by the given grammar using sentential form.

$$\begin{aligned}
S &\Rightarrow A1B \\
&\Rightarrow 0A1B \quad [A \rightarrow 0A] \\
&\Rightarrow 00A1B \quad [A \rightarrow 0A] \\
&\Rightarrow 001B \quad [A \rightarrow \epsilon] \\
&\Rightarrow 0010B \quad [B \rightarrow 0B] \\
&\Rightarrow 00101B \quad [B \rightarrow 1B] \\
&\Rightarrow 00101 \quad [B \rightarrow \epsilon]
\end{aligned}$$

Thus the string $00101 \in L(G)$.

Types of Sentential Form

There are 2 types of sentential forms namely,

* Left sentential forms
* Right sentential forms

Left Sentential Forms:

When the derivations are generated by expanding the leftmost symbol is called left sentential form.

$$\boxed{S \overset{*}{\Rightarrow} \alpha}$$

Examples:

1) consider the grammar given below $S \to A1B$, $A \to 0A/\epsilon$, $B \to 0B/1B/\epsilon$. Given the leftmost derivation for the string 1001.

Soln:

$$S \underset{lm}{\Rightarrow} A1B$$

$$\underset{lm}{\Rightarrow} 1B \quad [\because A \to \epsilon]$$

$$\underset{lm}{\Rightarrow} 10B \quad [\because B \to 0B]$$

$$\underset{lm}{\Rightarrow} 100B \quad [\because B \to 0B]$$

$$\underset{lm}{\Rightarrow} 1001B \quad [\because B \to 1B]$$

$$\underset{lm}{\Rightarrow} 1001 \quad [\because B \to \epsilon]$$

Right sentential Form:

The derivation generated by performing substitution on the right most variable is called right sentential form.

$$\boxed{S \overset{*}{\underset{rm}{\Rightarrow}} \alpha}$$

Example:

consider the Grammar, $S \to A1B$, $A \to 0A/\epsilon$, $B \to 0B/1B/\epsilon$. Generate the right most derivation for the String 1001.

Soln:

$$S \underset{rm}{\Rightarrow} A1B$$

$$\underset{rm}{\Rightarrow} A10B \quad [\because B \to 0B]$$

$$\underset{rm}{\Rightarrow} A100B \quad [\because B \to 0B]$$

$$\underset{rm}{\Rightarrow} A1001B \quad [\because B \to 1B]$$

$$\underset{rm}{\Rightarrow} A1001 \quad [\because B \to \epsilon]$$

$$\underset{rm}{\Rightarrow} 1001 \quad [\because A \to \epsilon]$$

— x —

1) Let G be the grammar P: {S → aB | bA, A → a | aS | bAA, B → b | bS | aBB}. For the string aabbbbaa, find LMD and RMD.

Soln:

LMD:

$$S \underset{lm}{\Rightarrow} aB$$

$$\underset{lm}{\Rightarrow} aaS \quad [\because B \to aS]$$

$$\underset{lm}{\Rightarrow} aabA \quad [\because S \to bA]$$

$$\underset{lm}{\Rightarrow} aabbAA \quad [\because A \to bAA]$$

2. Derivation Trees / Parse Trees:

The representation of a derivation in the form of a tree is called a parse tree / derivation tree.

$v_1 \to \alpha$ where $v_1 \in V$ an $\alpha \in T$, then v_1 can have any number of children based on its production rules.

If $v \to \varepsilon$, then ε should be the only child of v_1.

Example:

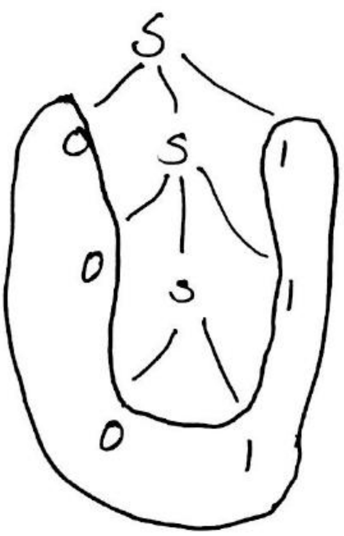
For the grammar $S \to 0S1 \mid 01$, generate derivation of the string 000111.

Soln:

The derivation is given by,

$$S \Rightarrow 0S1$$
$$\Rightarrow 00S11 \quad [\because S \to 0S1]$$
$$\Rightarrow 000111 \quad [\because S \to 01]$$

The parse tree is given by



Types of Parse Tree:

There are 2 types of parse tree

* Left most derivation tree / left derivation tree
* Right most derivation tree / Right derivation tree

Left most derivation Tree:

A derivation tree representing $A \Rightarrow \alpha$ is called a leftmost derivation tree if the production rule is applied only to the leftmost variable at every step.

Right most derivation Tree:

A derivation tree representing $A \Rightarrow \alpha$ is said to be a rightmost derivation tree if ~~there~~ the production rule is applied only to the rightmost variable at every step.

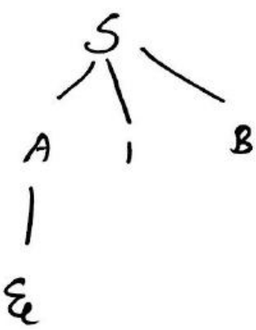
Example:

1. For the grammar given below give the parse tree for leftmost and rightmost derivation of the string 1001.

Soln:

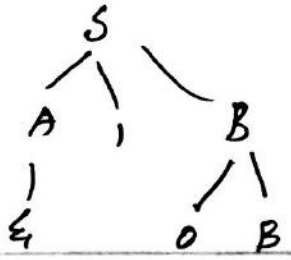
Left most derivation tree:

1)



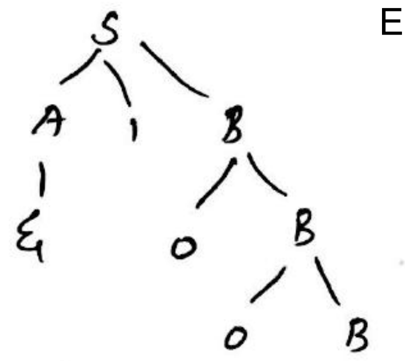
using $A \rightarrow \epsilon$.

2)



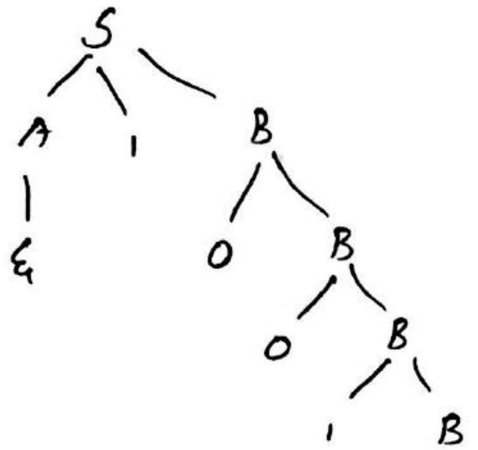
using $B \rightarrow 0B$

3)



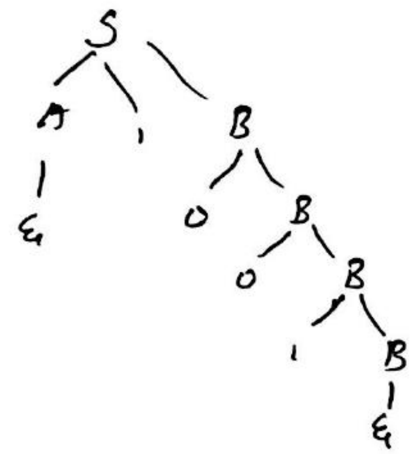
using $B \to 0B$

4)



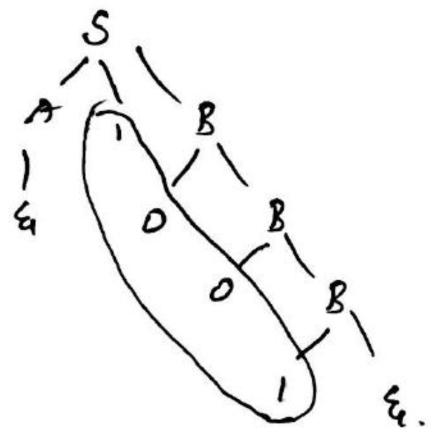
using $B \to 1B$

5)



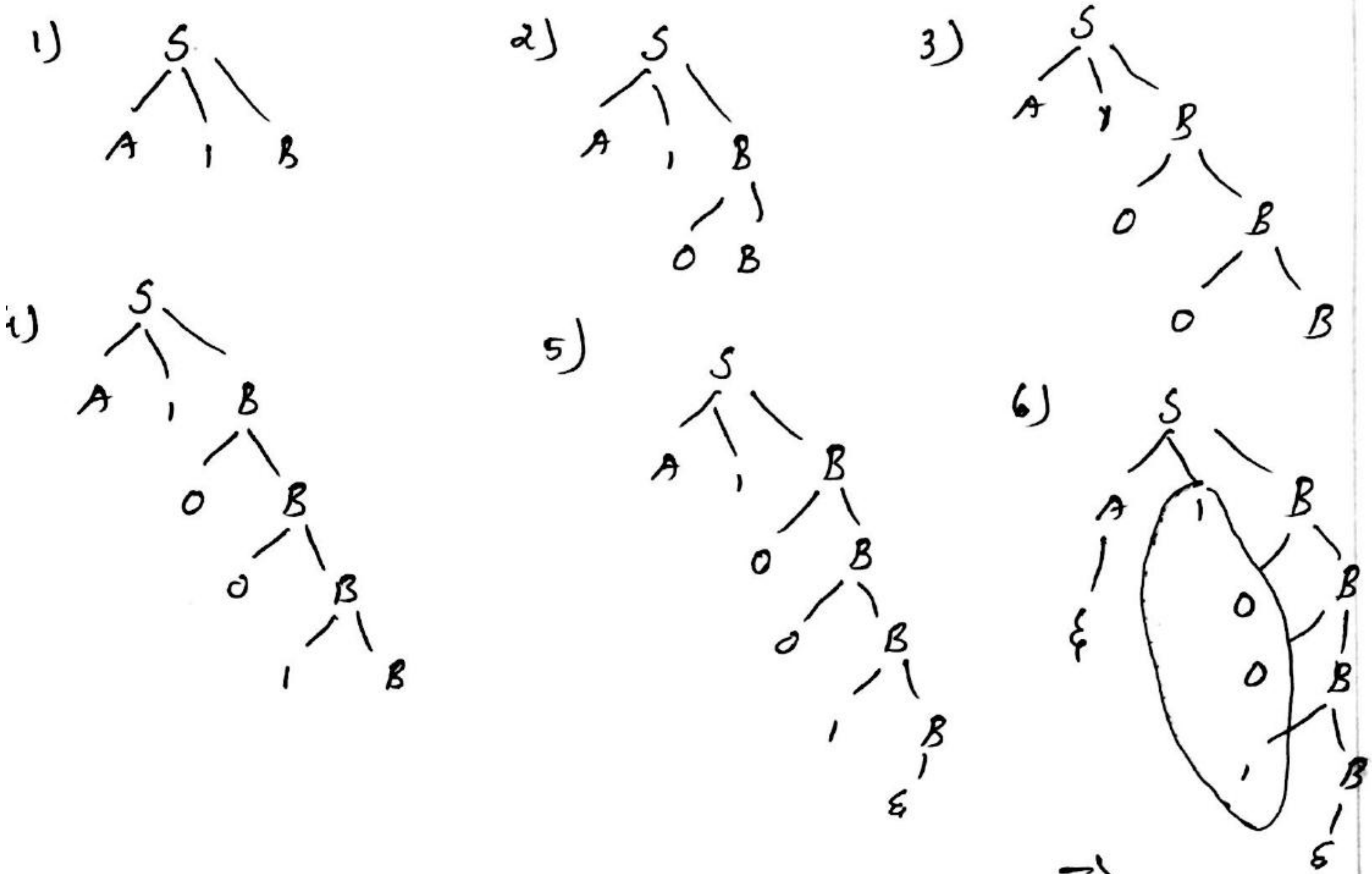
using $B \to \epsilon$.

6)



$\underset{lm}{\Rightarrow} 1001$

Right most derivation tree:



$\underset{lm}{\Rightarrow}$ 1001

Recursive Production / Inference:

A production is said to be recursive if the left side variable occurs on the right hand side that substitutes the same production for 'n' number of times.

Example:

S->aS => thus would lead to application of the same rule recursively as S =>aS =>aaS =>aaaS=>....

AMBIGUITY:

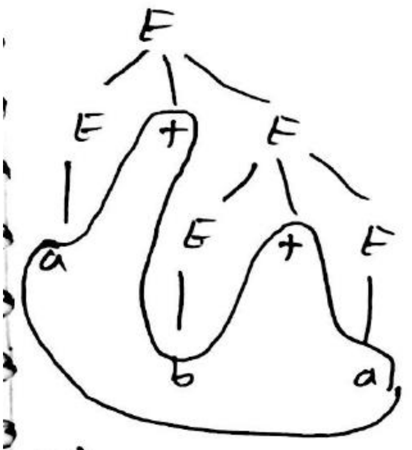
Ambiguous grammar:

A grammar, G is said to be ambiguous if there exists 2 different parse tree for atleast one string 'w' where $w \in T^*$, each parse tree with the same symbol.

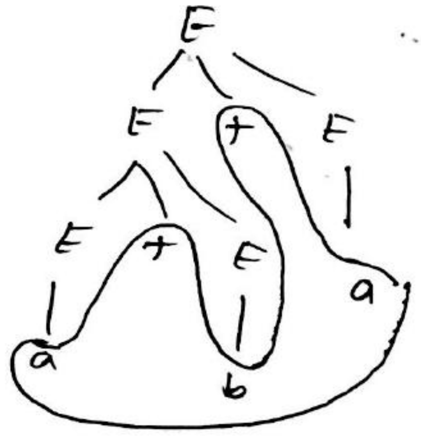
Example:

Consider the grammar given below, prove that it is ambiguous. $E \to E+E \mid a \mid b$.

Soln:



$\Rightarrow a+b+a$.



$\Rightarrow a+b+a$

Since there are 2 different parse trees with same start symbol, leads to the same string $w \in T^*$, the grammar is ambiguous.

—x—

Removing ambiguity from grammar:

The only possibility of removing ambiguity from grammar is to introduce one or more different variables.

Example 1:

Remove the ambiguity from the grammar, $E \rightarrow E+E \mid a \mid b$

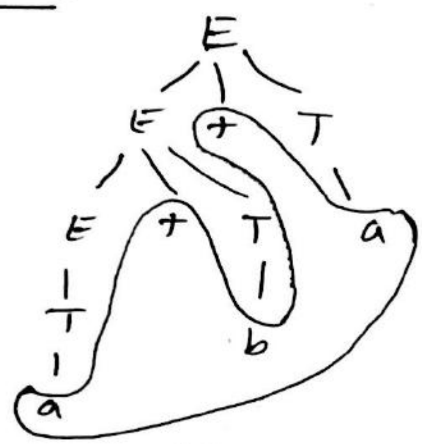
Soln:

Given Grammar, $E \rightarrow E+E \mid a \mid b$

The grammar can be rewritten by introducing a new variable T as,

$$E \rightarrow E+T \mid T$$
$$T \rightarrow a \mid b$$

Parse Tree:



—x—

2) Consider the grammar $E \rightarrow E+E \mid E*E \mid (E) \mid I$, $I \rightarrow a \mid b$

i) Show that the grammar is ambiguous

ii) Remove the ambiguity.

Soln:

i) Ambiguity Grammar

1) $E \Rightarrow E+E$

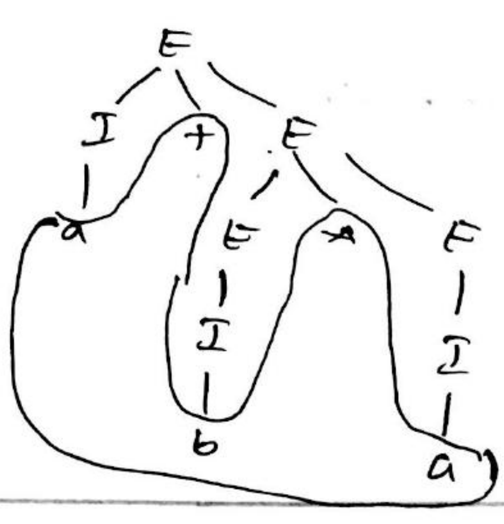
$\Rightarrow I+E$

$\Rightarrow a+E$

$\Rightarrow a+E*E$

$\Rightarrow a+I*E$

$\Rightarrow a+b*a$



2) $E \Rightarrow E * E$

$\Rightarrow E + E * E$

$\Rightarrow I + E * E$

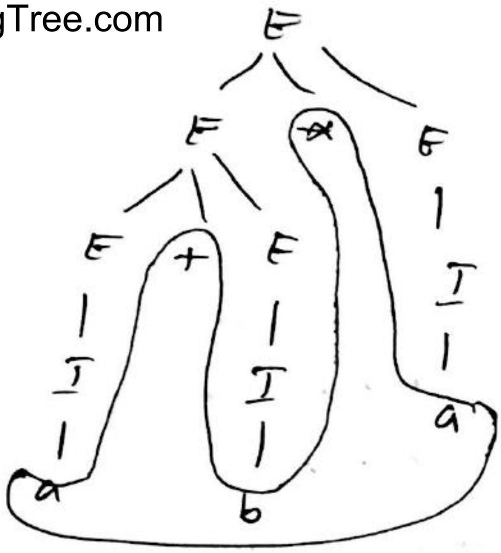
$\Rightarrow a + I * E$

$\Rightarrow a + I * E$

$\Rightarrow a + b * E$

$\Rightarrow a + b * \underline{I}$

$\Rightarrow a + b * a$



from ① & ② we have generated 2 different parse tree for the same string $a+b*a$ using the same production.

$\therefore$ The grammar is ambiguous.

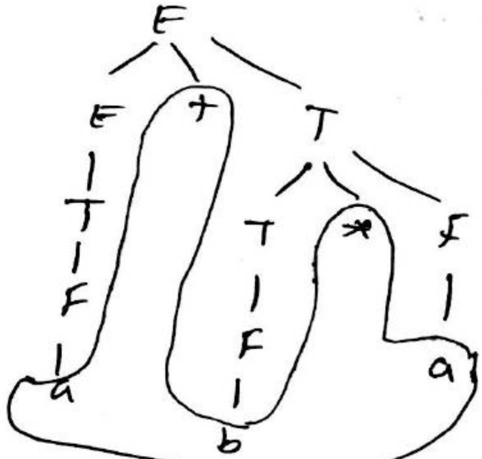
ii) <u>Removing Ambiguity</u> :

Introducing new variable T to remove ambiguity.

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid a \mid b$



— x —

1) Whether the following grammars are ambiguous.

a) $S \rightarrow 0S1S \mid 1S0S \mid \epsilon$

b) $S \rightarrow AA$

$A \rightarrow aAb \mid bAa \mid \epsilon$.

2.7 RELATIONSHIP BETWEEN DERIVATION AND DERIVATION TREE

2.7.1 FROM TREES TO DERIVATIONS

Theorem **(AU – Dec 2003, Dec 2004, May 2005, Dec 2005)**

Let the grammar, $G_l = (V, T, P, S)$ be context free. Then $S \overset{*}{\underset{G}{\Rightarrow}} \alpha$ if and only if there is a derivation tree in grammar G, that generates the string 'α'.

Proof

Let S be a variable, where $S \in V$, then $S \overset{*}{\underset{G}{\Rightarrow}} \alpha$ if and only if there is a parse tree called S-tree, starting from the root node (S) to generate the string, 'α'.

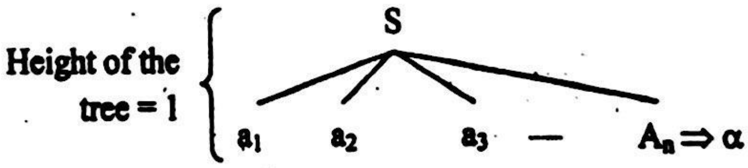
The problem can be easily proved by the principle of mathematical induction.

Basis of induction

Consider the lowest possible input value and prove the theorem given.

So we assume that there is only one interior node [height = 1] which forms the root node, that yields the string a, by deriving the leaf nodes of S as $S \rightarrow a_1\, a_2\, a_3 \ldots a_n$.

The derivation tree for the generation of $S \overset{*}{\Rightarrow} \alpha$ is given as



Thus $S \Rightarrow a_1\, a_2 \ldots a_n \Rightarrow \alpha$ is the input string generated from S.

Inductive step

Assume that the theorem is true for 'n^th' input data and we need to prove the same for n = n+1 data.

Hence we assume that the derivation tree can be drawn for (n – 1) interior nodes.

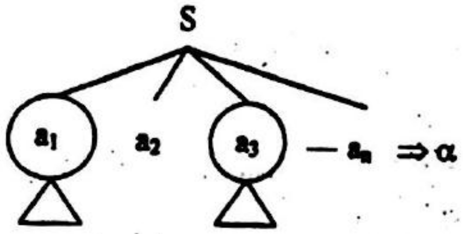
And we need to prove that the derivation tree can be drawn for 'n' interior nodes to derive 'a' from S.

Here, we need to analyze that certain nodes shall be leaf nodes, whereas others can be interior nodes.

Hence if a node $X_i \in T$, then $\alpha_i = X_i$

If the node $X_i \in V$, then $X_i \underset{\ell m}{\overset{*}{\Rightarrow}} \alpha_i$ is in G.

The derivation tree is given as, **[By inductive hypothesis]**



where,

$a_1, a_3 \in V$

$a_2, a_n \in T$

Thus, $S \Rightarrow a_1\ a_2 \ldots a_n \Rightarrow \alpha$ can be obtained.

Hence the theorem is proved by mathematical induction. —— X ——

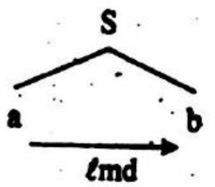
Example

Let G = (V, T, P, S) be a CFG with productions, $S \to aSb \mid ab$

Let us consider the basis of induction according to it, the lowest possible integer height of the tree should be considered.

Let the height of the tree = 1

$\therefore$ The grammar, S yields only one possible string $S \to ab$ which is given by



$\alpha = ab\ [S \Rightarrow ab]$

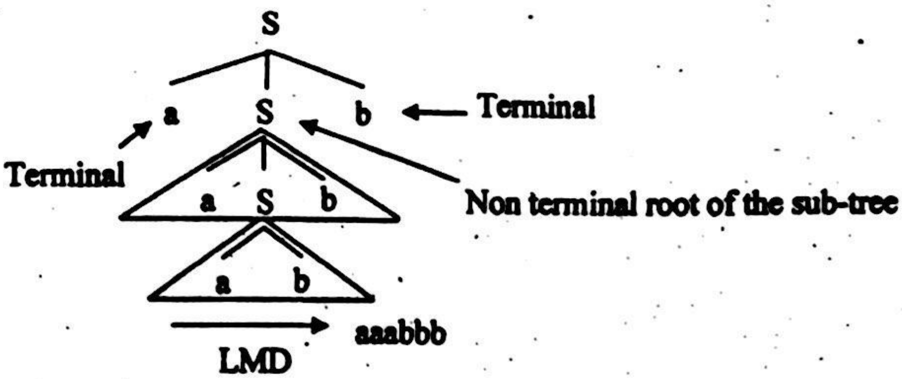
According to the inductive step let us assume that we can generate a derivation tree of height (n – 1) and we need to prove that it is true for the tree of height, n.

