# FORMAL LANGUAGES AND AUTOMATA THEORY

## UNIT-I

III YEAR- I SEMESTER
CSE

BY :

G M SUBHANI
Asst.Professor

MRCE

## Introduction to Defining Language:

**Alphabet**: An alphabet is a finite collection of symbols.

We cannot define symbol formally.

The example of alphabet is,

$S = \{a, b, c \text{ -----} z\}$

The elements a, b, ----- z are alphabets

$W = \{0, 1\}$

Here 0 and 1 are alphabets, denoted by W.

**String**: string is finite collection of symbols from alphabet.

For eg:- If $\Sigma = \{a, b\}$ then various strings that can be formed from $\Sigma$ are {ab, bb, aaa, bbb, ba, aba ~}

An infinite number of strings can be generated from this set.

- Empty string can be denoted by $\varepsilon$.
- The prefix of any string is any number of leading symbols of that string and suffix is any number of trailing.

**Symbols**:

For eg:- In string "0011" the prefixes can be 0, 00, 001. IIIrly suffixes can be 1, 11, 011

In string "Mango" the prefix can be 'Man' and the suffix can be 'go'.

**Language**: the language is a collection of appropriate strings.

- The language is defined using an input set. The input set is typically denoted by letter $\Sigma$.

Eg:- $\Sigma = \{\varepsilon, 0, 00, 000 \text{ ----}\}$.

Operations on string:

Various operations that can be carried out on strings are

(1). concatenation: 2 strings are combined together to form a single string.

Eg:- $x = \{white\}$ $y = \{house\}$

$xy = \{white\ house\}$

(2) Transpose : The transpose of operation is also called reverse operation.

Eg:- $(x a)^T = a(x)^T$

if $(ababbb)^T = bbb(aba)^T$

(3) Palindrome : string can be read same from left to right as well as from right to left.

Operations on language :

- Language is collection of strings.
- Hence operations that can be carried out on strings are the operations that can be carried out on language.

If L1 and L2 are two languages then,

(i) Union of two languages is denoted by $L_1 \cup L_2$

(ii) concatenation of two languages is denoted by $L_1L_2$

(iii) Intersection of two languages is denoted by $L_1 \cap L_2$

(iv) Difference of two languages is denoted by $L_1 - L_2$.

Example : Consider the string x = 110 and y = 0110. find,

(i) xy (ii) yx (iii) $x^2$ (iv) ϵy

Solⁿ : (i) xy ⇒ 110 × 0110

xy ⇒ 1100110

(ii) yx ⇒ 0110110.

(iii) $x^2$ = x . x ⇒ 110110.

(iv) ϵy means the string belonging to set y which is 0110

KLEEN CLOSURES :

- The Kleeen closure is also called star closure.
- This is set of strings of any length (including null string ε).
- Each string is obtained from input set Σ.

Example 1 : Let Σ = {a,b} obtain $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3$ ----

Solⁿ $\Sigma^*$ = { ε, a, b, aa, bb, ab, ba, aba, aaa, bbb, baba, abaab ----

is a set of strings of any length.

- The positive closure $\Sigma^+$ can be defined as $\Sigma^1 \cup \Sigma^2 \cup \Sigma^3$

that means it consists of all the strings of any length except a null string.

Example 2 : Let Σ = {a,b} obtain $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3$ ----

Solⁿ $\Sigma^*$ = {a, b, aa, bb, ab, ba, aba, aaa, bbb, baba, abaab, ----}

is a set of strings of any length except null string (ε).

$\Sigma^* = \Sigma^+ + \varepsilon$.

# FINITE STATE MACHINE

- The finite state systems represents a mathematical model of a system with certain input. The input when is given to the machine it is processed by various states, these states are called as intermediate states.

Definition :

A finite automata is a collection of 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

$Q$ = finite set of states which is non empty

$\Sigma$ = input alphabet

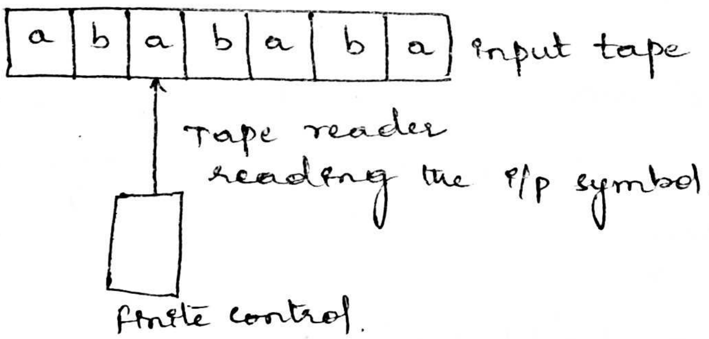
$q_0$ = initial state i,e $q_0 \in Q$

$F$ = set of final states.

$\delta$ = transition

Finite automata Model :

The finite automata can be represented as :



- The finite automata can be represented using

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

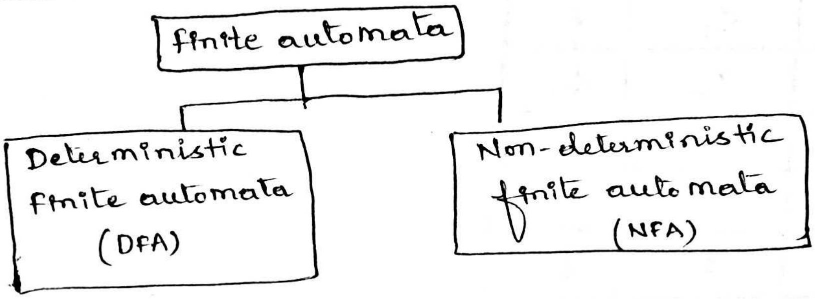
(ii) Finite control :- The finite control decides the next state on receiving particular input from input tape.