Let n – 1 = 2

$S \underset{\ell m}{\Rightarrow} aSb$

$$\underset{lm}{\Rightarrow} aaSbb$$

$$\underset{lm}{\Rightarrow} aaabbb$$



This is assumed.

When n = 3 [(n – 1) + 1] the derivation becomes,

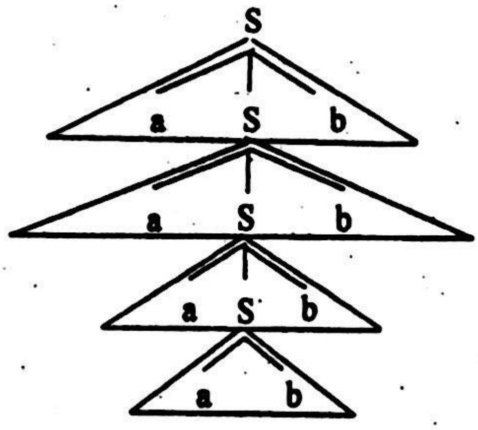
$$S \underset{lm}{\Rightarrow} aSb$$

$$\underset{lm}{\Rightarrow} aaSbb$$

$$\underset{lm}{\Rightarrow} aaaSbbb$$

$$\underset{lm}{\Rightarrow} aaaabbbb \Rightarrow \alpha$$

$$\therefore S \Rightarrow \alpha$$



$$S \Rightarrow aaaabbbb \Rightarrow \alpha$$

Thus if the theorem is true for a tree of height (n – 1), then it is true for height n.

2.7.2 From Derivation To Recursive Inferences

Theorem

Let G = (V, T, P, S) be a CFG and if there is a derivation $A \overset{*}{\underset{lm}{\Rightarrow}} \alpha$, where α is in T^*. Then the recursive inference procedure applied to G determines that α is in the language of non-terminal 'A'.

Proof

We shall prove the theorem by the principle of mathematical induction.

Basis of induction

Let us consider the lowest possible input of the grammar.

Let $A \Rightarrow \alpha$, then there must be a production $A \rightarrow \alpha$ in G. then we can infer that α is in the language of variable A.

Inductive step

Assume that the theorem holds for the derivation has (n-1) number of steps and we need to prove that it holds for 'n' no. of steps.

Let the derivation be stated as,

$$A \Rightarrow X_1 X_2 X_3 \ldots\ldots X_k$$

$$\overset{*}{\Rightarrow} \alpha$$

Then α can be broken as

$$\alpha = \alpha_1 \alpha_2 \ldots \alpha_k$$

where,

$$X_i \overset{*}{\Rightarrow} \alpha_i$$

The derivation $X \overset{*}{\Rightarrow} \omega_i$ can take at most 'n-1' number of steps to generate.

Now we have a production,

$$A \rightarrow X_1 X_2 \ldots . X_k$$

Infer α_i to be in the language of X_i

$\therefore \alpha_1 \alpha_2 \ldots . \alpha_k$ is inferred as in the language of A.

3

Leftmost derivation of a tree yields string

Theorem

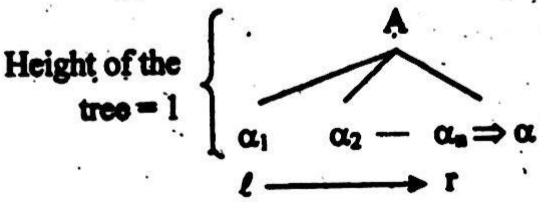
Let G = (V, T, P, S) be a CFG. Suppose there is a parse tree with root labeled by A | A $\in$ V and with yield α where $\alpha \in T^*$. Then there is a leftmost derivation $A \underset{lm}{\overset{*}{\Rightarrow}} \alpha$ in G.

Proof

The theorem is proved by mathematical induction on the height of the parse tree.

Basis of induction

Let the height of the tree be 1, the tree looks like,



Here the root is labeled A, A$\in$V and the children are the leaf nodes $\alpha_1, \alpha_2, \ldots .\alpha_n$ which are read from the left to right to generate α.

Thus $A \underset{lm}{\Rightarrow} \alpha$ is proved.

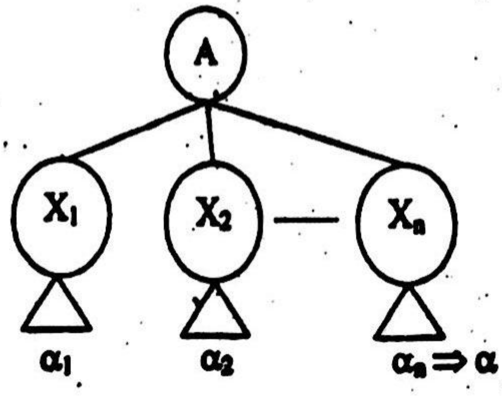
Inductive step

We assume the height of the tree as n | n>1, then $A \underset{lm}{\overset{*}{\Rightarrow}} \alpha$ stands.

We need to prove the same for n = n+1.

There are two issues:

The child of A can be terminal / variable as given



1. If $A_i \in T$, then $X_i \Rightarrow \alpha_i$ [$\omega_i \rightarrow$ terminal]

2. If $A_i \in V$, then there must be some leftmost derivation, $A_i \overset{*}{\Rightarrow} \alpha_i$

For n nodes, $\alpha_1 = \alpha_1 \alpha_2 \alpha_3 .....a_n$

$$A \overset{*}{\underset{lm}{\Rightarrow}} A_1 A_2 A_3 .... A_n \quad \text{(for } i = 1,2,3,.....n)$$

$$\overset{*}{\underset{lm}{\Rightarrow}} \alpha_1 \alpha_2 .....a_{i-1}\ A_i\ A_{i+1}\ A_{i+2} .... A_n$$

If A_i is a terminal, then

$$A_i \overset{*}{\underset{lm}{\Rightarrow}} \alpha_1 \alpha_2 .....a_i\ A_{i+1}\ A_{i+2} .... A_n$$

If A_i is a variable, then

$$A \underset{lm}{\Rightarrow} \beta_1 \underset{lm}{\Rightarrow} \beta_2 \underset{lm}{\Rightarrow} ..... \underset{lm}{\Rightarrow} \alpha_1$$

and

$$A \Rightarrow \alpha_1 \alpha_2 .....a_{i-1}\ A_i\ A_{i+1} .... A_n$$

$$\Rightarrow \alpha_1 \alpha_2, ...\beta_1$$

$$A_i\ A_{i+1} ...... A_n \overset{*}{\underset{lm}{\Rightarrow}} \alpha_1 \alpha_2 \beta_2 A_i$$

$$A_{i+1} .... A_n \overset{*}{\underset{lm}{\Rightarrow}} \alpha_1 \alpha_2, \alpha_3 .....\alpha_i\ A_{i+1}\ A_{i+2} .... A_n$$

Thus,

$$A \overset{*}{\underset{lm}{\Rightarrow}} \alpha_1 \alpha_2, ...\alpha_i\ A_{i+1} . A_{i+2} .... A_n$$

$$\overset{*}{\underset{lm}{\Rightarrow}} \omega$$

Thus the theorem has been proved.

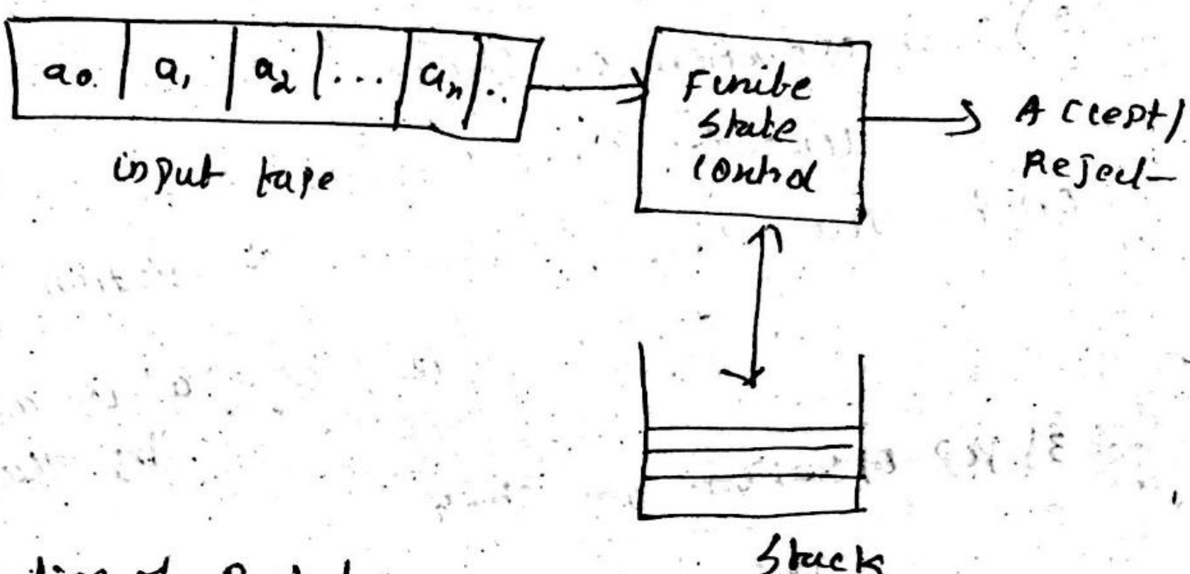
2.8 SIMPLIFICATION OF CFG

- The need for simplifying CFG is to make it easier to analyze and prove facts about CFL.

PUSHDOWN AUTOMATA

Pushdown Automata:

A Pushdown automata shortly called as a PDA is a ϵ-NFA with stack, which is used to define regular language.



Definition of Pushdown Automata:

A PDA can be formally defined by 7-tuples.

$$P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

where,

Q - Finite set of states.

Σ - Finite set of input symbols

Γ - Finite set of stack symbols

δ - Transition function

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

q_0 - Start state

z_0 - Start symbol (Stack)

F - accepting state.

Transitions with stack operations

1) Read input with no-operation on stack:

The transition that reads an input from the input tape, but performs no-operations on the stack is given as,

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

2) Push operation on stack:

consider that the input = 'a' pushed on to the stack. Then the transition becomes,

$$\delta(q_1, a, b) = (q_2, ab)$$ - 'a' is accepted by the stack.

3) Pop operation on stack:

The transition of a pop operation is given as, $\delta(q_1, a, b) = (q_2, \varepsilon)$ 'a' cancels 'b' from the stack.

Example:

1) $L = \{ w \in (a,b)^* \mid w$ is of the form $a^n b^n, n \geq 1 \}$.

Soln:

δ:

$$\delta(q_0, a, z_0) = (q_0, a z_0)$$
$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, a) = (q_1, \varepsilon)$$
$$\delta(q_1, b, a) = (q_1, \varepsilon)$$
$$\delta(q_1, \varepsilon, z_0) = (q_2, z_0)$$

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

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

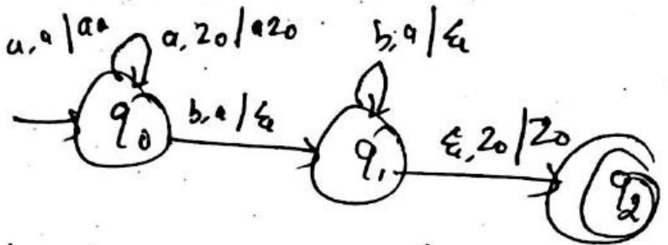
$\Sigma = \{a, b\}$

$\Gamma = \{z_0, a, b\}$

$q_0 = \{q_0\}$

$z_0 = \{z_0\}$

$F = \{q_2\}$

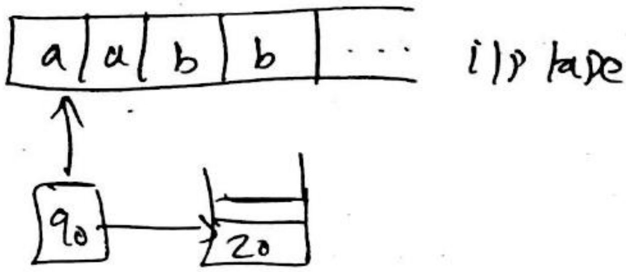
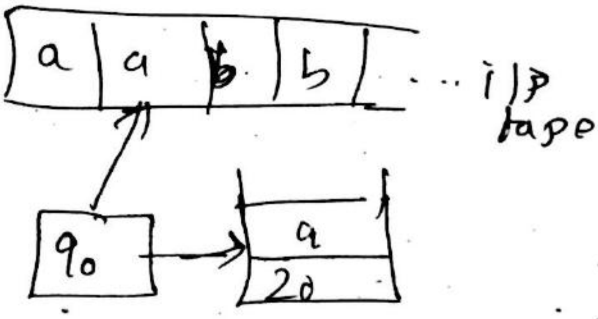
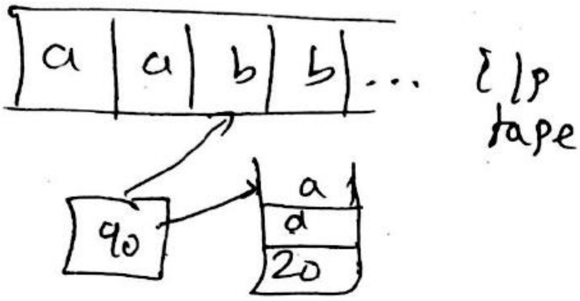
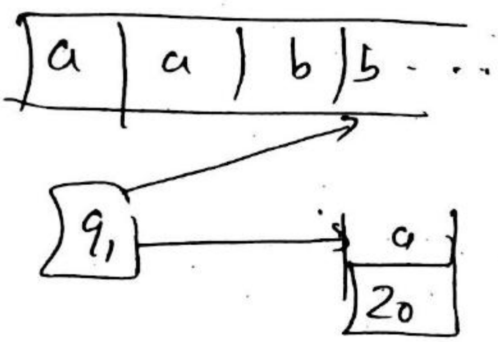
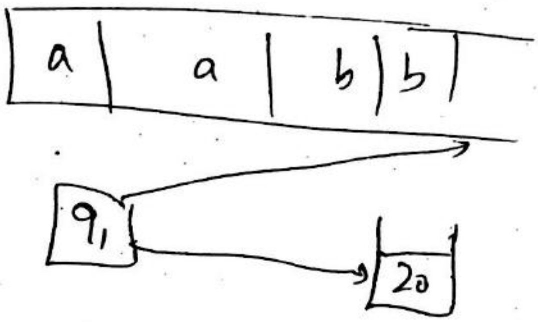
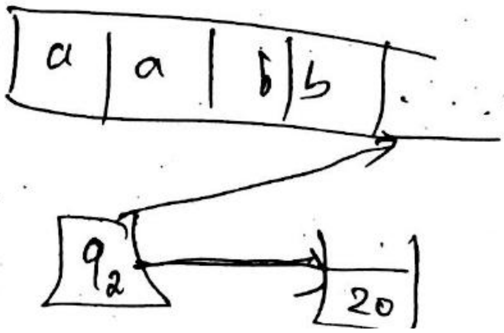


Instantaneous Description of a PDA:

* Instantaneous description of a PDA is an informal notation or description of a PDA.
* It is a pictorial / diagrammatic representation of a string processed by a PDA.
* This is used to depict the processing of a string by a PDA.

Ex:

$L = \{a^n b^n \mid n \geq 1\}$.

Graphical representation (Instantaneous description)	Tuple format
1) 	$(q_0, aabb, Z_0)$
2) 	(q_0, abb, aZ_0)
3) 	(q_0, bb, aa)
4) 	(q_1, b, aZ_0)
5) 	(q_1, ϵ, Z_0)
6) 	(q_2, ϵ, Z_0)

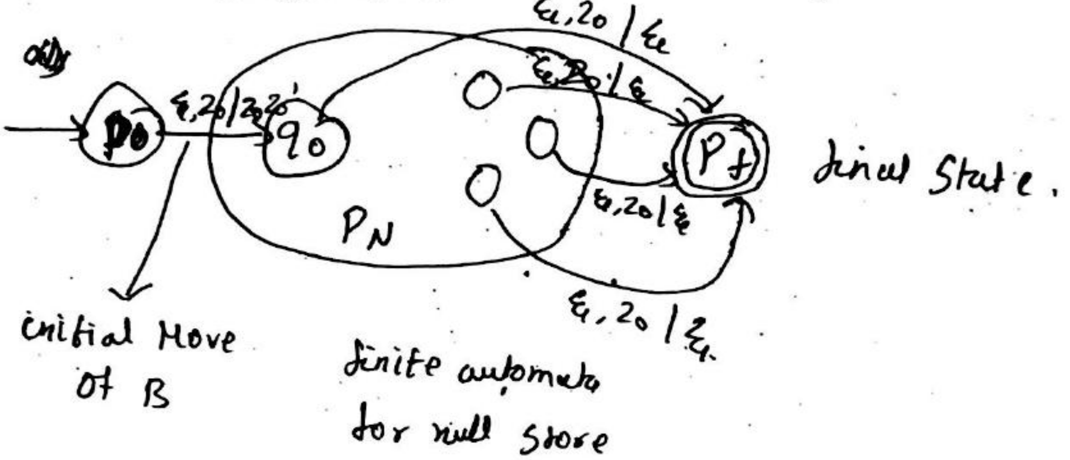
Equivalance of acceptance of PDA from empty stack to Final State

Theorem:

If $A = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ is a pda accepting L by empty store, we can find a pda $B = (Q', \Sigma, \Gamma', \delta_B, q_0', z_0', F')$ which accepts L by final state (i.e) $L = N(A) = L(B)$

Proof:

B is constructed in such a way that



Let us define B as follows:

$$B = (Q', \Sigma, \Gamma', \delta_B, q_0', z_0', F)$$

Where

$Q' - Q \cup \{p_0, p_f\}$

$\Gamma' - \Gamma \cup \{z_0'\}$

$F' - \{p_f\}$

$q_0' = p_0$

$z_0' =$ Start symbol for stack

δ_B is given by rules

$R_1 : \delta_B(p_0, \epsilon, z_0') = (q_0, z_0 z_0')$

$R_2 : \delta_B(q, a, z) = \delta(q, a, z)$

$R_3 : \delta_B(q, \epsilon, z_0') = (p_f, \epsilon)$

we have to show $N(A) = L(B)$

Let $w \in N(A)$. Then by definition of $N(A)$.

$(q_0, w, z_0) \overset{*}{\underset{A}{\vdash}} (q, \epsilon, \epsilon)$

By previous theorem

$(q, x, \alpha) \vdash^{*} (p, y, \beta)$

we get

$(q_0, w, z_0 z_0') \overset{*}{\underset{A}{\vdash}} (q, \epsilon, z_0')$

Since null store (or) empty store is a subset of δ_B

$\therefore$ we conclude that

$(p_0, w, z_0') \overset{*}{\underset{B}{\vdash}} (q_0, w, z_0 z_0')$

$\overset{*}{\underset{B}{\vdash}} (q, \epsilon, z_0')$

$\overset{*}{\underset{B}{\vdash}} (p_f, \epsilon, \epsilon)$

$\therefore \boxed{N(A) = L(B)}$

1) Construct a PDA for $\{a^n b^m a^{m+n}\}$

2) Construct a PDA for $\{a^m b^m c^n\}$

2) Convert the grammar $S \to aSb \mid A$, $A \to bSa \mid S \mid \epsilon$ to a PDA that accepts the same language by empty stack. Check whether the string aababb is accept or not.

3) Convert the grammar $S \to aSb \mid aAb$, $A \to bAa \mid ba$ to PDA. Check whether the string abbaab is accept or not.

2) Convert the grammar $S \to 0S1 \mid A$, $A \to 1A0 \mid S \mid \epsilon$ into PDA that accepts the same language by empty stack. Check whether 0101 belongs to $N(M)$.

3) Let $P = (\{p, q\}, \{0, 1\}, \{X, Z_0\}, \delta, q, Z_0)$ where δ is given by

$\delta(q, 1, Z_0) = \{(q, XZ_0)\}$
$\delta(q, 1, X) = \{(q, XX)\}$
$\delta(q, 0, X) = \{(p, X)\}$
$\delta(q, \epsilon, X) = \{(q, \epsilon)\}$
$\delta(p, 1, X) = \{(p, \epsilon)\}$
$\delta(p, 0, Z_0) = \{(q, Z_0)\}$

2)
$\delta(q_0, b, Z_0) = \{(q_0, ZZ_0)\}$
$\delta(q_0, \lambda, Z_0) = \{(q_0, \lambda)\}$
$\delta(q_0, \lambda, Z_0) = \{(q_0, ZZ)\}$
$\delta(q_0, b, Z) = \{(q_0, ZZ)\}$
$\delta(q_0, a, Z) = \{(q_1, Z)\}$
$\delta(q_1, b, Z) = \{(q_1, \lambda)\}$
$\delta(q_1, a, Z_0) = \{(q_0, Z_0)\}$

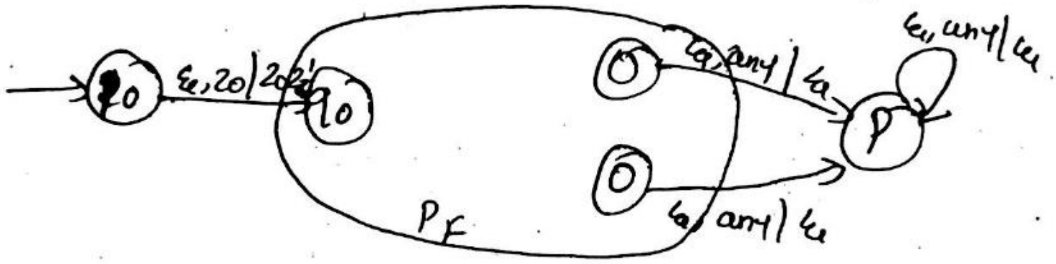
Equivalance of acceptance of PDA from final state to empty stack:

Theorem:

If $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ accepts L by final state, we can find PDA B, accepting L by empty stack.
(ie) $L = L(A) = N(B)$

Proof:

B is constructed from A in such a way that



finite automata
for final state acceptance

B is as follows:

$$B = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{Z_0'\}, \delta_B, p_0, Z_0', \phi)$$

δ_B is defined by R_1, R_2, R_3 & R_4 as:

R_1 : $\delta_B(p_0, \epsilon, Z_0') = (q_0, Z_0 Z_0')$

R_2 : $\delta_B(p, \epsilon, Z) = (p, \epsilon)$

R_3 : $\delta_B(q, a, Z) = \delta(q, a, Z)$

R_4 : $\delta_B(p, \lambda, Z) = \delta(q, \lambda, Z)$

We have to show that $L(A) = N(B)$. Then

$(q_0, w, Z_0) \vdash_A^* (q, \lambda, \alpha) \longrightarrow (1)$

Since $\delta_B \subseteq \delta$ and by previous theorem

$$(q, x, \alpha) \vdash^{*} (p, y, \beta)$$

we can write (1) has

$$(q_0, w, Z_0 Z_0') \vdash^{*}_{B} (q, \epsilon, \alpha Z_0')$$

Then B can be computed has

$$(p_0, w, Z_0') \vdash_{B} (q_0, w, Z_0 Z_0') \vdash^{*}_{B} (q, \lambda, \alpha Z_0') \vdash^{*}_{B} (p, \epsilon, \epsilon)$$

Thus

$$\boxed{L = N(B) = L(A)}$$

The language of PDA:

The language of a PDA can be accepted of two ways.

* Acceptance by final state
* Acceptance by empty stack

Acceptance By Final state:

A PDA that accepts its input and enters the final/accepting state is called as a PDA accepted by final state.

$$L(P) = \{ w \mid (q_0, w, z_0) \vdash_P^* (q, \varepsilon, \alpha) \}$$

Final state   No i/p   Stack symbols

Acceptance By Empty stack:

The PDA that accepts its input by emptying the stack is the PDA accepted by empty stack.

$$N(P) = \{ w \mid (q_0, w, z_0) \vdash^* (q, \varepsilon, \alpha) \}$$

Final state   No i/p   Stack empty

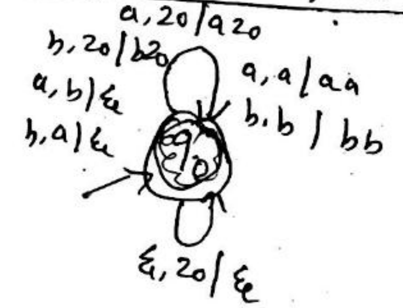
Example:

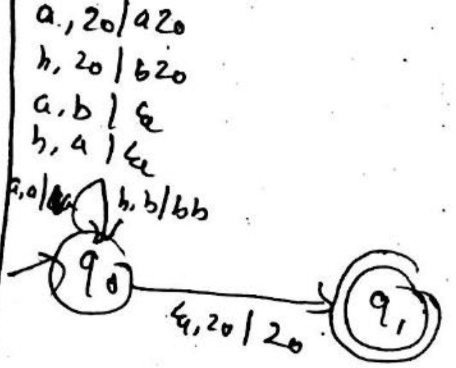
1) Design a PDA that accepts a set of strings over $\{a, b\}$ with equal number of a's and b's.

Soln:

PDA accepting through an empty stack	PDA accepting through final state
Transition function	
$\delta(q_0, a, z_0) = (q_0, az_0)$	$\delta(q_0, a, z_0) = (q_0, az_0)$
$\delta(q_0, b, z_0) = (q_0, bz_0)$	$\delta(q_0, b, z_0) = (q_0, bz_0)$
$\delta(q_0, a, b) = (q_0, \epsilon)$	$\delta(q_0, a, b) = (q_0, \epsilon)$
$\delta(q_0, b, a) = (q_0, \epsilon)$	$\delta(q_0, b, a) = (q_0, \epsilon)$
$\delta(q_0, a, a) = (q_0, aa)$	$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, b) = (q_0, bb)$	$\delta(q_0, b, b) = (q_0, bb)$
$\delta(q_0, \epsilon, z_0) = (q_0, \epsilon)$	$\delta(q_0, \epsilon, z_0) = (q_1, z_0)$

Transition diagram





PDA:

PDA accepting through an empty stack:

$M = (\{q_0\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \phi)$

PDA accepting through final state:

$M = (\{q_0, q_1\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \{q_1\})$

2) Let $L = \{a^n b^n c^m d^m \mid n, m \geq 1\}$ Find a PDA for L.

Soln:

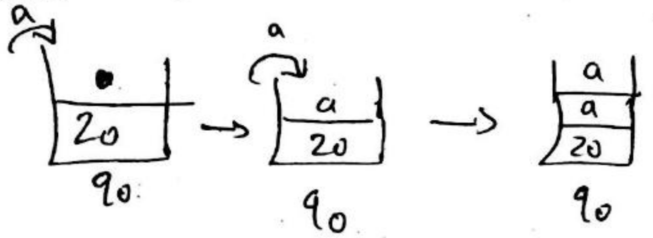
Transition Function:

Step 1:

$w = aabbcd$, $n=2$, $m=1$

Push 'a' onto the stack

$w = aabbcd$

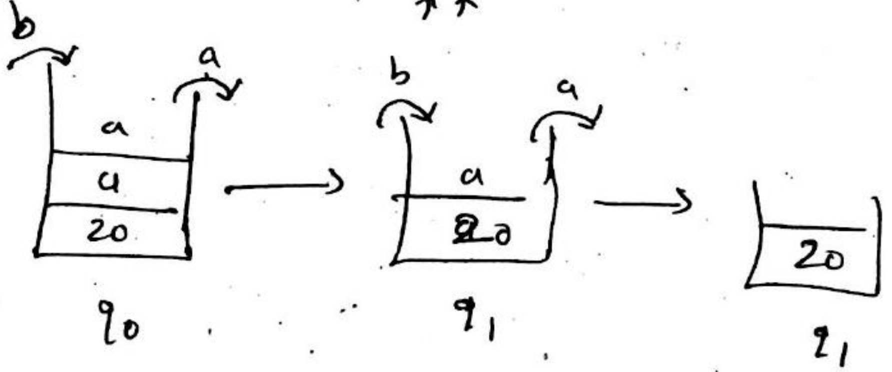


$\delta[q_0, a, Z_0] = (q_0, aZ_0)$

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

Step 2: For every 'b' as input, 'a' should be popped out.

$w = aabbcd$

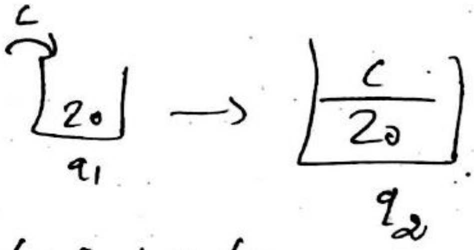


$\delta[q_0, b, a] = (q_1, \epsilon)$

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

Step 3: Push 'c' onto the stack.

w = aabbcd

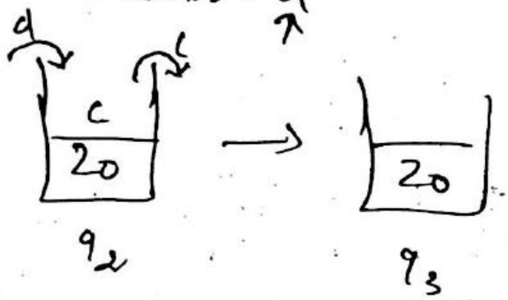


$\delta(q_1, c, z_0) = (q_2, cz_0)$

$\delta(q_2, c, c) = (q_2, cc)$

Step 4: For every 'd' as input 'c' should be popped out.

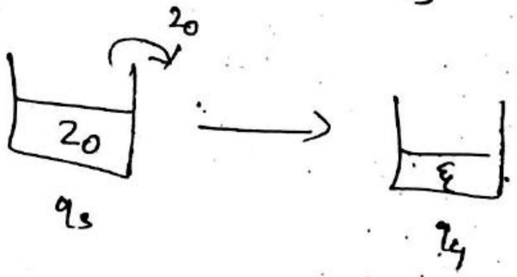
w = aabbcd



$\delta(q_2, d, c) = (q_3, \varepsilon)$

$\delta(q_3, d, \cancel{c}) = (q_3, \varepsilon)$

Step 5: String is accepted as



$\delta(q_3, \varepsilon, z_0) = (q_4, \varepsilon)$

The PDA is given as

$M = (\{q_1, q_2, q_3, q_4\}, \{a, b, c, d\}, \{\cancel{a,b,c,z_0}\}, \{a, c, z_0\}, \delta, q_0, z_0, \phi)$

3) let $L = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ \& } i+j = k\}$. Provide the transition function.

i) Accepted by final state

ii) Accepted by empty stack.

Soln:

PDA: Through final state	PDA: Through empty stack
δ:	δ:
$\delta(q_0, a, z_0) = (q_0, xz_0)$	$\delta(q_0, a, z_0) = (q_0, xz_0)$
$\delta(q_0, a, x) = (q_0, xx)$	$\delta(q_0, a, x) = (q_0, xx)$
$\delta(q_0, b, z_0) = (q_1, xz_0)$	$\delta(q_0, b, z_0) = (q_1, xz_0)$
$\delta(q_0, b, x) = (q_1, xx)$	$\delta(q_0, b, x) = (q_1, xx)$
$\delta(q_1, b, x) = (q_1, xx)$	$\delta(q_1, b, x) = (q_1, xx)$
$\delta(q_1, c, x) = (q_2, \epsilon)$	$\delta(q_1, c, x) = (q_2, \epsilon)$
$\delta(q_2, c, x) = (q_2, \epsilon)$	$\delta(q_2, c, x) = (q_2, \epsilon)$
$\delta(q_2, \epsilon, z_0) = (q_3, z_0)$	$\delta(q_2, \epsilon, z_0) = (q_2, \epsilon)$
$\delta(q_0, \epsilon, z_0) = (q_3, z_0)$	$\delta(q_0, \epsilon, z_0) = (q_0, \epsilon)$

3) Design a PDA for the language $\{a^m b^n c^m \mid m,n \geq 1\}$ using empty stack. (oa).

Soln:

$$w = \underbrace{aa}_{\text{Push}}\ \overset{\text{NO-OP}}{b}\ \underbrace{cc}_{\text{Pop}}$$

δ:

$$\left.\begin{aligned} \delta(q_0, a, Z_0) &= (q_0, aZ_0) \\ \delta(q_0, a, a) &= (q_0, aa) \end{aligned}\right\} \text{Push}$$

$$\left.\begin{aligned} \delta(q_0, b, a) &= (q_1, a) \\ \delta(q_1, b, a) &= (q_1, a) \end{aligned}\right\} \text{NO-OP}$$

$$\left.\begin{aligned} \delta(q_1, c, a) &= (q_2, \varepsilon) \\ \delta(q_2, c, a) &= (q_2, \varepsilon) \end{aligned}\right\} \text{Pop}$$

$$\delta(q_2, \varepsilon, Z_0) = (q_2, \varepsilon) \rightarrow \text{empty stack.}$$

$$M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{Z_0, a\}, \delta, \{q_0\}, \{Z_0\}, \phi)$$

—x—

4) Construct a PDA for the language accepting by empty stack. $L = \{a^m b^m c^n \mid m,n \geq 1\}$

Soln:

$$w = \underbrace{aa}_{\text{Push}}\ \underbrace{bb}_{\text{Pop}}\ \underbrace{c}_{\text{No-op}}$$

From CFG's to PDA's:

Theorem:

For any context free language L, there exist an PDA M such that $L = L(M)$

Proof:

Let $G = (V, T, P, S)$ be a grammar. There exists a GNF then we can construct PDA which simulates left most derivations in this grammar.

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

$Q = \{q_0, q, q_f\}$ set of states

Σ = terminals of grammar G

$\Gamma = V \cup \{Z\}$

$F = \{q_f\}$ final state.

The transition function will include

$\delta(q_0, \varepsilon, Z) = (q, SZ)$, so that after first move of M, the stack contains S.

In addition, the set of transition rules

$\delta(q, \varepsilon, A) = \{(q, \alpha)\}$

for each $A \rightarrow \alpha$ in P

$\delta(q, a, a) = \{(q, \varepsilon)\}$

for each $a \in \Sigma$

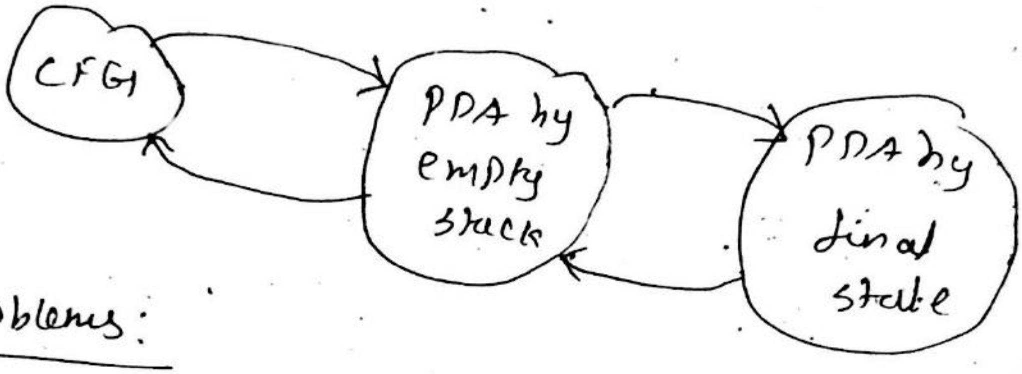
Equivalance of PDA and CFG:

7

For a given CFG, $G = (V, T, P, S)$, we can construct a PDA, M that simulates the left-most derivation of G.

The PDA accepting L(G) by empty stack is given by

$$M = (\{q\}, T, V \cup T, \delta, q, S, \phi)$$



Problems:

1) Find a PDA for the given grammar $S \to 0S1 \mid 00 \mid 11$

Soln:

$$\delta(q, \varepsilon, S) = \{(q, 0S1), (q, 00), (q, 11)\}$$

$$\delta(q, 0, 0) = \{(q, \varepsilon)\}$$

$$\delta(q, 1, 1) = \{(q, \varepsilon)\}$$

$$M = (\{q\}\{0,1\}, \{0,1,S\} \delta, q, S, \phi)$$

String [000011].

2) construct a PDA for the grammar

$S \to aB \mid bA$, $A \to a \mid aS \mid bAA$, $B \to b \mid bS \mid aBB$

Soln:

where δ is given by

$\delta(q, \varepsilon, S) = \{(q, aB), (q, bA)\}$

$\delta(q, \varepsilon, A) = \{(q, a), (q, aS), (q, bAA)\}$

$\delta(q, \varepsilon, B) = \{(q, b), (q, bS), (q, aBB)\}$

$\delta(q, a, a) = \{(q, \varepsilon)\}$

$\delta(q, b, b) = \{(q, \varepsilon)\}$

$M = (\{q\}, \{a, b\}, \{a, b, S, A, B\}, \delta, q, S, \phi)$

3) Construct a PDA for $S \to aAA$, $A \to aS \mid bS \mid a$.

4) construct a PDA for $S \to aSbb \mid ab$.

5) construct a PDA for $G = (\{S, A\}, \{a, b\}, P, S)$ where P: $S \to AA \mid a$, $A \to SA \mid b$.

6. Let G be the grammar given by $S \to 0BB$, $B \to 0S \mid 1S \mid 0$. Construct a PDA. Test if 010^4 is in the language.

Soln:

δ: $\delta(q, \varepsilon, S) = \{(q, 0BB)\}$

$\delta(q, \varepsilon, B) = \{(q, 0S), (q, 1S), (q, 0)\}$

$\delta(q, 0, 0) = (q, \varepsilon)$, $\delta(q, 1, 1) = (q, \varepsilon)$

$M = (\{q\}, \{0, 1\}, \{0, 1, S, B\}, \delta, q, S, \phi)$

machine of 0104 by M.

$\delta(q, 010000, S) \vdash (q, \underline{0}10000, \underline{0}BB)$

$\vdash (q, 10000, BB)$

$\vdash (q, 10000, 1SB)$

$\vdash (q, 0000, SB)$

$\vdash (q, 0000, 0BBB)$

$\vdash (q, 000, BBB)$

$\vdash (q, 000, 0BB)$

$\vdash (q, 00, BB)$

$\vdash (q, 00, 0B)$

$\vdash (q, 0, B)$

$\vdash (q, 0, 0)$

$\vdash (q, \varepsilon)$

∴ the string 010^4 is accepted by MEL.

From PDA'S to CFG'S:

Theorem:

If L is $N(M)$ for some PDA M, then L is CFL.

Proof:

1. It has single final state q_f iff the stack is empty.
2. All transitions must have the form.

$\delta[q_i, a, A) = \{c_1, c_2, \cdots c_n\}$, where

$\delta[q_i, a, A] = [q_j, \varepsilon) \longrightarrow (1)$

$\delta[q_i, a, A) = (q_j, BC) \longrightarrow (2)$

Given $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, \{q_f\})$ statisfies the condition (1) & (2)

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

$T = \Sigma$

V – elements of the form $[q, A, p]$, q & p in Q and A is Γ

S — Start symbol

$S \to [q_0, z_0, q]$ for each q in Q

P consist of: $u, v \in \Sigma^*$

$A, x \in \Gamma^*$, $q_i, q_j \in Q$.

$(q_i, uv, Ax) \vdash^* (q_f, v, x)$ implies $(q_i, u, A) \to u$

Consider (1) $(q_i, A, q_k) \to a(q_j, B, q_1)(q_1, C, q_k)$ the corresponding transition for PDA is $\delta(q_i, a, A) = \{(q_j, BC), \ldots\}$

similarly if (2) $(q_i, A, q_j) \to a$ then the transition is $\delta(q_i, a, A) = \{(q_j, \varepsilon)\}$

The conclusion is

$(q_0, w, z_0) \vdash^* (q_f, \varepsilon, \varepsilon)$ is true iff $[q_0, z_0, q_f) \overset{*}{\Rightarrow} w$

consequently $L(M) = L(G)$

Convert PDA to CFG.

$M = (\{p, q\}, \{0,1\}, \{x, z\}, \delta, q, z, \phi)$

δ: $\delta(q, 1, z) = (q, xz)$

$\delta(q, 1, x) = (q, xx)$

$\delta(q, \varepsilon, x) = (q, \varepsilon)$

$\delta(q, 0, x) = (p, x)$

$\delta(p, 1, x) = (p, \varepsilon)$

$\delta(p, 0, z) = (q, z)$

Soln:

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

$V = \{ \overset{A}{[q, x, q]}, \overset{B}{[q, x, p]}\ \overset{c}{[p, x, q]}\ \overset{d}{[p\ x\ p]}$

$\underset{e}{[q\ z\ q]}\ \underset{f}{[q\ z\ p]}\ \underset{g}{[p\ z\ q]}\ \underset{h}{[p\ z\ p]} \}$

$T = \{0, 1\}$

$S = \{S\}$

$S \rightarrow E \mid F$

$E \rightarrow 1AE \mid 1BG$

~~$[p\ x\ q]$~~

P:

1) $P_1: S \rightarrow [q\ z\ q]$

$P_2: S \rightarrow [q\ z\ p]$

2) $\delta(q, 1, z) = (q, xz)$

$[q\ z\ q] \rightarrow 1\ [q\ x\ q]\ [q\ z\ q]$

$[q\ z\ q] \rightarrow 1\ [q\ x\ p]\ [p\ z\ q]$

$[q\ z\ p] \rightarrow 1\ [q\ x\ q]\ [q\ z\ p]$

$[q\ z\ p] \rightarrow 1\ [q\ x\ p]\ [p\ z\ p]$

3) $\delta(q, 1, x) = (q, xx)$

$[q\,x\,q] \rightarrow 1\,[q\,x\,q]\,[q\,x\,q]$

$[q\,x\,q] \rightarrow 1\,[q\,x\,p]\,[p\,x\,q]$

$[q\,x\,p] \rightarrow 1\,[q\,x\,q]\,[q\,x\,p]$

$[q\,x\,p] \rightarrow 1\,[q\,x\,p]\,[q\,x\,p]$

4) $\delta(q, \epsilon, x) = (q, \epsilon)$

$[q\,x\,q] \rightarrow \epsilon$

5) $\delta(q, 0, x) = (p, x)$

$[q\,x\,q] \rightarrow 0\,[p\,x\,q]$

$[q\,x\,p] \rightarrow 0\,[p\,x\,p]$

6) $\delta(p, 1, x) = (p, \epsilon)$

$[p\,x\,p] \rightarrow 1$

7) $\delta(p, 0, z) = (q, z)$

$[p\,z\,q] \rightarrow 0\,[q\,z\,q]$

$[p\,z\,p] \rightarrow 0\,[q\,z\,p]$

<u>final production:</u>

2) Give CFG for the PDA,

$M = (\{q_0, q_1\}, \{0, 1\}, \{z_0, x\}, \delta, q_0, z_0, \phi)$

$\delta(q_0, 1, z_0) = (q_0, xz_0)$

$\delta(q_0, 1, x) = (q_0, xx)$

$\delta(q_0, 0, x) = (q_1, x)$

$\delta(q_0, \varepsilon, z_0) = (q_0, \varepsilon)$

$\delta(q_1, 1, x) = (q_1, \varepsilon)$

$\delta(q_1, 0, z_0) = (q_0, z_0)$

Soln:

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

$V = \{[q_0 z_0 q_0]\ [q_0 z_0 q_1]\ [q_1 z_0 q_0]\ [q_1 z_0 q_1]$
$[q_0 x q_0]\ [q_0 x q_1]\ [q_1 x q_0]\ [q_1 x q_1]\}$

$T = \{0, 1\}$

$S = \{S\}$.

P:

1) $S \rightarrow [q_0 z_0 q_0]$

$S \rightarrow [q_0 z_0 q_1]$ ×

$q_0 z_0 q_1$

$q_0 x q_1$

2) $\delta(q_0, 1, z_0) = (q_0, xz_0)$

$[q_0 z_0 q_0] \rightarrow 1 [q_0 x q_0][q_0 z_0 q_0]$

$[q_0 z_0 q_0] \rightarrow 1 [q_0 x q_1][q_1 z_0 q_0]$ ×

$[q_0 z_0 q_1] \rightarrow 1 [q_0 x q_0][q_0 z_0 q_1]$ ×

$[q_0 z_0 q_1] \rightarrow 1 [q_0 x q_1][q_1 z_0 q_1]$ ×

3) $\delta(q_0, 1, x) = (q_0, xx)$

$[q_0 x q_0] \rightarrow 1 [q_0 x q_0][q_0 x q_0]$

$[q_0 x q_0] \rightarrow 1 [q_0 x q_1][q_1 x q_0] \times$

$[q_0 x q_1] \rightarrow 1 [q_0 x q_0][q_0 x q_1] \times$

$[q_0 x q_1] \rightarrow 1 [q_0 x q_1][q_1 x q_1] \times$

4) $\delta(q_0, 0, x) = (q_1, x)$

$[q_0 x q_0] \rightarrow 0 [q_1 x q_0]$

$[q_0 x q_1] \rightarrow 0 [q_1 x q_1] \times$

5) $\delta(q_0, \varepsilon, z_0) = (q_0, \varepsilon)$

$[q_0, z_0 q_0] \rightarrow \varepsilon$

6) $\delta(q_1, 1, x) = (q_1, \varepsilon)$

$[q_1 x q_0] \rightarrow 1$

7) $\delta(q_1, 0, z_0) = (q_0, z_0)$

$[q_1 z_0 q_0] \rightarrow 0 [q_0 z_0 q_0]$

$[q_1 z_0 q_1] \rightarrow 0 [q_0 z_0 q_1] \times$

Final production:

$S \rightarrow [q_0 z_0 q_0]$

$[q_0 z_0 q_0] \rightarrow 1 [q_0 x q_0][q_0 z_0 q_0]$

$[q_0 x q_0] \rightarrow 1 [q_0 x q_0][q_0 x q_0]$

$[q_0 x q_0] \rightarrow 0 [q_1 x q_0]$

$[q_0 z_0 q_0] \rightarrow \varepsilon$

$[q_1 x q_0] \rightarrow 1$

$[q_1 z_0 q_0] \rightarrow 0 [q_0 z_0 q_0]$

UNIT IV PROPERTIES OF CONTEXT FREE LANGUAGES

Normal Forms for CFG – Pumping Lemma for CFL – Closure Properties of CFL – Turing Machines – Programming Techniques for TM.

Simplification of CFG:

The preliminary simplifications, which are applied on grammars to convert them to normal forms are,

* Elimination of useless symbols
* Elimination of ε - Productions
* Elimination of unit productions

1. Elimination of useless symbols:

Useless symbols:

1. The useless symbols are those variables/terminals that do not appear in any derivation of a terminal string from the start string.

2. If the derivation from the start symbol to a string of terminals does not depend on a variable x then x is non-reachable symbol.

Elimination:

1. The only way to simplify a grammar containing non-generating symbol is to eliminate such symbols directly from the grammar.

2. A grammar containing non reachable symbol can be simplified by deleting all the production containing the non-reachable symbol.

Example1:

Consider the grammar {$S \to aA \mid bB \mid a \mid b$, $A \to Aa \mid a$, $B \to bB$}. eliminate useless symbols.

Soln:

$B \to bB$ is a recursive grammar that substitution to S resulting is endless

$$S \Rightarrow bB \Rightarrow bbB \Rightarrow bbbB \Rightarrow \ldots$$

So eliminating the production for $B \to bB$

$$\boxed{\begin{array}{l} S \to aA \mid a \mid b \\ A \to Aa \mid a \end{array}}$$

—x—

2) Consider the grammar {$S \to AB \mid CA$, $B \to BC \mid AB$, $C \to aB \mid b$, $A \to a$}. eliminate useless symbols.

Soln:

Here B is useless symbol. so eliminate B production & variable.

$$\boxed{\begin{array}{c} S \to CA \\ C \to b \\ A \to a \end{array}}$$

Elimination of Null Production:

Null productions are those productions that are the form : $X \rightarrow \epsilon$.

These are also called as ϵ-production.

Ex1:

$S \rightarrow aS \mid \epsilon$

Here $S \rightarrow aS$ and $S \rightarrow \epsilon$ are the productions $S \rightarrow aS$ doesn't contain null production. But $S \rightarrow \epsilon$ is an null production.

$\therefore$ S is said to be nullable variable. The nullable variable used in the production is removed.

Ex2:

Remove ϵ production from the grammar, $P = \{S \rightarrow ABA, A \rightarrow \epsilon, B \rightarrow \epsilon\}$.

Soln:

A, B and S are nullable variables.

After eliminating ϵ,

$$S \rightarrow ABA \mid BA \mid AA \mid AB \mid A \mid B$$

Ex3: $P = \{S \rightarrow aS \mid AB, A \rightarrow \epsilon, B \rightarrow \epsilon\}$

Soln:

$S \rightarrow aS \mid AB \mid A \mid B$

Elimination of unit Production:

The unit Productions are those productions of the form $X \to Y$ where X & Y are variable of the grammar.

Elimination:

* Select the unit production $X \to Y$
* Add the Production $X \to \alpha$ to the grammar since $X \to Y$ and $Y \to \alpha$
* Remove the unit Production, $X \to Y$ from the grammar.

Example:

Eliminate the unit production from the grammar,

1) $P = \{S \to ABA \mid BA \mid AA \mid AB \mid A \mid B,\ A \to 0A \mid 0,\ B \to 1B \mid 1\}$

Soln:

$S \to ABA \mid BA \mid AA \mid AB \mid 0A \mid 0 \mid 1B \mid 1$

$A \to 0A \mid 0$

$B \to 1B \mid 1$

—x—

2) $P = \{E \to E+T \mid T,\ T \to T*F \mid F,\ F \to (E) \mid I,\ I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1\}$

Soln:

$E \to E+T \mid T*F \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

$F \to (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

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

2) simply the following grammar

$S \to ASB \mid \epsilon$

$A \to aAS \mid a$

$B \to SbS \mid A \mid bb$

Soln:

i) Elimination of Null Production

$S \to ASB \mid AB$

$A \to aAS \mid a \mid aA$

$B \to SbS \mid Sb \mid bS \mid b \mid A \mid bb$

ii) Elimination of unit Production

The unit production is the above grammar

$B \to A$

Final Production

$S \to ASB \mid AB$

$A \to aAS \mid a \mid aA$

$B \to SbS \mid Sb \mid bS \mid b \mid aAS \mid a \mid aA \mid bb$

There is no useless symbols.

—x—

3) Simply the following grammar

$S \to aAa \mid bBb \mid BB$

$A \to C$

$B \to S \mid A$

$C \to S \mid \epsilon$

Soln:

i) Elimination of null production

$C \to \epsilon, A \to \epsilon, B \to \epsilon.$

$S \to aAa \mid aa \mid bBb \mid bb \mid BB \mid B$

$A \to C$

$B \to A$

$C \to S$

Normal Forms For CFG:

Let the grammar $G = (V, T, P, S)$ be a context free grammar. If the production rules in G satisfy some restrictions, then they are said to be in normal form.

There are two types of normal forms:

* Chomsky Normal Form [CNF]
* Greibach Normal form [GNF]

Chomsky Normal Form [CNF]:

A context free grammar, $G = (V, T, P, S)$ is said to be in CNF if each production in G, is of the form $X \to YZ$; $X \to \alpha$, where $X, Y, Z \in V$ and $\alpha \in T$.

Algorithm to convert a CFG to CNF:

1. Eliminate the useless symbols, unit and null productions from the grammar.
2. Each variable should be having a derivation of length 2 or more, having only variable as of the form, $A \to \alpha$ Where $|\alpha| \geq 2$, $\alpha \in V$.
3. The production have three more than three variables as derivation, must be broken down into a cascade of productions with derivations containing at most 2 variables.

— x —

Ex:

Convert the grammar $S \to AB/aB$, $A \to aaB/\epsilon$, $B \to bbA$ into CNF.

Soln:

* Elimination of ϵ - production:

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

$A \rightarrow aaB$

$B \rightarrow bbA \mid bb$

* Elimination of unit production

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

$A \rightarrow aaB$

$B \rightarrow bbA \mid bb$

There is no useless symbol in the grammar.

* Conversion to CNF

1) Adding production to terminals

$C_a \rightarrow a$

$C_b \rightarrow b$

2) Rewriting the grammar

$S \rightarrow AB \mid C_bC_bA \mid C_bC_b \mid C_aB$

$A \rightarrow C_aC_aC_b$

$B \rightarrow C_bC_bA \mid C_bC_b$

Conversion the grammar to CNF gives

$S \rightarrow AB$

$S \rightarrow C_bC_bA \Rightarrow S \rightarrow C_bC_1$

$C_1 \rightarrow C_bA$

[illegible]

[illegible]

$S \rightarrow C_bC_b$

$S \rightarrow C_aB$

$A \rightarrow C_a C_a C_b \Rightarrow A \rightarrow C_a C_2$

$C_2 \rightarrow C_a C_b$

$B \rightarrow C_b C_b A \Rightarrow B \rightarrow C_b C_1$

$C_1 \rightarrow C_b A$

$B \rightarrow C_b C_b$

$C_a \rightarrow a$

$C_b \rightarrow b$

Final CNF grammar:

$S \rightarrow AB \mid C_b C_1 \mid C_b C_b \mid C_a B$

$A \rightarrow C_2 C_2$

$B \rightarrow C_b C_1 \mid C_b C_b$

$C_1 \rightarrow C_b A$

$C_2 \rightarrow C_a C_b$

$C_a \rightarrow a$

$C_b \rightarrow b$

Greibach Normal Form [GNF]

Definition:

A context free grammar is in GNF if all productions have the form

$$A \rightarrow a\alpha \ ; \ A \rightarrow a$$

Where, A is the variable, a is the terminal and α is any number of non terminals.

In GNF, there is no restrictions on the length of right side of the production.

Example:

1) Find GNF for the following grammar.

$S \rightarrow AB \ ; \ A \rightarrow BS \mid b \ ; \ B \rightarrow SA \mid a$

Soln:

To write the above grammar G into GNF, follow the following steps:

Step 1: Check the given grammar G whether it is in CNF, It is already in CNF.

Step 2: Replace the variables $S = A_1 \ ; \ A = A_2 \ ; \ B = A_3$

$$A_1 \rightarrow A_2 A_3$$
$$A_2 \rightarrow A_3 A_1 \mid b$$
$$A_3 \rightarrow A_1 A_2 \mid a$$

In the first production $1 < 2$, then there is no change.
In the second production $2 < 3$ " " " "
In the third production $3 > 1$,

$$A_{\underline{3}} \rightarrow A_{\underline{1}} A_2 \mid a$$

The replace A_1 with its productions

$$A_3 \rightarrow A_2 A_3 A_2 \mid a$$

After replacement, the production has 3 > 2, Then replace A_2,

$$A_3 \rightarrow A_3 A_1 A_3 A_2 \mid b A_3 A_2 \mid a$$

In the above production, i = j, 3 = 3, the Introduce the new variable B.

$$A_3 \rightarrow b A_3 A_2 \mid a \mid b A_3 A_2 B_3 \mid a B_3$$

$$B_3 \rightarrow A_1 A_3 A_2 \mid A_1 A_3 A_2 B_3$$

At this stage the grammar now looks like:

$$A_1 \rightarrow A_2 A_3$$

$$A_2 \rightarrow A_3 A_1 \mid b$$

$$A_3 \rightarrow b A_3 A_2 \mid a \mid b A_3 A_2 B_3 \mid a B_3 \Rightarrow GNF$$

$$B_3 \rightarrow A_1 A_3 A_2 \mid A_1 A_3 A_2 B_3$$

GNF Conversion:

A_3 is already in GNF. Apply A_3 value to A_2,

$$A_2 \rightarrow b A_3 A_2 A_1 \mid a A_1 \mid b A_3 A_2 B_3 A_1 \mid a B_3 A_1 \mid b$$

Now A_2 is in GNF. Apply A_2 value to A_1

$$A_1 \rightarrow b A_3 A_2 A_1 A_3 \mid a A_1 A_3 \mid b A_3 A_2 B_3 A_1 A_3 \mid a B_3 A_1 A_3 \mid b A_3$$

Now A_1 is in GNF. Apply A_1 value to B_3

$$B_3 \rightarrow b A_3 A_2 A_1 A_3 A_3 A_2 \mid a A_1 A_3 A_3 A_2 \mid b A_3 A_2 B_3 A_1 A_3 A_3 A_2 \mid$$
$$a B_3 A_1 A_3 A_3 A_2 \mid b A_3 A_3 A_2 \mid b A_3 A_2 A_1 A_3 A_3 A_2 B_3 \mid$$
$$a A_1 A_3 A_3 A_2 B_3 \mid b A_3 A_2 B_3 A_1 A_3 A_3 A_2 B_3 \mid a B_3 A_1 A_3 A_3 A_2 B_3 \mid$$
$$b A_3 A_3 A_2 B_3 .$$

The resultant grammar is:

P:

$A_1 \rightarrow bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_2B_3A_1A_3 \mid aB_3A_1A_3 \mid bA_3$

$A_2 \rightarrow bA_3A_2A_1 \mid aA_1 \mid bA_3A_2B_3A_1 \mid aB_3A_1 \mid b$

$A_3 \rightarrow bA_3A_2 \mid a \mid bA_3A_2B_3 \mid aB_3$

$B_3 \rightarrow bA_3A_2A_1A_3A_3A_2 \mid aA_1A_3A_3A_2 \mid bA_3A_2B_3A_1A_1A_3A_3A_2 \mid$
$aB_3A_1A_3A_3A_2 \mid bA_3A_3A_2 \mid bA_3A_2A_1A_3A_3A_2B_3 \mid$
$aA_1A_3A_3A_2B_3 \mid bA_3A_2B_3A_1A_1A_3A_3A_2B_3 \mid aB_3A_1A_3A_3A_2B_3 \mid$
$bA_3A_3A_2B_3$.

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

$V = \{A_1, A_2, A_3, B_3\}$

$T = \{a, b\}$

$P = \{ \quad \}$

$S = \{A_1\}$

PUMPING LEMMA FOR CFL:

1) Prove That $L = \{a^i b^i c^i \mid i \geq 1\}$ is not CFL.

Soln:

1) Let us assume that L is CFL

2) Let us pick up; $w = a^n b^n c^n$, where n is constant

3) w is rewritten as $uvxyz$, where.

i) $|vxy| \leq n$

ii) $vy \neq \epsilon$

iii) for all i, $uv^i xy^i z \in L$

If vy contains all three symbols a, b, c

$v = ab|bc|ac$ or $y = ab|bc|ac$

If $i = 2$, $uv^i xy^i z \Rightarrow uv^2 xy^2 z$

Case 1: If $v = ab$ and $y = c$

$uv^2 xy^2 z = (ab)^2 c \Rightarrow uv^i xy^i z \notin L$

Case 2: if $v = a$ and $y = bc$

$uv^2 xy^2 z = a(bc)^2 \Rightarrow uv^i xy^i z \notin L$

Hence L is not CFL.

2) Prove that $L = \{0^i 1^j 2^i 3^j \mid i \geq 1 \ \& \ j \geq 1\}$ is not context-free.

Soln:

1) Let us assume that L is CFL

2) Let $w = 0^n 1^n 2^n 3^n$, where n is a constant

3) Let w is rewritten as uvxyz where

i) $|vxy| \leq n$

ii) $vy \neq \epsilon$

iii) $uv^ixy^iz \in L$ for $i = 0, 1, 2, \ldots$

Let vy takes the symbols 012 or 123.

i) if v = 01 and y = 2

At i = 2, $uv^2xy^2z = (01)^2 (2)^2 3$ — ①

ii) if v = 12 and y = 3

At i = 2, $uv^2xy^2z = (12)^2 (3^2) 0$ — ②

From ① & ② there are unequal number of 012 and 3.

Hence L is not a CFL.

PROPERTIES OF CFL:

A CFL is closed under the following operations.

→ Union

→ Concatenation

→ Kleene star

A CFL is not closed under

→ Intersection

→ Complementation

→ Reversal.

Applications of Substitution Theorem:

- Union
- Concatenation
- Closure (*) and positive closure (+)
- Homomorphism.

1. **Union:**

   Let L_1 and L_2 be CFL's. Then the union of L_1 and L_2,

   $$S\,(L) = L_1 \cup L_2$$

   where L is the language $\{1, 2\}$ and S is the substitution given by $S(1) = L_1$ and $S(2) = L_2$.

2. **Concatenation:**

   Let L_1 and L_2 be CFL's. Then the concatenation of L_1 and L_2,

   $$S(L) = L_1.L_2$$

   where L is the language $\{1, 2\}$ and S is the substitution given by $S(1) = L_1$ and $S(2) = L_2$.

3. **Closure:**

Kleene closure

Let L_1 is a CFL's. Then the kleene closure of L_1 is given by,

$$S\ (L) = L_1*$$

where $S(1) = L, L = \{1\}*$.

Positive closure

Let L_1 is a CFL's. Then the positive closure of L_1 is given by,

$$S\ (L) = L_1^+$$

$$\text{where } S(1) = L, L = \{1\}^+$$

4. **Homomorphism:**

Let L be a CFL over alphabet Σ and h is a homomorphism on Σ. Let S be the substitution that replaces each symbol a in Σ, by one string h (a).

$S\ (a) \ = \{h(a)\}$ for all a in Σ

Thus $h\ (L) = S\ (L)$

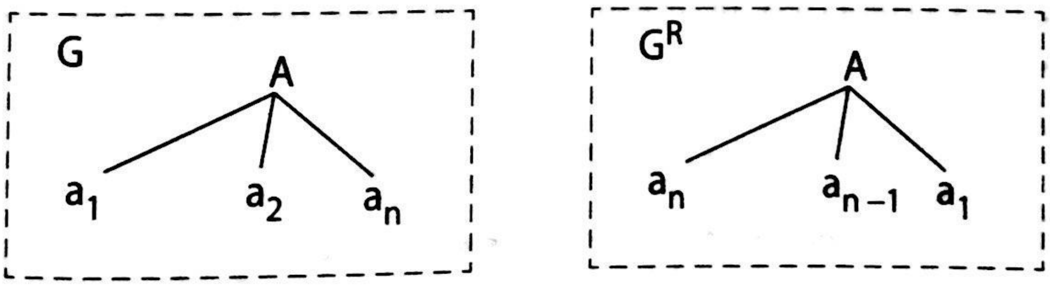
5. **Reversal:**

Theorem:

If L is a CFL, then is also a CFL.

Proof:

Let L = L (G) for some CFL. The CFL generated from CFG G = (V, T, P, S). Then construct the reverse of the grammar G^R = (V, T, P^R, S), where P^R is reverse of each production in P.



$$L(G^R) = L^R$$

All the sentential forms of G^R are reverse of the sentential forms of G.

6. **Intersection:**

Theorem:

L_1 and L_2 are context-free does not closed under intersection. i.e., $L_1 \cap L_2$ is not possible for context-free languages.

Example:

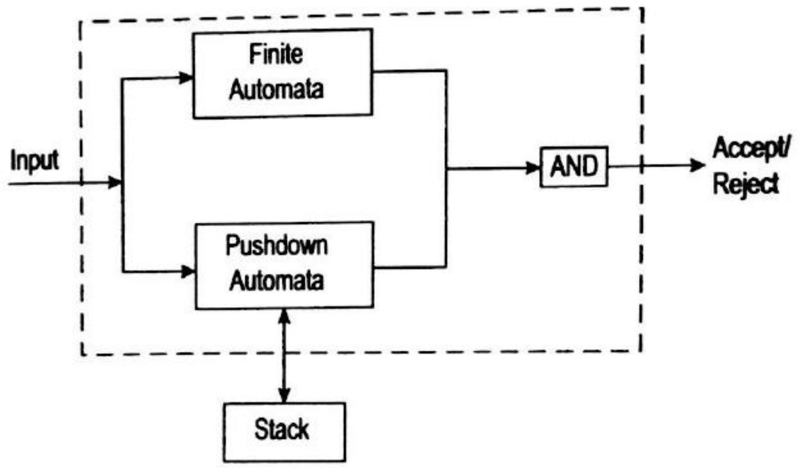
The languages $L_1 = \{0^m 1^m 0^n / m, n > 0\}$ and $L_2 = \{0^m 1^n 0^n / m, n > 0\}$, are both context-free languages. Then $L_1 \cap L_2$ is not possible. Because L_1 requires m number of 1's and L_2 requires number of 1's. The non-context-free intersection $L_1 \cap L_2 = \{0^m 1^{m+n} 0^n / m, n > 0\}$ is not possible.

Theorem:

The class of context-free languages is closed under intersection with regular languages, that is, for every context-free language L and regular language R, the language $L \cap R$ is context-free.

Proof:

Let the language L be context-free language and the language R be regular language,



To run the two automaton "in parallel" and result in another PDA.

Let $P = (Q_P, \Sigma, \Gamma, \delta_P, q_P, Z_0, F_P)$ be a PDA that accepts L by final state. Let $A = (Q_A, \Sigma, \delta_A, q_A, F_A)$ be a DFA for regular language L. Construct a new PDA, intersection of P and A

$$P' = (Q_P \times Q_A, \Sigma, \Gamma, \delta, \{q_P, q_A\}, Z_0, F_P \times F_A)$$

where $\delta((p, q), a, x)$ is the set of all pairs $((r, s), \gamma)$, such that

i. $r = \delta\hat{}(p, a)$ in Finite automata

ii. $(r, \gamma) = \delta\hat{}\ (q, a\ X)$ in Pushdown automata

Each move of PDA P and FA A, then there should be corresponding move in PDA P'. In finite automata

i. **If a is any value,** then $\delta^\wedge(p, a) = \delta^\wedge(p)$ for some states.

ii. **ii. If a** $= \varepsilon$ **then,** then $\delta^\wedge(p, a)$, A does not change the state.

In PDA

$$(q_P, w, Z_0) \vdash^* (q, \varepsilon, \gamma)$$

If and only if

$$((q_P, q_A)\ w, Z_0) \Vdash^* ((q, p)\varepsilon, \gamma)$$

where (q, p) is an accepting state of P′, if and only if q is an accepting state of FA A **and P is an accepting** state of PDA P.

To conclude that,

P′ accepts w if and only if both P and A accepts w, where w is in $L \cap R$

Difference:

Let the language L be context-free and the language R be regular. Then the language **L-R is context-free.**

$$L - R = L \cap \overline{R}$$

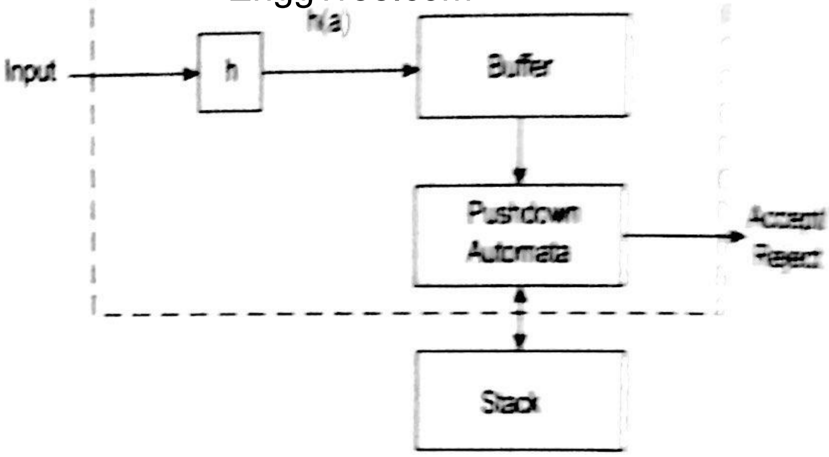
Complement:

Let L context-free does not imply that $\overline{L}$ is context-free

$$L \cup R = \overline{\overline{L} \cap \overline{R}}$$
$$L \cap R = \overline{\overline{L} \cup \overline{R}}$$

Inverse Homomorphism:

L is any language, h is homomorphism, then $h^{-1}(L)$ is the set of strings w such that **h(w) is in L. To** construct a PDA to accept the inverse homomorphism of given PDA **accepts.**



After reading a, applying homomorphism h(a) is placed in a buffer. The symbols of h(a) are used one at a time. When the buffer is empty, constructed PDA read another of its input symbols and apply the homomorphism to it.

Theorem:

Let L be a CFL and h be a homomorphism, then $h^{-1}(L)$ is a CFL.

Proof:

Construct a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ that accepts L by final state. h is a homomorphism applies to symbols of alphabet Σ and produces string s in T^*. L is a language over alphabet T. We construct a new PDA,

$$P' = (Q', \Sigma, \Gamma, \delta', (q_0, \varepsilon), Z_0, F \times \{\varepsilon\})$$

where

Q' is the set of pairs (q, x) such that,

i. q is a state in Q

ii. x is a suffix of some string h(a) for input symbol a in Σ.

First component of the stack of P' is the state of P, and the second component is the buffer. δ' is defined by

i. $\delta'((q, \varepsilon), a, x) = \{(q, (h(a)), X)\}$ where a in Σ, q in Q and X in Γ.

ii. $\delta(q, b, x) = (p, \gamma)$ where b is in T or $b = \varepsilon$, then

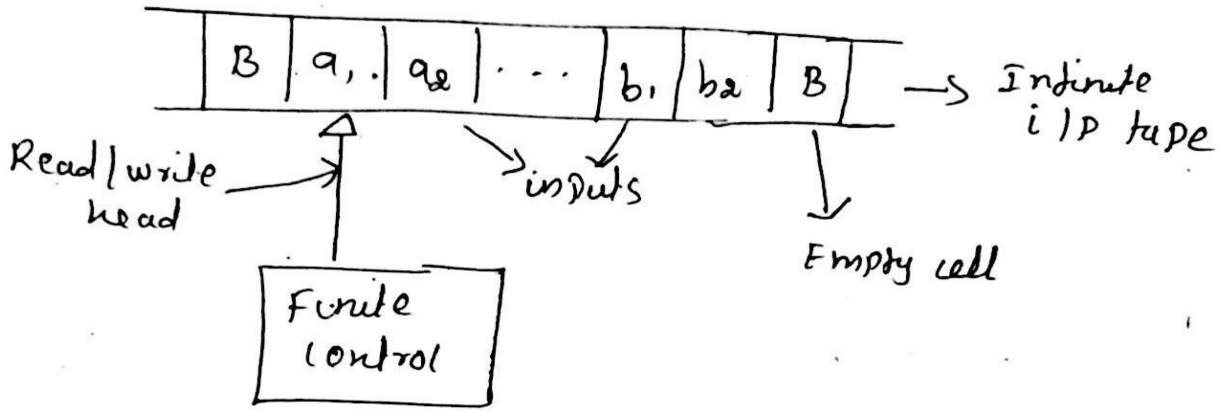
$$\delta'((q, bx), \varepsilon, X) = ((p, x), \gamma)$$

P' starts in the start state of P with an empty buffer. Accepting states of P', is an accepting state of P.

Turing Machine:

A turing Machine is an automatic machine that manipulates the input strings according to the transition rule.

Model of Turing Machine:



Definition of TM:

A Turing Machine M is a 7 tuple given by

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

where,

Q – Finite set of states, Σ – Finite set of inputs

Γ – Finite set of tape symbols $[\Sigma \cup B]$

δ – Transition function given by.

$\delta(q, a) = (q', b, M)$ [M → movement – Left, Right, no movement]

q_0 – Initial state

B – Blank symbol

F – set of final states

Instantaneous Description (ID) for TM:

Instantaneous description for TM is the snapshot of how the input string is processed by the Turing Machine.

It describes,

* The input string
* Position of the head
* State of the Machine

Ex: $L = \{a^n b^n \mid n \geq 1\}$

Instantaneous Description:

$n = 2$

B	a	a	b	b	B
	↑ q_0				

$\delta(q_0, a) = (q_1, X, R)$

B	X	a	b	b	B
		↑ q_1			

$\delta(q_1, a) = (q_1, a, R)$

B	X	a	b	b	B
			↑ q_1		

$\delta(q_1, b) = (q_1, b, R)$

B	X	a	b	b	B
				↑ q_1	

$\delta(q_1, b) = (q_1, b, R)$

B	X	a	b	b	B
					↑ q_1

$\delta(q_1, B) = (q_2, B, L)$

B	X	a	b	b	B
				↑ q_2	

$\delta(q_2, b) = (q_3, Y, L)$

②

B	x	a	b	Y	B
			↑ q_3		

$\delta(q_3, b) = (q_3, b, L)$

B	x	a	b	Y	B
		↑ q_3			

$\delta(q_3, a) = (q_3, a, L)$

B	x	a	b	Y	B
	↑ q_3				

$\delta(q_3, x) = (q_0, x, R)$

B	x	a	b	Y	B
		↑ q_0			

$\delta(q_0, a) = (q_1, x, R)$

B	x	x	b	Y	B
			↑ q_1		

$\delta(q_1, b) = (q_1, b, R)$

B	x	x	b	Y	B
				↑ q_2	

$\delta(q_1, Y) = (q_2, Y, L)$

B	x	x	b	Y	B
			↑ q_2		

$\delta(q_2, b) = (q_3, Y, L)$

B	x	x	Y	Y	B
		↑ q_3			

$\delta(q_3, x) = (q_0, x, R)$

B	x	x	Y	Y	B
			↑ q_0		

$\delta(q_0, Y) = (q_4, Y, R)$

1) Design a Turing Machine that accepts all strings of the form $a^n b^n$ for $n \geq 1$ and reject all other strings.

Soln:

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

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

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

$\Gamma = \{a, b, B, x, y\}$

$q_0 = \{q_0\}$

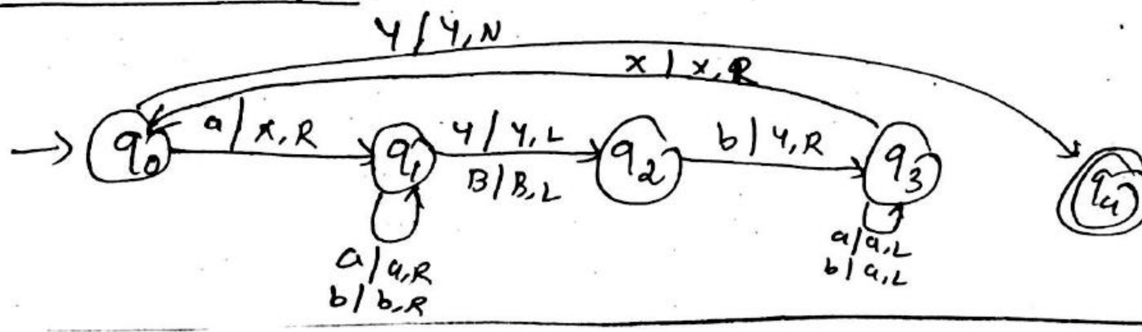
$B = \{B\}$

$F = \{q_4\}$

δ:

Q	a	b	x	y	B
$\rightarrow q_0$	(q_1, x, R)	-	-	(q_4, y, N)	
q_1	(q_1, a, R)	(q_1, b, R)	-	(q_2, y, L)	(q_2, B, L)
q_2	-	(q_3, y, R)	-	-	-
q_3	(q_3, a, L)	(q_3, b, L)	(q_0, x, R)	-	-
$*q_4$	ϕ	ϕ	ϕ	ϕ	ϕ

Transition diagram



③

2) Design TM that recognizes strings of the form $\{a^n b^n c^n \mid n \geq 1\}$ over $\Sigma = \{a, b, c\}$.

Soln:

$n = 2$ aabbcc

$\underset{\underset{q_0}{\uparrow}}{a}abbccB \vdash x\underset{\underset{q_1}{\uparrow}}{a}bbccB \vdash xa\underset{\underset{q_1}{\uparrow}}{b}bccB \vdash xay\underset{\underset{q_2}{\uparrow}}{b}ccB \vdash$

$xayb\underset{\underset{q_2}{\uparrow}}{c}cB \vdash xaybc\underset{\underset{q_2}{\uparrow}}{c}B \vdash xayb\underset{\underset{q_3}{\uparrow}}{z}cB \vdash xay\underset{\underset{q_3}{\uparrow}}{b}zcB \vdash$

$xa\underset{\underset{q_3}{\uparrow}}{y}bzcB \vdash \underset{\underset{q_3}{\uparrow}}{x}aybzcB \vdash x\underset{\underset{q_0}{\uparrow}}{a}ybzcB \vdash xx\underset{\underset{q_1}{\uparrow}}{y}bzcB \vdash$

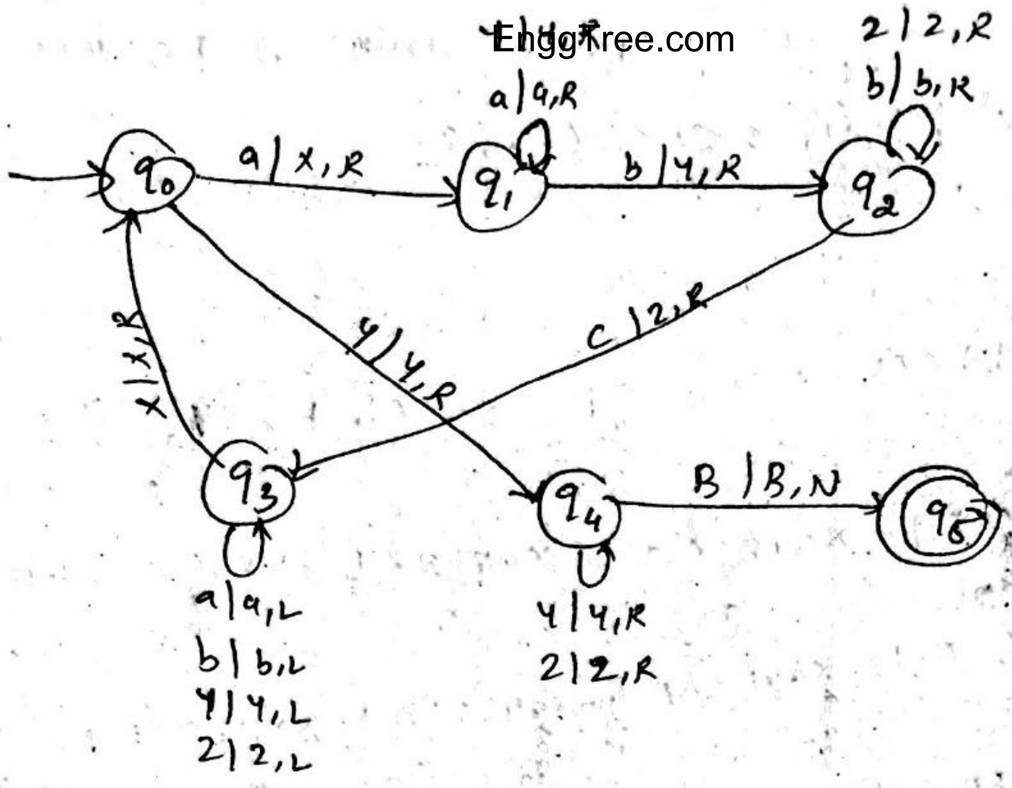
$xxy\underset{\underset{q_1}{\uparrow}}{b}zcB \vdash xxyy\underset{\underset{q_2}{\uparrow}}{z}cB \vdash xxyyz\underset{\underset{q_2}{\uparrow}}{c}B \vdash xxyy\underset{\underset{q_3}{\uparrow}}{z}zB \vdash$

$xxy\underset{\underset{q_3}{\uparrow}}{y}zzB \vdash xx\underset{\underset{q_3}{\uparrow}}{y}yzzB \vdash x\underset{\underset{q_3}{\uparrow}}{x}yyzzB \vdash xx\underset{\underset{q_0}{\uparrow}}{y}yzzB \vdash$

$xxyyzzB \vdash xxy\underset{\underset{q_4}{\uparrow}}{y}zzB \vdash xxyy\underset{\underset{q_4}{\uparrow}}{z}zB \vdash xxyyz\underset{\underset{q_4}{\uparrow}}{z}B \vdash xxyyzz\underset{\underset{q_4}{\uparrow}}{B}$

$xxyyzz\underset{\underset{q_4}{\uparrow}}{B} \vdash xxyyzzB\underset{q_5}{\uparrow} \Rightarrow$ Halt

Q	a	b	c	x	y	z	B
q_0	(q_1, x, R)	–	–	–	(q_4, x, R)	–	–
q_1	(q_1, a, R)	(q_2, y, R)	(q_2, b, R)		(q_1, y, R)	–	–
q_2	–	–	(q_3, z, L)	–	–	(q_2, z, R)	–
q_3	(q_3, a, L)	(q_3, b, L)	–	(q_0, x, R)	(q_3, y, L)	(q_3, z, R)	–
q_4	–	–	–	(q_4, y, R)	(q_4, y, R)	(q_4, z, R)	(q_5, B, N)
q_5	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ



3) Construct a TM to make a copy of a string over $\Sigma = \{0,1\}$.

4) Design a TM which recognizes palindrome over $\Sigma = \{a,b\}$ (or) $L = \{ww^R \mid w \in (a,b)^*\}$

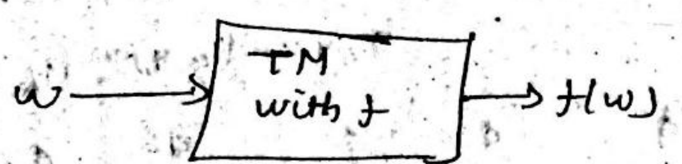
5) Design a TM that accept the language $L = \{a^n b^n c^n \mid n \geq 1\}$.

II Computable Functions:

It is capable of perform any sort of computations such as,

* Addition
* Subtraction
* Multiplication
* Division
* 1's complementation
* 2's complementation
* squaring number
* comparing two numbers

Function



2) Construct a TM to compute the function $f: N \rightarrow N$ such that $f(x) = x+1$.

Soln:

$x = 2 \Rightarrow 2+1 = 3$

$$\underset{\uparrow q_0}{0}0B \vdash 00\underset{\uparrow q_0}{B} \vdash 00\underset{\uparrow q_1}{0}B$$

δ:

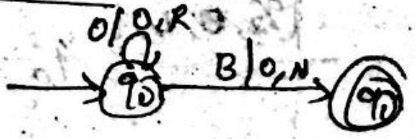
	0	B
q_0	$(q_0, 0, R)$	$(q_1, 0, N)$
q_1	–	–

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

$Q = \{q_0, q_1\}$; $\Sigma = \{0\}$, $\Gamma = \{0, B\}$, $B = \{B\}$, $q_0 = \{q_0\}$

$F = \{q_1\}$

Transition Diagram



3) Design a TM to compute proper subtraction

(i.e) $m-n$ for $m \geq n$

0 for $m < n$

4. Design a TM to compute proper multiplication

(i.e) $m \times n$: $f(m,n) = m \times n$.

5. Design a TM to compute the function $f(x) = 2x$

1) Design a TM to compute addition of two numbers (or) $f(x+y) = x+y$

Soln:

$w = 2, 3 \Rightarrow 2+3 = 5$

ID:

$B\underset{q_0}{\uparrow 0}0\#000B \vdash B0\underset{q_0}{\uparrow 0}\#000B \vdash B00\underset{q_0}{\uparrow \#}000B \vdash B000\underset{q_1}{\uparrow 0}00B$

$\vdash B0000\underset{q_1}{\uparrow 0}0B \vdash B000000\underset{q_1}{\uparrow B} \vdash B000000B \vdash B00000\underset{q_2}{\uparrow 0}B \vdash$

$\vdash B00000\underset{q_3}{\uparrow B}B \Rightarrow$ Halt.

δ:

Q	0	#	B
q_0	$(q_0, 0, R)$	$(q_1, 0, R)$	–
q_1	$(q_1, 0, R)$	–	(q_2, B, L)
q_2	(q_3, B, N)	–	–
q_3	ϕ	ϕ	ϕ

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

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

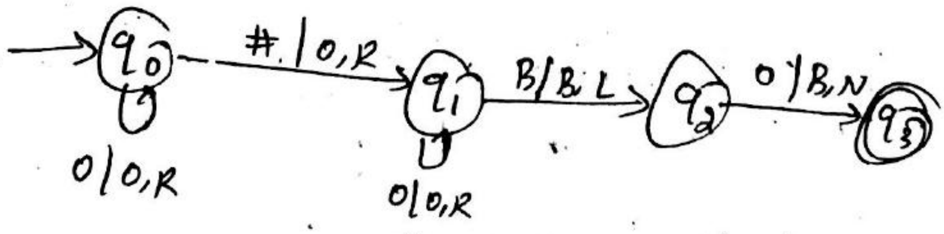
$\Sigma = \{0, \#\}$

$\Gamma = \{0, \#, B\}$

$q_0 = q_0$

$B = B$

$F = \{q_3\}$



Programming Techniques for Turing Machine Construction:

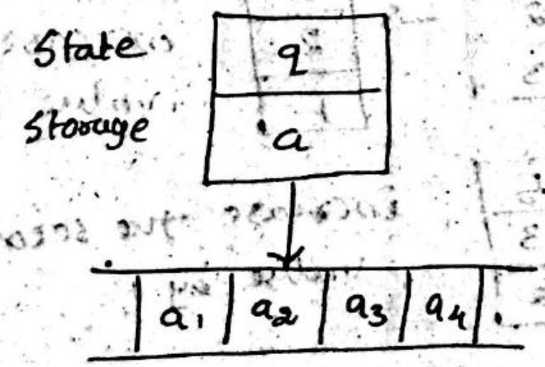
A turing Machine is also as powerful as a conventional computer. The following are the different techniques of constructing a TM to meet high-level needs.

1. Storage in the finite control (or) State
2. Multiple tracks
3. Subroutines
4. Checking off symbols.
5. Two-way infinite tape TM

1. Storage in State (or) Storage in Finite control:

The finite control can also be used to hold a finite amount of information along with the task of representing a position in the program.

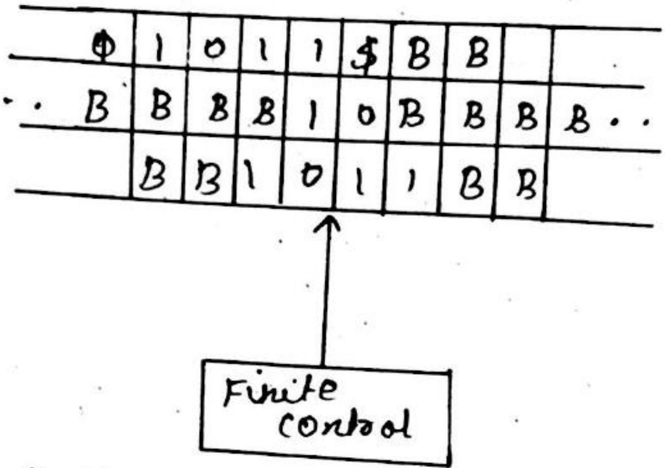
The state is written as a pair of elements, one for control and other storing a symbol.



Storage in finite control

2. Multiple tracks TM

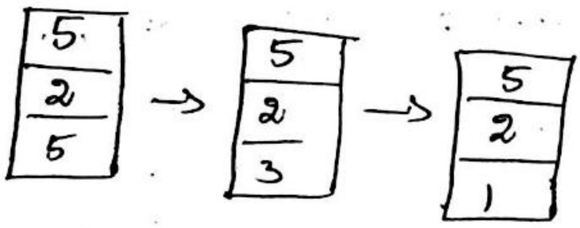
It is also possible that a TM input tape can be divided into several tracks. Each track can hold one symbol, and the tape alphabet of the TM consists of tuples with one component for each track.



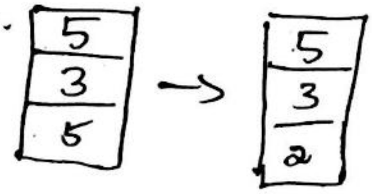
A three track Turing Machine

Ex: Design a TM to check whether the given input is prime or not using multiple tracks

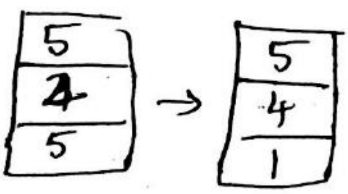
Ex: 5



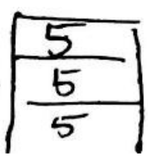
increase the second track value by 1



increase the second track value by 1



increase the second track value by 1



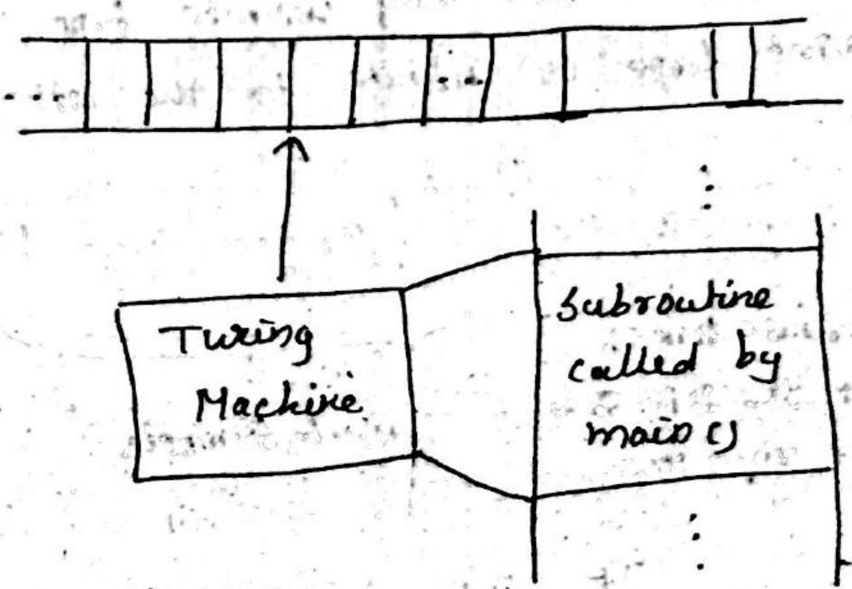
Here number on II track = number on III track

∴ the given number is a prime number.

3. Subroutine:

Subroutines are sub functions that can be used to execute repeated tasks for any number of times depending on the applications.

In such case, the Turing Machine has to be designed that handles subroutines.



The subroutine has two states

* Initial State
* Return State

When the main function is executed, the subroutine is called.

4. Checking off symbols:

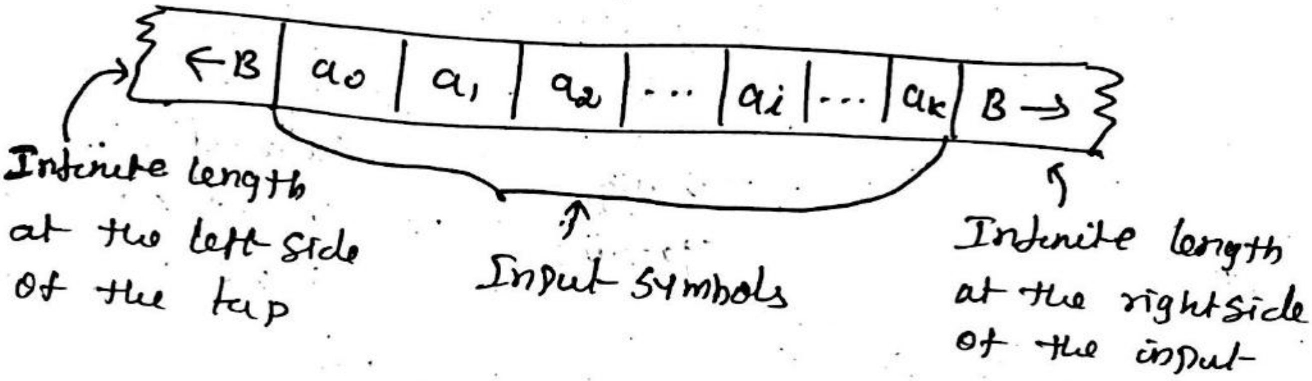
TM can be extended by using checking of symbol this method is used to by the TM for the language that contains repeated strings and ~~compute~~ compare the length of two ~~string~~ sub string.

Ex: Design TM for $L = \{a^n b^n \mid n \geq 1\}$.

~~a a b b B~~

5. Two-way infinite tape TM:

A TM, $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is said to be a two way infinite tape TM if the input tape is infinite to the left and right.



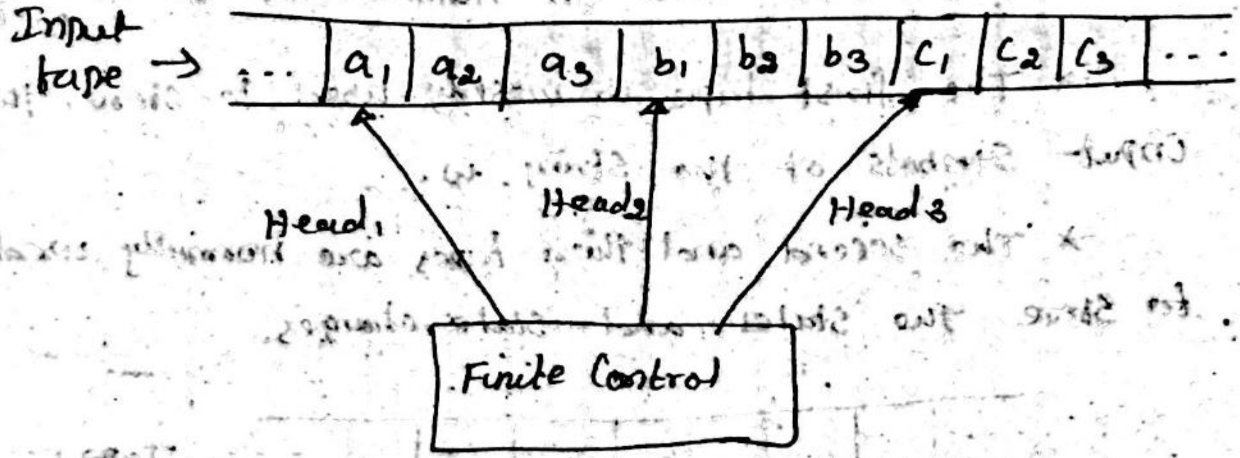
It can be viewed as a finite sequence of input symbols with infinite sequence of blank symbols to the left and right of the input.

As with the standard TM, there is fixed left end, the two way infinite tape TM has no left end.

Hence it can more as far as possible towards left as well as right.

MULTI HEAD TURING MACHINE:

A Turing Machine with a single tape can have any number of heads to provide simultaneous accesses over the tape.



The finite control processes two or more tape heads to access the input tape for performing multiple reads/writes in a simultaneous and independent manner.

The transition behaviour of a 2 headed tape is given as

$$\delta(q', H_1(a), H_2(a)) = ((q'', H_1(b), M_1), (H_2(b), M_2))$$

where

q', q'' → States of Q

$H_1(a)$ → Symbol to be processed (read) by H_1

$H_2(a)$ → Symbol to be processed (read) by H_2

$H_1(b)$ → Symbol to be replaced by H_1

$H_2(b)$ → Symbol to be replaced by H_2

M_1 → Movement of H_1 (L/R/N)

M_2 → Movement of H_2 (L/R/N)

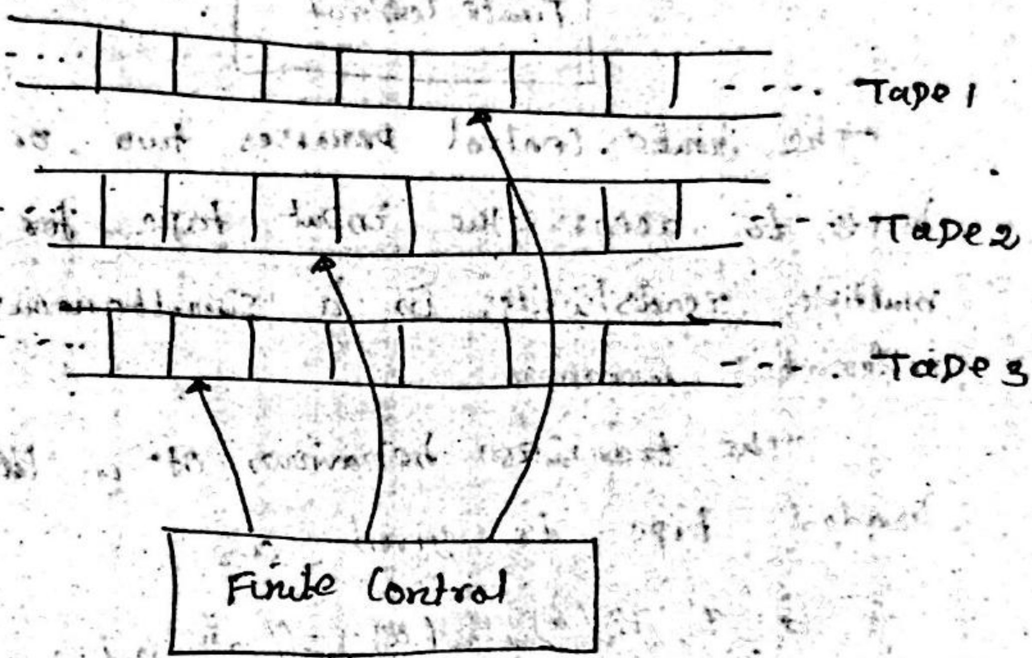
MULTI TAPE TURING MACHINE:

→ A multitape TM is finite control with more than one tape for read/writing symbols storing states etc.

→ Each tape has 'n' number of individual cells

→ The first tape is mostly used to store the input symbols of the string, w.

→ The second and third tapes are normally used to store the states and state-changes.



→ The transition of a two tape TM is given as,

$$\delta(q, a_1, a_2) = (q', H_1(b), M_1, H_2(b), M_2)$$

where,

q → current state to be processed

q' → next state to be reached

a_1 → symbol read by tape 1

a_2 → symbol read by tape 2

$H_1(b)$ → symbol to be written by tape 1

$H_2(b)$ → symbol to be written by tape 2

M_1 → movement by tape 1 (R/L/N)

M_2 → movement by tape 2 (R/L/N)

UNIT V UNDECIDABILITY

Non Recursive Enumerable (RE) Language – Undecidable Problem with RE – Undecidable Problems about TM – Post's Correspondence Problem, The Class P and NP.

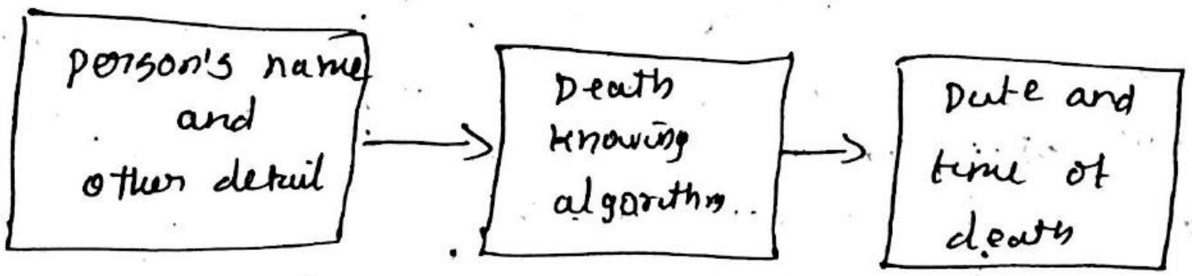
UNSOLVABLE PROBLEMS AND COMPUTABLE FUNCTIONS

UNSOLVABLE PROBLEMS:

A problem whose language is recursive is said to be decidable otherwise, the problem is undecidable. That is, a problem is undecidable if there is no algorithm that takes as input an instance of the problem and determines whether the answer to that instance is yes or no.

Ex:

Can you develop an algorithm which will correctly tell us when a person will die? whatever you want may be taken as input.



Undecidability of Death

For unsolvable problem, let us see the following problem.

1. Unsolvable problem of a non Recursive language
2. Unsolvable problem of Reduction
3. Unsolvable problem in Rice's Theorem
4. Post correspondence problem
5. Unsolvable problems of context Free Grammars

i) Unsolvable problem of a Non Recursive Language:

It a language is recursively enumerable then it is non Recursive. So now we have find a non recursive language that is unsolvable.

Let us see the following languages

1. Empty language (Le)
2. Non Empty language (Lne)
3. Non Self Accepting language (NSA)
4. Self Accepting language (SA)

1. Empty language "Le":

If $L(Mi) = \phi$, that is Mi does not accept any i/p, then w is in Le. Thus, Le is the language consisting of all TM's whose language is empty.

$$\boxed{Le = \{ M \mid L(M) = \phi \}}$$

2. Non Empty language (Lne):

If $L(Mi) \neq \phi$, then w is in Lne. Thus Lne is the language for TM that accept at least one input string

$$\boxed{Lne = \{ M \mid L(M) \neq \phi \}}$$

3. Non self Accepting language (NSA):

The Non self Accepting language (NSA) is defined as.

$$\boxed{NSA = \{ w \in \{0,1\}^* \mid w = E(T) \}}$$

for some Turing Machine T, and the i/p string $w \notin L(T)$

4. Self Accepting Language (SA):

The self Accepting language (SA) is defined as,

$$\boxed{SA = \{ w \in \{0,1\}^* \mid w = E(T) \}}$$

for some Turing Machine T, and the i/p string $w \in L(T)$

Theorem:

Non Empty language "Lne" is Recursively Enumerable.

Proof:

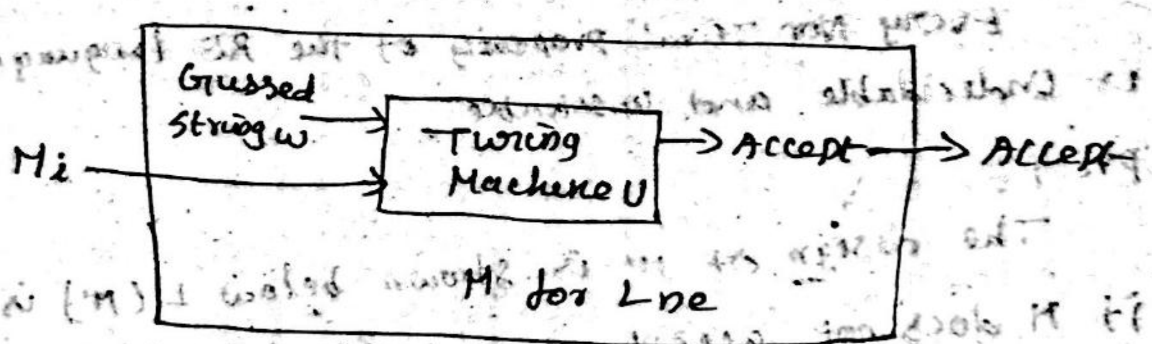
Now we have to construct a TM that accepts Lne.

M takes as inputs a Turing Machine code Mi

1) Using its Nondeterministic capability, M guesses an input w that Mi might accept.

2) M tests whether Mi accept 'w'. If Mi accepts 'w' then M also accepts its own input 'w'. This is done by M simulating the universal TM "U" that accept Lu.

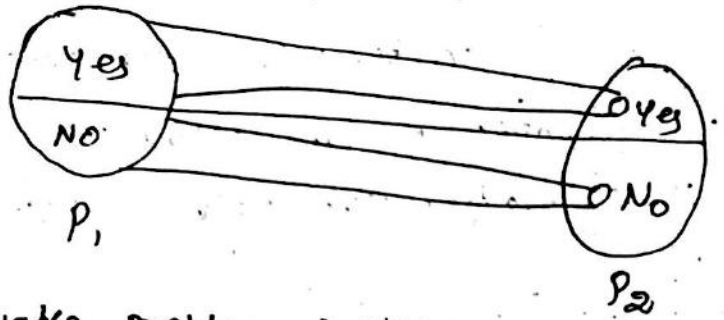
3) If Mi accepts w, then M accepts its own input, which is Mi



Thus if Turing Machine code Mi accepts even one string, M will guess that string and accept Mi. However, if $L(Mi) = \phi$, then no guess w leads to acceptance by Mi, so M does not accept Mi. Thus $L(M) = Lne$ and Lne is recursively enumerable.

2) Unsolvable Problem of Reduction:

Formally, a reduction from P_1 to P_2 is a TM that takes an instance of P_1 written on its tape and halts with an instance of P_2 on its tape. So reduction takes an instance of P_1 as input and produces an instance of P_2 as output.



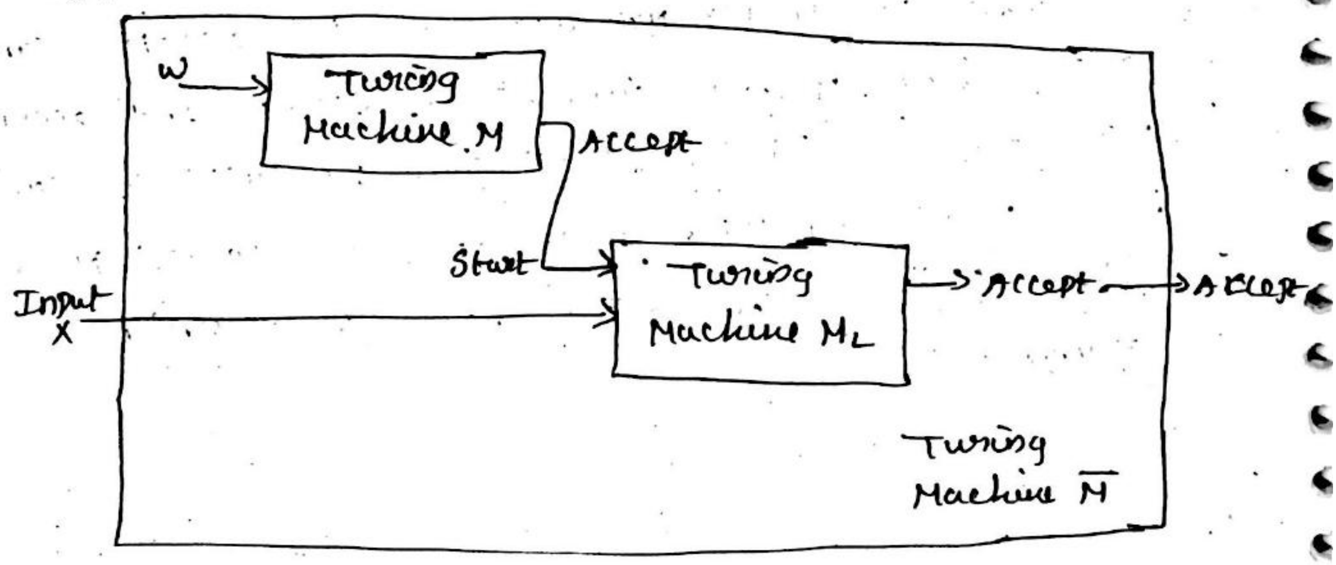
3) Unsolvable problem in Rice's Theorem:

⊗ Rice's theorem:

Every Non Trivial property of the RE language is Undecidable and unsolvable.

Proof:

The design of M' is shown below L(M') is ϕ if M does not accept w, and L(M') = L if M accept w.



Construction of M' for Rice Theorem

* $L(M') = \phi$ if M does not accept w
* $L(M') = L$ if M accept w

The Turing Machine M' is a two tape Turing Machine.

1) One Tape is used to simulates M on w
2) The other Tape of M' is used to simulates M_L on the input x to M'.

The Turing Machine M' is constructed to perform following,

→ Simulate M on input w. The String 'w' is not the input to M', rather M' writes M and w on to one of its tape and simulates the Universal Turing Machine 'U' on that pair.

→ If M does not accept w, then M' does not perform anything. M' never accept its own i/p x.

→ If M accept w, then M' accept its own i/p x.

This algorithm is a reduction of Lu to Lp, and proves that the property P is undecidable and unsolvable.

4) Post's correspondence Problem (PCP):

The undecidability of Strings is determined with help of Post's correspondence problem (PCP). Let us define the PCP.

The Post's Correspondence problem consist of two lists of strings that are of equal length over the input Σ.

The two list are

$$A = w_1, w_2, w_3, \ldots, w_n$$

and $$B = x_1, x_2, x_3, \ldots x_n$$

then there exists a non empty set of integers $i_1, i_2, i_3 \ldots i_n$ such that

$$w_1, w_2, w_3, \ldots, w_n = x_1, x_2, x_3, \ldots, x_n$$

to find $w_i = x_i$ then we say that PCP has a solution.

Example problem:

1) let $\Sigma = \{0,1\}$ and let A and B be the lists defined as follows,

i	List A w_i	List B x_i
1	1	111
2	10111	10
3	10	0

Find the instance of PC

Solution:

$w_1 = 1$ $x_1 = 111$
$w_2 = 10111$ $x_2 = 10$
$w_3 = 10$ $x_3 = 0$

Now find instance of PCP

$$w_1, w_2, w_3, \ldots w_n = x_1, x_2, x_3 \ldots x_n$$

let us take $m = 4$ and take combination 2, 1, 1, 3

$$w_2 w_1 w_1 w_3 = x_2 x_1 x_1 x_3$$

$$1011110 = 1011110$$

Instance of PCP

2) Let $\Sigma = \{0,1\}$ and A & B be the list as. 4.

i	List A	List B
i	w_i	x_i
1	10	101
2	011	11
3	101	011

Find the instance of PCP

Modified PCP:

An intermediate version of PCP is modified PCP (MPCP), there is the additional requirement on solution that the first pair on the A and B list must be the first pair in the solution.

An instance of MPCP is two lists,

$$A = w_1 w_2, \dots w_k$$
$$B = x_1 x_2, \dots x_k$$

And solution is a list of 0 or more integer $i_1, i_2, \dots i_m$ such that

$$w_1 w_{i_1}, w_{i_2} \dots w_{i_m} = x_1 x_{i_1}, x_{i_2}, \dots x_{i_m}$$

Example problem:

Consider the following list A & B and find instance

i	List A	List B
i	w_i	x_i
1	10	10
2	110	11
3	11	011

1	10
110	0
0	11

1,3,2

Soln:

Now the sequence taken is 2, 3

so $w_1 w_2 w_3 = x_1 x_2 x_3$

$1011011 = 1011011$

$\boxed{\text{Instance of MPCP} = 2,3}$

Completion of the Proof of PCP undecidability:

Rules:

1) The First pair is

List A	List B
#	# $q_0 w$ #

2) Tape symbols and separator # can be appended

List A	List B
X	X
#	#

3) To simulate a move of M,

List A	List B	Transition
qX	Yp	If $\delta(q,X) = (p,Y,R)$
ZqX	pZY	If $\delta(q,X) = (p,Y,L)$ Z is tape symbol
q#	Yp#	If $\delta(q,B) = (p,Y,R)$
Zq#	pZY#	If $\delta(q,B) = (p,Y,L)$

4) For each q in F, then for all tape symbols X and Y:

List A	List B
XqY	q
Xq	q
qY	q

5) Finally, we use the final pair to complete the solution

List A	List B
q##	#

Example:

1) Let us convert the Turing Machine, $M = (\{q_1, q_2, q_3\}, \{0,1\}, \{0,1,B\}, \delta, q_1, B, \{q_3\})$. where δ is

q_i	$\delta(q_i, 0)$	$\delta(q_i, 1)$	$\delta(q_i, B)$
q_1	$(q_2, 1, R)$	$(q_2, 0, L)$	$(q_2, 1, L)$
q_2	$(q_3, 0, L)$	$(q_1, 0, R)$	$(q_2, 0, R)$
q_3	–	–	–

And the input string $w = 01$. Find solution.

Soln:

Rule i:

List A	List B	Source
#	$\# q_1 01 \#$	$w = 01$, q_1 = Initial state
0	0	Tape symbols and the separator # is appended.
1	1	
#	#	
$q_1 0$	$1 q_2$	$\delta(q_1, 0) = (q_2, 1, R)$
$0 q_1 1$	$q_2 00$	$\delta(q_1, 1) = (q_2, 0, L)$
$1 q_1 1$	$q_2 10$	
$0 q_1 \#$	$q_2 01 \#$	$\delta(q_1, B) = (q_2, 1, L)$
$1 q_1 \#$	$q_2 11 \#$	
$0 q_2 0$	$q_3 00$	$\delta(q_2, 0) = (q_3, 0, L)$
$1 q_2 0$	$q_3 10$	
$q_2 1$	$0 q_1$	$\delta(q_2, 1) = (q_1, 0, R)$

List A	List B	Source
$q_2\#$	$0q_2\#$	$\delta(q_2,B)=(q_2,0,R)$
$0q_30$	q_3	
$0q_31$	q_3	
$1q_30$	q_3	
$1q_31$	q_3	
$0q_3$	q_3	q_3 is an Accepting State.
$1q_3$	q_3	
q_30	q_3	
q_31	q_3	
$q_3\#\#$	$\#$	q_3 is an Accepting State.

$w = 01$

$q_1 01 \vdash 1q_2 1 \vdash 10q_1 \vdash 1q_2 01 \vdash q_3 101$ //accepted

5. Unsolvable problems of CFG:

Theorem:

It is undecidable whether CFG is ambiguous.

Proof: we have to prove that " G_{AB} is ambiguous If and only If instance (A,B) of PCP has a solution.

If part:

There are two derivation G_{AB} as

$A \rightarrow w_1Aa_1 \mid w_2Aa_2 \mid \dots \mid w_kAa_k \mid w_1a_1 \mid \dots \mid w_ka_k$

$B \rightarrow x_1Ba_1 \mid x_1Ba_2 \mid \dots \mid x_kBa_k \mid x_1a_1 \mid \dots \mid x_ka_k$

where A and B are the list generated by CFG.

Now derivations are,

$$S \to A \to w_{i_1} w_{i_2} \cdots w_{i_m} a_{i_m} \cdots a_{i_1}$$

$$S \to B \to x_{i_1}, x_{i_2} \cdots x_{i_m} a_{i_m} \cdots a_{i_1}$$

The solution is

$$w_{i_1}, w_{i_2} \cdots w_{i_m} = x_{i_1}, x_{i_2} \cdots x_{i_m}$$

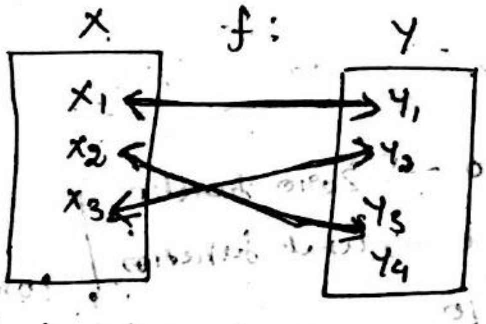
This implies that G_{A+B} is ambiguous.

COMPUTABLE FUNCTIONS:

Primitive Recursive Functions:

Basic Definitions

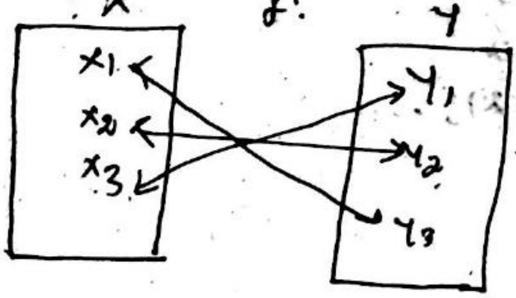
1. Partial Function



Partial function, f from x to y is a function that assigns every elements of x to at most one element of y.

Ex: $f(m,n) = m-n$ is a partial function [since m−n generate +ve & −ve values]

2. Total Function:



Total function, from x to y is defined as the function that assigns every element of x to unique element of y

Ex: $f(m,n) = m+n$ [generates only five values]

3. Initial Function:

The initial function include

i) constant Function

ii) successor Function

iii) Projection function

i) Constant function:

A function of the form,

$C_a^k : N^k \rightarrow N$ for $k \geq 0$ & $a \geq 0$ is called constant Function

General Form:

$$\boxed{C_a^k(x) = a \quad \text{for } x \in N^k}$$

Example:

$C(5) = 0 \rightarrow$ zero function

$C(4) = 1 \rightarrow$ unit function

$C(10) = 10$

⋮

} constant Function

ii) Successor Function [S]:

A function, ~~S~~ $S: N \rightarrow N$ defined by $S(x) = x+1$ is said be successor function.

Example:

$S(4) = 5$

$S(1) = 2$

$S(20) = 21, \ldots$

iii) Projection Function [P_i^k]:

A function of the form, $P_i^k : N^k \to N$ defined by $P_i^k(x_1, x_2, x_3 \ldots x_i \ldots x_k) = x_i$ for $k \geq 1$ and $1 \leq i \leq k$.

Example: $P_1^3(1, 3, 5) = 1$

$P_3^4(2, 4, 6, 8) = 6$

$P_2^2(0, 1) = 1$, etc

4. Complex Primitive Recursive Functions:

Complex functions are obtained by applying certain operations on the initial functions. They are,

i) Composition

ii) Primitive Recursion

i. Composition:

Let 'f' be a partial function defined as,

$N^k \to N$, for $1 \leq i \leq k$

and 'g' is a partial function from $N^m \to N$, then the composition of 'f' and 'g' is a partial function, defined by 'h' as

$$h(x) = f(g_1(x), g_2(x), \cdots g_k(x)) \quad [\because x \in N^m]$$

that is,

$$h(x) = f(g(x_1, x_2, \cdots x_n), g_2(x_1, x_2, \cdots x_n), \cdots g_k(x_1, x_2 \cdots x_n))$$

ii. Primitive Recursion:

A function 'f' over N is recursive if there exists a constant 'k' and a function 'h' (x, y) such that

$f(0) = k$

$f(n+1) = h(n, f(n))$

let g be function of 'n' variable and 'h' be

another function with '(n+2)' variables.

The primitive Recursion function is obtained as,

$$\boxed{\begin{aligned} &f: N^{n+1} \to N, \\ &f(x,0) = g(x) \\ &f(x,k+1) = h(x,k,f(x,k)),\ x \in N^n,\ k \geq 0 \end{aligned}}$$

That is,

$$f(x_1, x_2, \dots x_n, 0) = g(x_1, x_2, \dots x_n) \text{ and}$$

$$f(x_1, x_2, \dots x_n, k+1) = h(x_1, x_2, \dots x_n, k, f(x_1, x_2, \dots x_n, k))$$

Example problems:

1) Let $f_1(x,y) = x+y$, $f_2(x,y) = 2x$, $f_3(x,y) = xy$, $g(x,y,z) = x+y+z$. Find the composition of g with f_1, f_2, f_3.

Soln:

The composition function, $h(x,y)$ is given as,

$$h(x,y) = g(f_1(x,y), f_2(x,y), f_3(x,y))$$
$$= g(x+y, 2x, xy)$$
$$= x+y+2x+xy$$
$$\boxed{h(x,y) = x+y+2x+xy}$$

Here, f_1, f_2, f_3, g are total function then h also total function

2) Given: $f_1(x,y) = x-y$, $f_2(x,y) = y-x$, $g(x,y) = x+y$
~~Obtain~~ obtain $h(x,y)$ over g and f.

Soln:

$$h(x,y) = g(f_1(x,y), f_2(x,y))$$
$$= g(x-y, x-y)$$
$$\boxed{h(x,y) = x-y+x-y}$$

3) Let $f_1(x_1, x_2) = x_1 x_2$, $f_2(x_1,x_2) = \lambda$, $f_3(x_1, x_2) = x_1$, $g(x_1, x_2, x_3) = x_2 x_3$. Generate composition function $h(x_1, x_2)$.

Soln:

$$h(x_1, x_2) = g(f_1(x_1, x_2), f_2(x_1, x_2), f_3(x_1, x_2))$$
$$= g(x_1 x_2, \lambda, x_1)$$
$$= \lambda x_1$$

$$\boxed{h(x_1, x_2) = \lambda x_1 = x_1}$$

4) Show that $f_1(x, y) = x + y$ is primitive recursive.

Soln:

To prove that f_1 is primitive recursive.

Let g be a function of single variable and h be a function of three variables.

When $y = 0$:

$$f_1(x, y) = f_1(x, 0)$$
$$f_1(x, 0) = x + 0 = x = g(x) = U_1^1(x)$$

When $y = y+1$

$$f_1(x, y) = f_1(x, y+1)$$
$$f(x, y+1) = h(x, y, f_1(x, y))$$
$$= U_3^3(x, y, z)$$

$\therefore$ g and h are primitive recursive

Hence $f_1(x, y)$ is primitive recursive.

—x—

2) Show that $f(x, y) = x * y$ is primitive Recursive.

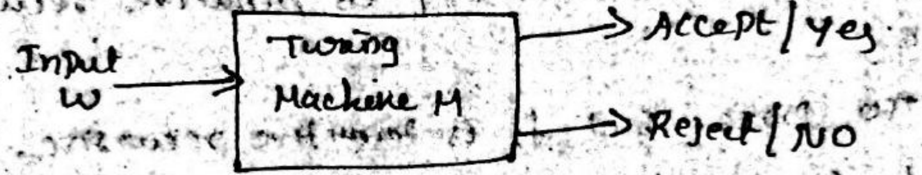
RECURSIVE AND RECURSIVELY ENUMERABLE LANGUAGES:

Recursive language:

A language 'L' is said to be Recursive language if $L = L(M)$ for some Turing Machine M such that,

1) If w is in L, then M accept
2) If w is not in L, then M reject but it halt

So L is called decidable language if it is a Recursive language



Recursive Enumerable Language [RE]:

A language is Recursively Enumerable if there exists a Turing Machine that accepts every string present in the language and does not accept the string and it may cause the Turing Machine to enter into an infinite loop.

Properties of Recursive And RE languages:

1) The Union of two Recursively Enumerable [RE] is Recursively Enumerable.
2) The language L and its complement $\bar{L}$ are recursively enumerable, then L and $\bar{L}$ are recursive
3) The complement of a recursive language is also recursive.
4) The union of two recursive language is recursive.
5) The intersection of two recursive language is recursive.
6) ~~The union of the two recursively enumerable language is recursively enumerable.~~
6) The intersection of two recursively enumerable language is recursively enumerable.

Theorem:

If L is a recursive language, then L' is also a Recursive language.

(or)

If L is recursive language, so is L'.

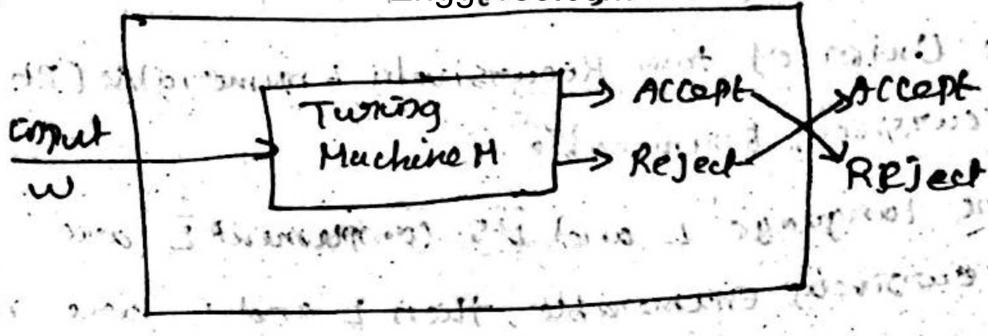
(or)

The complement of recursive language is also recursive language.

Proof:

To prove L is recursive then L' is also recursive.

let us construct the complement of the Turing Machine M as M' such that L' = L(M') and its shown below



Construction of M' accepting the complement of M

M' just behaves like M. The Turing Machine M is constructed as follows:

1) The Accepting States of M are made as non accepting States of M' with no transitions.

2) M' has ~~no~~ a new accepting state 'r' and there are no transitions from 'r'.

Since M is guaranteed to halt, then M' is also guaranteed to halt. The Turing Machine M' exactly accepts those strings that are not accepted by TM

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

. So the complement of recursive Language is also recursive.

— . —

Theorem:

If a language L and L' are Recursively Enumerable (RE) then L is Recursive.

(or)

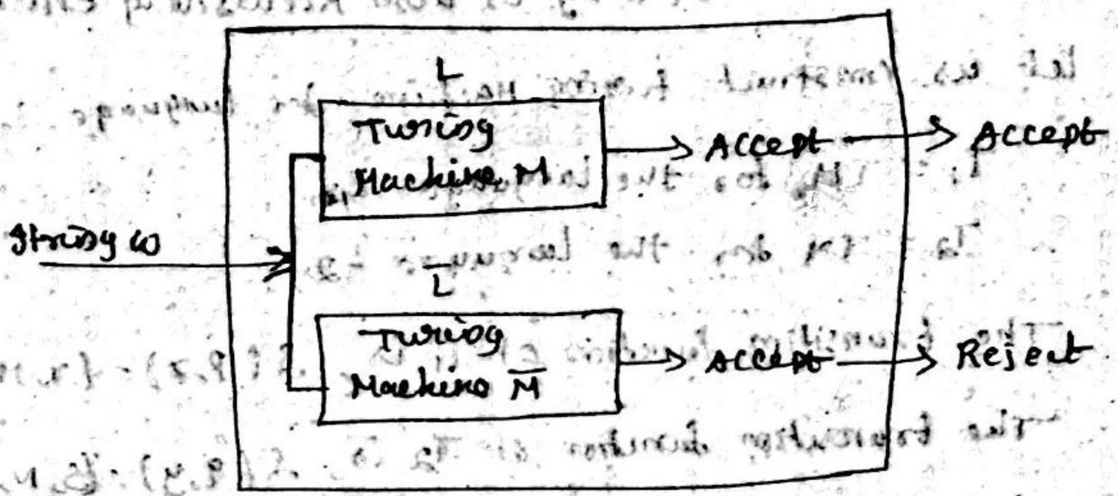
If both a language L and its complement L' are RE then L is recursive language.

Proof:

To prove L and L' is RE, then L is Recursive.

Let $L = L(M_1)$ and $L' = L(M_2)$ for some Turing Machine M_1 and M_2. Both M_1 and M_2 are simulated in

Parallel by a Turing Machine M. And it is shown below.



Simulation of Two TM accepting a lang & its complement

Here we are converting the two tape TM M' is to one tape TM 'M' to make simulation easy.

* One tape of M simulates the tape of M_1
* The other tape of M simulates the tape of M_2
* The states of M_1 & M_2 are each components of the state of M.
* If input w to M is in L, then M_1 will also accept. then M accept and halts.
* If input w is not in L, then M halts without accepting.

∴ We can conclude that L is Recursive language.

Theorem:

If L_1 and L_2 are recursively enumerable language over Σ, then $L_1 \cup L_2$ is also RE.

(or)

Union of two recursively enumerable language is RE.

Proof:

To prove $L_1 \cup L_2$ is also Recursively enumerable.

let us construct turing Machine for language L_1 and L_2.

T_1 = TM for the language L_1

T_2 = TM for the language L_2

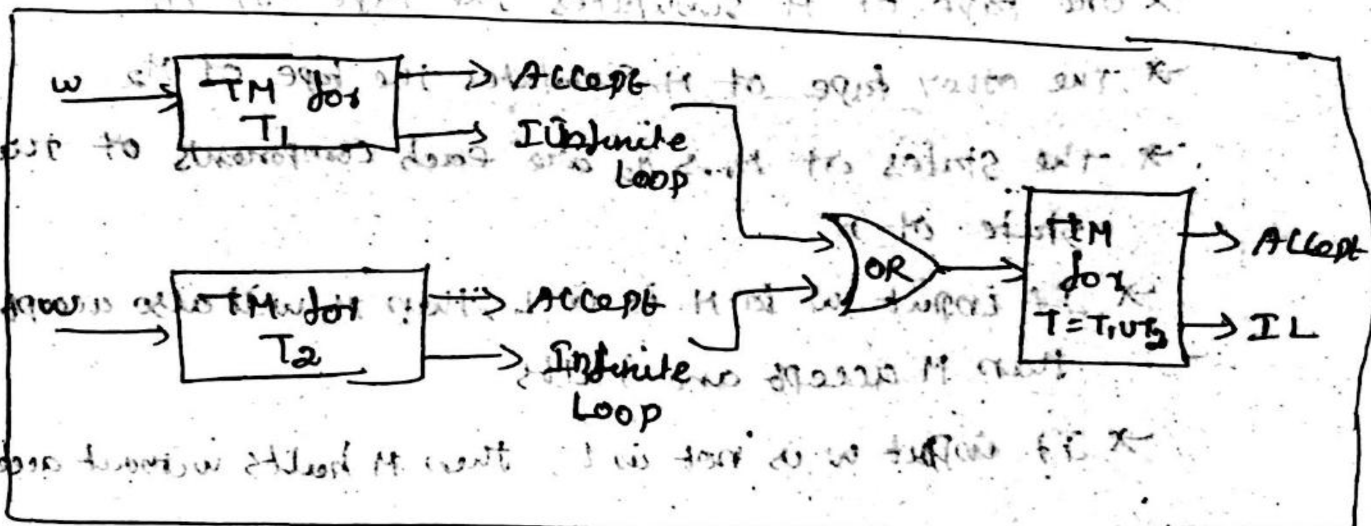
The transition function of T_1 is, $\delta(p, x) = (r, M, L)$

The transition function of T_2 is, $\delta(q, y) = (s, N, R)$

Then the transition of $T_1 \cup T_2$ will be,

$$\delta[(p,q),(x,y)] = [(r,s),(M,N),(L,R)]$$

The simulation of this machine is given below,



$T = T_1 \cup T_2$ is also Recursively Enumerable.

Here the turing machine T accept the string is accepted by any one of T_1 and T_2 and T enter in to infinite loop if both T_1 and T_2 reject the string.

Hence $T = T_1 \cup T_2$ and its language $L = L_1 \cup L_2$ is recursively enumerable.

Theorem:

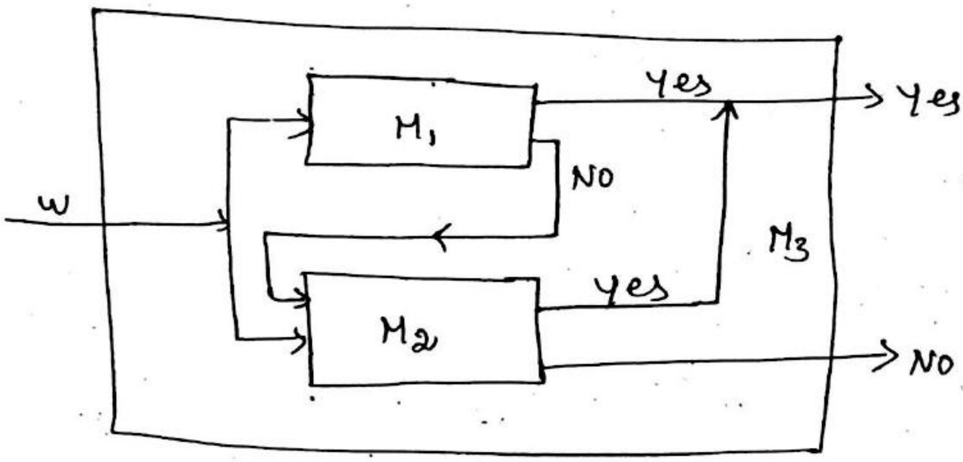
The union of two recursive language is recursive.

Proof:

Let L_1 and L_2 be two recursive language that are accepted by the Turing machine M_1 and M_2, given by

$$L(M_1) = L_1$$
$$L(M_2) = L_2$$

Let M_3 be the Turing Machine constructed by the union of M_1 and M_2. M_3 is constructed as,



* If $w \in L_1$, then M_1 accepts and thus M_3 also accepts since $L(M_3) = L(M_1) \cup L(M_2)$

* If M_1 reject. [$w \notin L_1$], then M_3 simulates M_2. M_3 halt with "yes" if M_2 accepts "w" else returns "NO".

Hence M_3, M_2, M_1 halt with either Yes or NO.

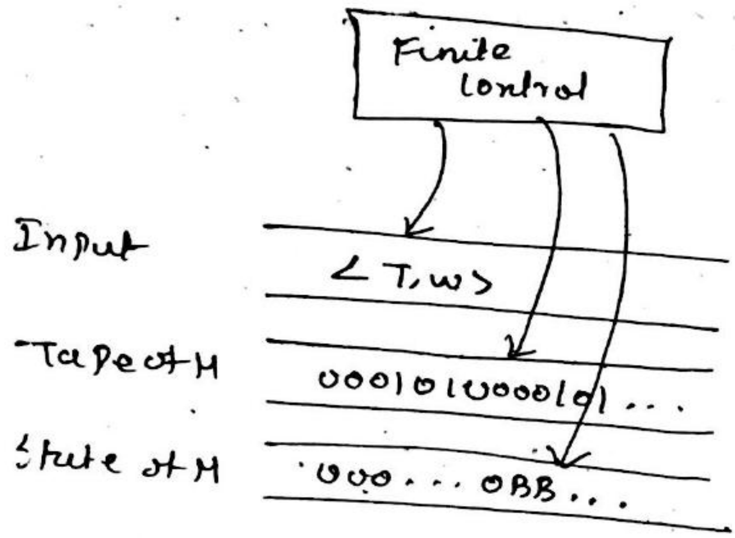
Thus the union of two recursive language is also recursive.

Universal Turing Machine:

→ The universal Turing machine, T_u takes over the program and input set to process the program

→ The program and the inputs are encoded and stored on different tapes of multi-tape TM.

→ The T_u thus take up $<T, w>$ where T is the special purpose TM that passes the program in the form of binary string, w is the data set that is to be processed by T.



The universal language L_u, is the set of all binary string (α), where α represents the ordered pair $<T, w>$ where.

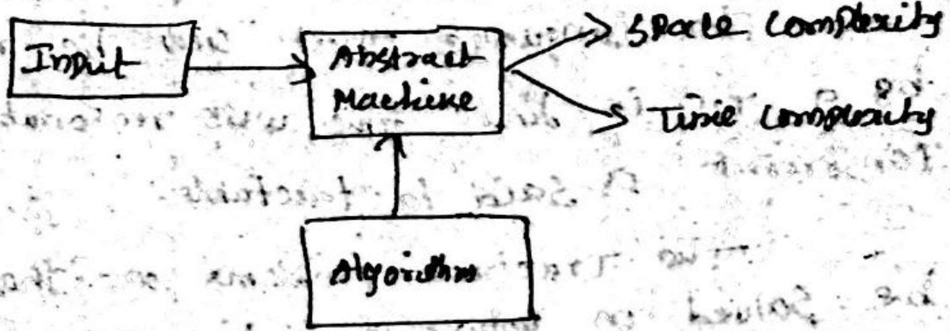
T → Turing Machine

w → any input string accepted by T

It can also be represented as $\alpha = <e(T)\,e(w)>$

MEASURING AND CLASSIFYING COMPLEXITY:

Growth Rates of Functions:



Complexity problem is classified in two types

- Space complexity.
- Time complexity

→ The space complexity is the amount of memory space required by the algorithm) problems to complete its computation.

→ The time complexity is the amount of time required to run the program of the problem

The growth rates of the functions can also be compared using the asymptotic notations like

- Big-oh [O]
- Big-Omega [Ω]
- Big-Theta [Θ]

TC

TRACTABLE AND INTRACTABLE PROBLEMS:

Tractable Problems | Language:

The language that can be ~~regco~~ recognized by a TM in finite time with reasonable space constraint is said to tractable.

The Tractable problems are those that can be solved in Polynominal time period.

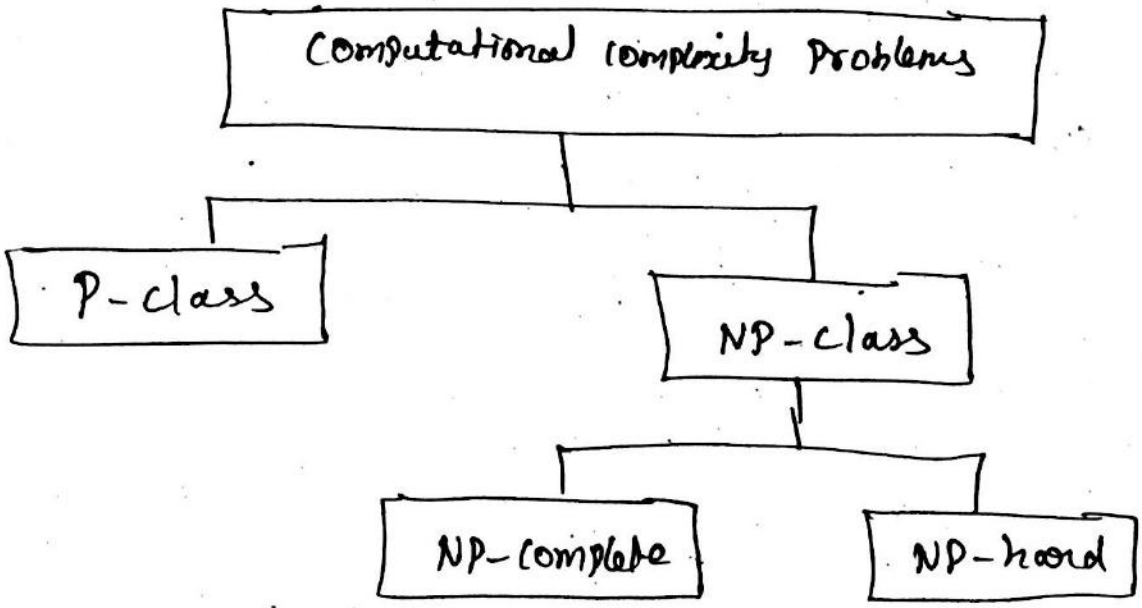
Intractable problem:

The language that cannot be recognized by a TM with reasonable space and time constraint is called as intractable problems.

These problems cannot be solved in Polynominal time period

Tractable and Possibly Intractable Problems: P and NP:

These are two groups in which a problem can be classified



P-class problem:

Problems that can be solved in Polynomial time.

Ex: Searching element, sorting element, multiplication of integers, finding all-pair-shortest path, finding minimum spanning tree, etc.

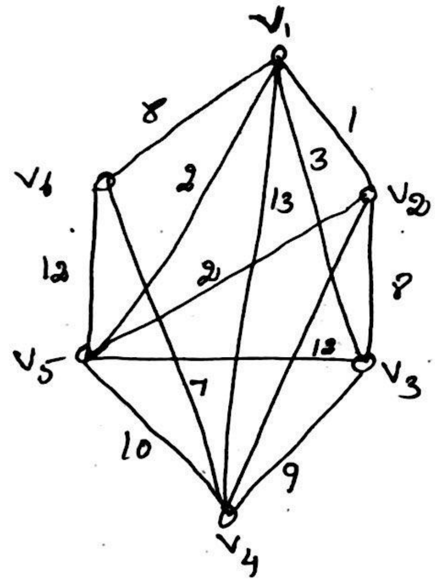
Example of P class problem:

Kruskal's Algorithm: → In Kruskal's alg. the minimum weight is obtained.

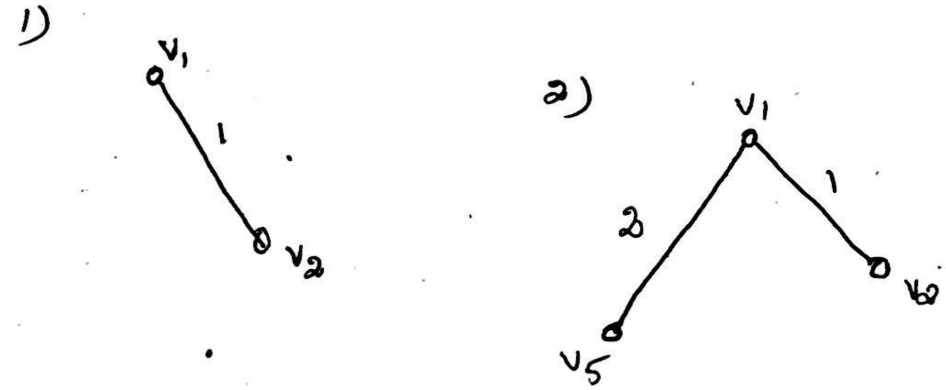
→ In this alg also the circuit should not be formed.

→ Each time the edge of minimum weight is selected.

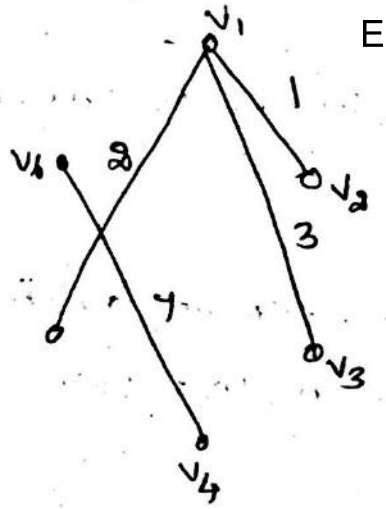
Example 1:



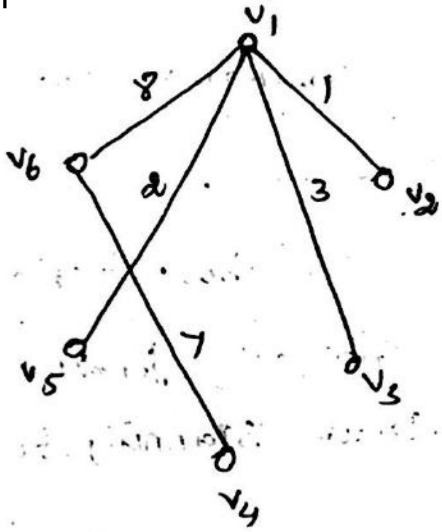
Soln:



3)

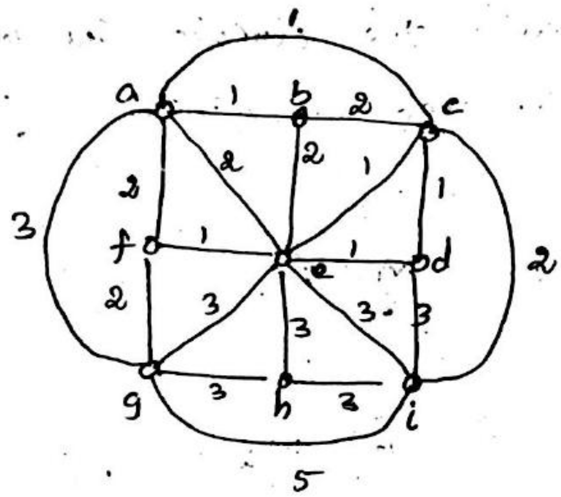


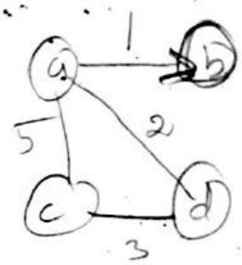
4)



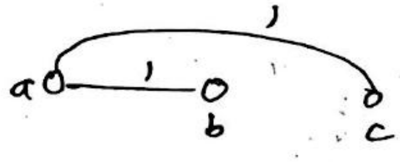
weight = 21

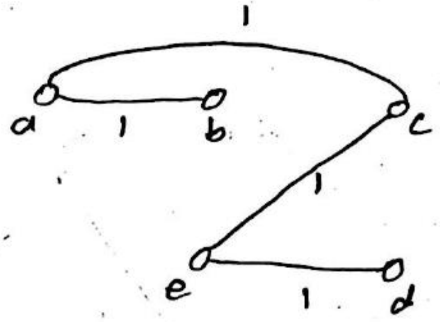
Example 2: Find the minimum spanning tree using Kruskal's algorithm.



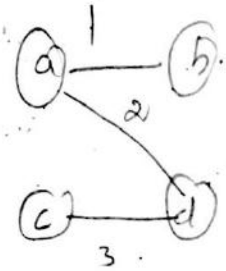


Soln:



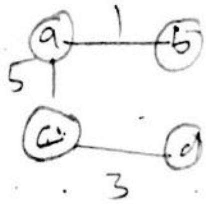


1)



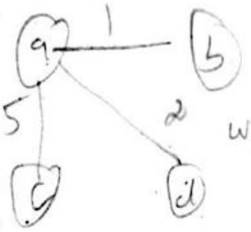
$w(T_1) = 6$

2)

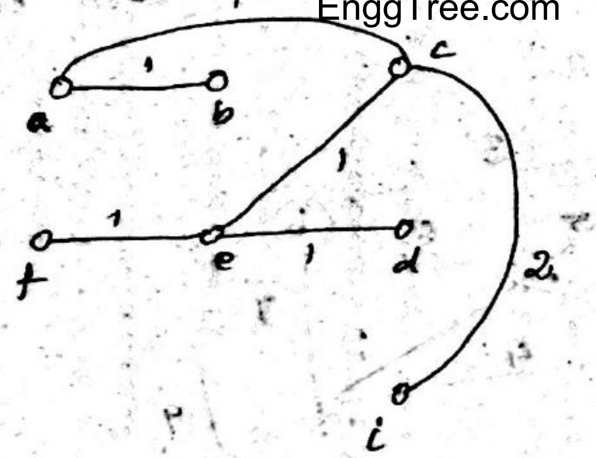


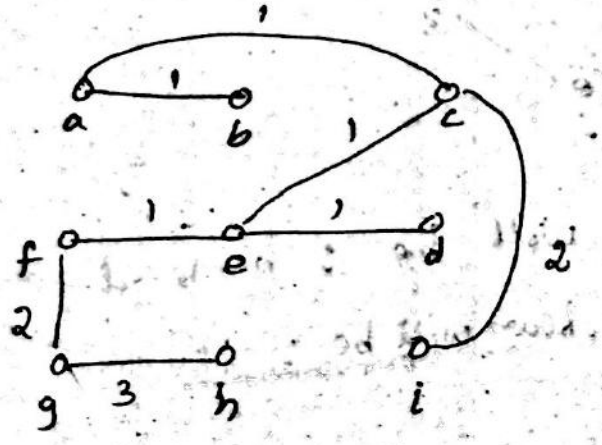
$w(T_2) = 9$

3)



$w(T_3) =$





NP-class problem:

Problems that can be sloved in non-deterministic polynomial time is called as NP-class problems

These types of NP problems are known as intractable problems

Example:

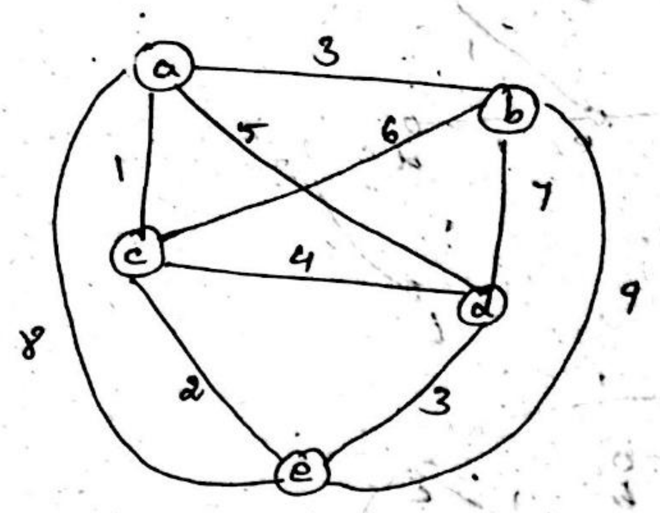
Towers of Hanoi, Traveling Salesman problem, Graph colouring problem, Hamiltonian circuit problem, Satisfiability problem, etc.

Example of NP class problem:

Traveling Salesman's problem (TSP)

-> Given a set of cities and cost to travel b/w each pair of cities, determine whether there is a path that visit every city once and returns to the

the first city. such that the cost travelled is less



The tour path will be : a - b - d - e - c - a

The total cost of tour will be : 16

NP - complete problem:

A problem is said to be NP - complete if it belongs to NP class problem and can be solved in polynomial time.

They are also called polynomial - time reducible problem.

NP - Hard problem:

→ A problem is said to be NP hard if there exists an algorithm for solving it and it can be translated into one for solving another NP problem.

→ A problem P_1 is NP-hard if

1) The problem is an NP class problem

2) For any other problem P_2 in NP, there is a polynomial time reduction of L_2 to L_1

→ Every NP complete problem must be NP - hard problem.

PROBLEMS ABOUT TURING MACHINE:

In problems about Turing machine, we are going to see the following concepts.

1) Decidable problems
2) Undecidable problems
3) Codes for the Turing Machine
4) Enumerating the Binary string of Turing Machine
5) Undecidable problems about Turing machine

Problem types,

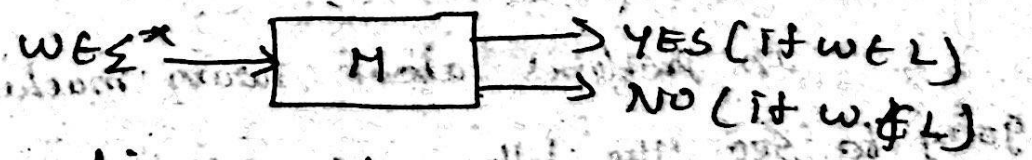
There are basically three types of problems namely,

* Decidable | solvable | recursive
* undecidable | unsolvable
* semi decidable | partial solvable | recursively enumerable

Decidable | Solvable problems:

A problem, P is said to be decidable if there exists a turing machine, TM that decides P. Thus P is said to be recursive.

Consider a TM, M that halts with either 'yes' or 'no' after computing the input.



The machine is defined as,

$$F_P(w) = \begin{cases} 1 & \text{if } P(w) \\ 0 & \text{if } \neg P(w) \end{cases}$$

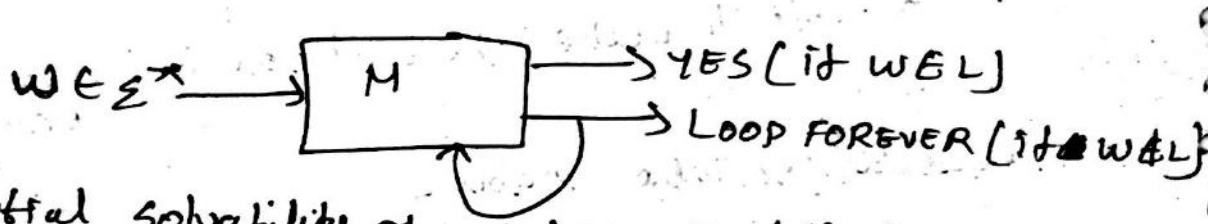
Undecidable Problem:

A problem, P is said to be undecidable if there is a Turing machine, TM that doesn't decides P.

Semi decidable | partial solvable | recursively enumerable

A problem, P is said to be semi decidable, if P is recursively enumerate.

A problem is RE if M terminates with 'YES' if it accepts $w \in L$; and doesn't halt if $w \notin L$



Partial solvability of machine is defined as

$$F_P(w) = \begin{cases} 1 & \text{if } P(w) \\ \text{undefined} & \text{if } \neg P(w) \end{cases}$$

3. Code for Turing Machine

Now we are going to generate a binary code for the Turing machine so that each Turing machine with input alphabet {0,1} may be as a binary string.

Let us assume the states $q_1, q_2, q_3, \cdots q_k$ some value of K.

For Example

$q_1 \rightarrow 0$
$q_2 \rightarrow 00$
$q_3 \rightarrow 000$ etc

1) Assume that tape symbols as $X_1 X_2 X_3 \cdots X_m$ for some value 'm' such that
   X_1 always will be symbol '0'
   X_2 always will be symbol '1'
   X_3 always will be 'B'

2) Assume the directions 'L' as D_1 and 'R' as D_2
   Left $\rightarrow L \rightarrow D_1 \rightarrow 0$
   Right $\rightarrow R \rightarrow D_2 \rightarrow 00$

we can encode the transition δ as follows

$$\delta(q_i, X_j) = (q_k, X_l, D_m)$$

$$\boxed{\text{Code for the String} = 0^i 1 0^j 1 0^k 1 0^l 1 0^m}$$

The code for the entire Turing machine M consists of all the codes for the transitions in some order

$$\boxed{\text{Complete code} = C_1 \, 11 \, C_2 \, 11 \, C_3 \, 11 \cdots 11 \, C_{N-1} \, 11 \, C_N}$$

where each C is the code for one transition of the Turing machine 'M'.

Example:

1) Find the code of the Turing Machine M be,

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

Where δ consist of the following rules,

$\delta(q_1, 1) = (q_3, 0, R)$

$\delta(q_3, 0) = (q_1, 1, R)$

$\delta(q_3, 1) = (q_2, 0, R)$

$\delta(q_3, B) = (q_3, 1, L)$

Soln:

According to the code for Turing Machine, the assumptions are,

1) The States are

$q_1 \rightarrow 0$

$q_2 \rightarrow 00$

$q_3 \rightarrow 000$

2) The Tape symbols are 0, 1, B. Assume

$0 \rightarrow x_1 \rightarrow 0$

$1 \rightarrow x_2 \rightarrow 00$

$B \rightarrow x_3 \rightarrow 000$

3) The Directions are of the forms of,

Left $\rightarrow L \rightarrow D_1 \rightarrow 0$

Right $\rightarrow R \rightarrow D_2 \rightarrow 00$

Now the transitions are,

1) $\delta(q_1, 1) = (q_3, 0, R)$

where $C_1 = 0^1 1 0^2 1 0^3 1 0^1 1 0^2$

$C_2 = 0100100010100$

2) $\delta(q_3, 0) = (q_1, 1, R)$

$C_2 = 0001010100100$

3) $\delta(q_3, 1) = (q_2, 0, R)$

$C_3 = 00010010010100$

4) $\delta(q_3, B) = (q_3, 1, L)$

$C_4 = 0001000100010010$

Complete code = $C_1 \| C_2 \| C_3 \| C_4$

$$\boxed{\text{Code} = 0100100010100 \,\|\, 0001010100100 \,\|\, 00010010010100 \,\|\, 00010010010100 \,\|\, 0001000100010010}$$

2) Write the code for the following TM transitions,

$\delta(q_1, a) = (q_1, x, R)$, $\delta(q_1, y) = (q_2, y, R)$, $\delta(q_1, x) = (q_1, 0, R)$

$\delta(q_1, 1) = (q_2, y, L)$, $\delta(q_1, y) = (q_1, y, R)$, $\delta(q_2, 0) = (q_2, 0, L)$

$\delta(q_2, x) = (q_1, x, R)$, $\delta(q_2, y) = (q_2, y, L)$, $\delta(q_3, y) = (q_3, y, R)$

$\delta(q_3, B) = (q_3, B, R)$.

---

4) Enumerating the Binary Strings:

In this binary string enumeration, we assign the integers to all the binary string, so that each string corresponds to one integer and each integer corresponds to one string.

Ex:

ϵ → First string

0 → Second string

1 → Third string

00 → Fourth string

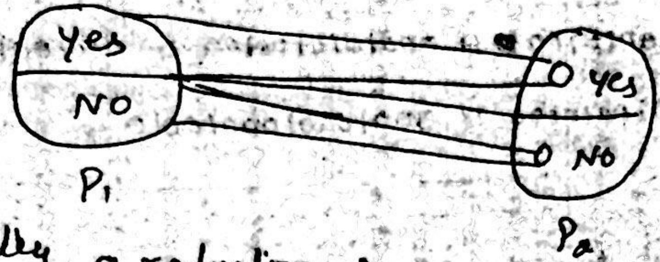
01 → Fifth string

10 → Sixth string and so on

CHOMSKIAN HIERARCHY OF LANGUAGES

5) Undecidable Problems about Turing Machine:

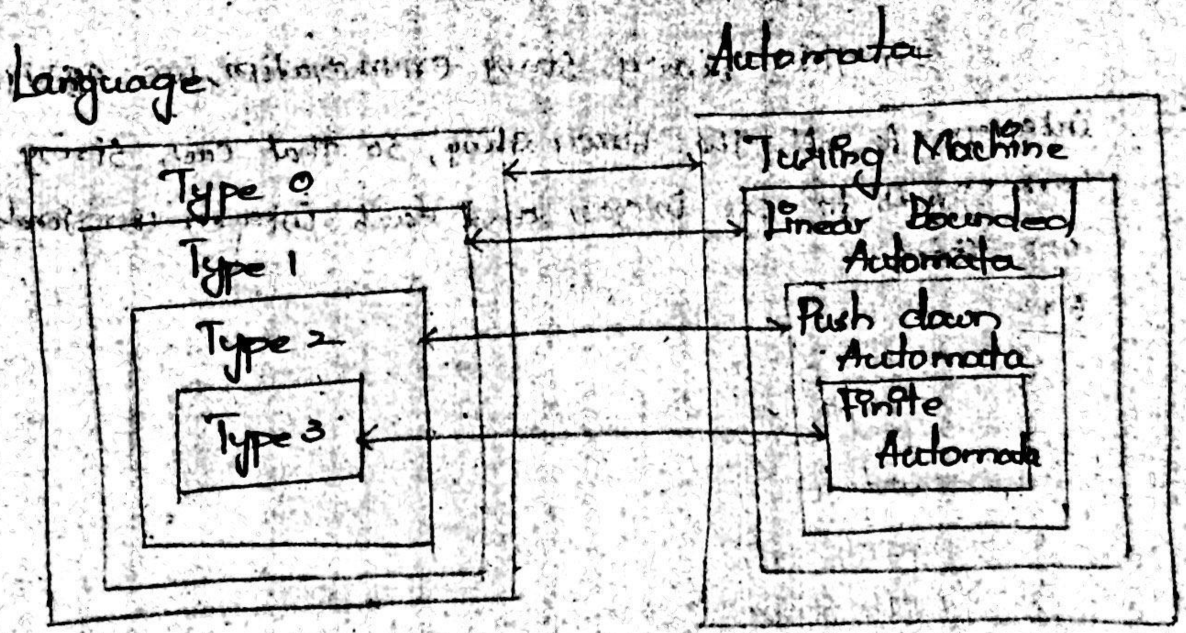
The proofs of undecidable problems about Turing machine says that any non trival property of turing machine that depends only on the language the turing machine accepts must be undecidable.



Formally, a reduction from P_1 to P_2 is a TM that takes an instance of P_1 written on its tape and halts with an instance of P_2 on its tape. so reduction takes an instance of P_1 as input and produces an instance of P_2 as output

—x—

CHOMSKIAN HIERARCHY OF LANGUAGES:



The Halting Problem

The halting problem is the problem of finding if the program/machine halts or loop forever.

The halting problem is undecidable over Turing Machines.

Ex:

```
While (1)
{
    printf("Halting problem");
}
```

The above code goes to infinite loop since the argument of while loop is true forever. Thus it doesn't halts.

Hence Turing problem is the example of undecidability.

Representation of the halting set

The halting set is represented as

$$h(M, w) = \begin{cases} 1 & \text{if } M \text{ halts on input } w \\ 0 & \text{otherwise} \end{cases}$$

where,

M → Turing Machine

w → Input string

Theorem:

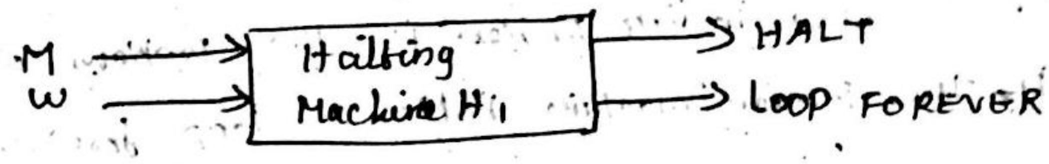
Halting problem of Turing machine is unsolvable/undecidable.

proof:

The theorem is proved by the method of proof by contradiction.

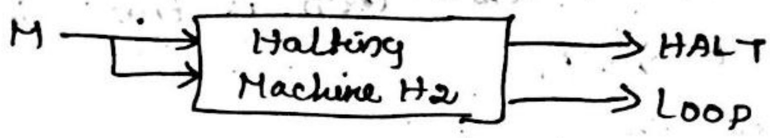
Let us assume that TM is solvable/decidable.

Construction of H_1:



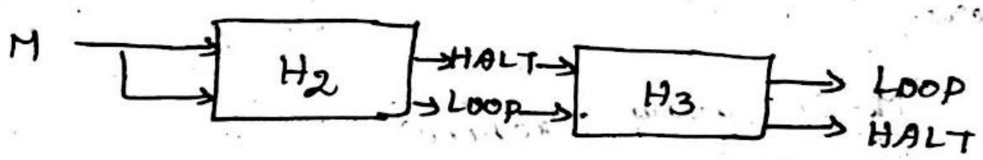
* Consider, a string describing M and i/p string, w for M.
* let H_1 generates "halt" if H_1 determines that the turing machine, M stops after accepting the i/p, w.
* otherwise H_1 loops forever when, M doesn't stop on processing w.

Construction of H_2:



* H_2 is constructed with both the inputs being M.
* H_2 determines M and halts if M halts otherwise loops forever.

Construction of H_3:



* let H_3 be constructed from the outputs of H_2
* If the o/p of H_2 are HALT then H_3 loops forever. ELSE, if the o/p of H_2 is loop forever, then H_3 halts.

Thus H_3 acts contractor to that of H_2.

Thus halting problem is undecidable.