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

- The set of strings accepted by a FA given by M then M is accepted by language L. The acceptance of M by some language L is denoted by L(M).
- The machine M accepts a language L iff,

$$L = L(M).$$

TYPES OF AUTOMATA :


finite automata
Deterministic finite automata (DFA)
Non-deterministic finite automata (NFA)

The deterministic finite automata is deterministic in nature. ~~In this~~ Each move in this automata is uniquely determined on current input and current state.

NFA is non deterministic in nature.

Acceptance of strings and Languages :

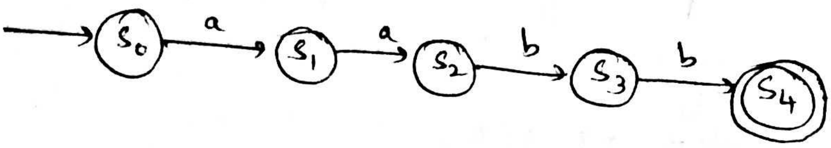
- The strings and languages can be accepted by a finite automata when it reaches to a final state.

1. Transition diagram :

It can be defined as collection of

1) Finite set of states K.
2) Finite set of symbols $\Sigma$
3) start state
4) Final state.
5) A transition function

Ex:



2. Transition Table:

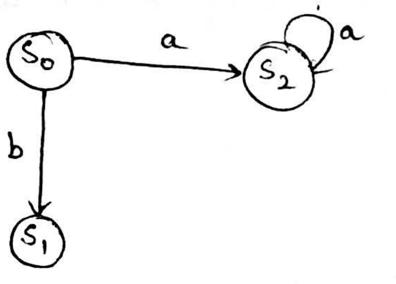
This is a tabular representation of FA.

| States \ I/P | a | b |
|---|---|---|
| $S_0$ | $S_1$ | $\phi$ |
| $S_1$ | $S_2$ | $\phi$ |
| $S_2$ | $\phi$ | $S_3$ |
| $S_3$ | $\phi$ | $S_4$ |
| $S_4$ | $\phi$ | $\phi$. |

## DETERMINISTIC FINITE AUTOMATA (DFA):

- The finite automata is called Deterministic Finite Automata, if there is only one path for a specific input from current state to next state.

For eg:- DFA can be shown as below.



From state $S_0$ for input a, there is only one path going to $S_2$. IIIly from $S_0$ there is only one path for input b going to $S_1$.

Definition of DFA :

The deterministic finite automaton is collection of following things.

1) $Q$ = finite set of states

2) $\Sigma$ = finite set of input symbols

3) $q_0$ = start state $q_0 \in Q$

4) $F$ = set of final states i,e $F \in Q$.

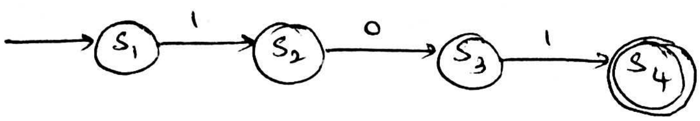
5) $\delta$ = mapping function

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

## PROBLEMS (DFA).

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

Sol^n



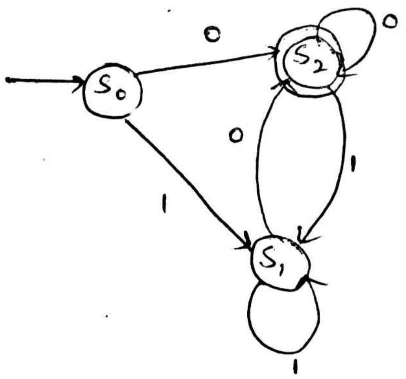
(2). Design a FA which checks whether the given binary number is even.

Sol^n The binary number is made up of 0's and 1's. When any binary number ends with 0, it is always even and when a binary number ends with 1 it is always odd.

For eg:-

0100 is an even number, it is equal to 4.
0011 is an odd number, it is equal to 3.

While designing FA, we assume one start state. one state ending in 0 and other state for ending with 1.



Let us take some input

$01000 \Rightarrow 0\underline{S_2}1000$

$\Rightarrow 01\underline{S_1}000$

$\Rightarrow 010\underline{S_2}00$

$\Rightarrow 0100\underline{S_2}0$

$\Rightarrow 01000\boxed{S_2} \rightarrow$ It is the final state.

So The string is accepted.

Let us take another input.

1011 ⇒ $1\, \underline{s_1}\, 011$

⇒ $10\, \underline{s_2}\, 11$

⇒ $101\, s_1\, 1$

⇒ $1011\, s_1$

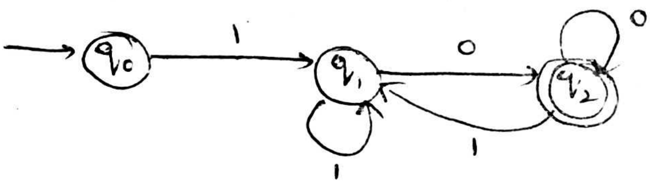
↳ It is a non final state.

Transition table :

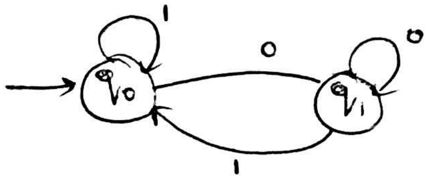
| state \ Input | 0 | 1 |
|---|---|---|
| → $S_0$ | $S_2$ | $S_1$ |
| $S_1$ | $S_2$ | $S_1$ |
| $S_2$ | $S_2$ | $S_1$ |

③ Design FA which accepts only those strings which start with 1 and ends with 0.

Soln

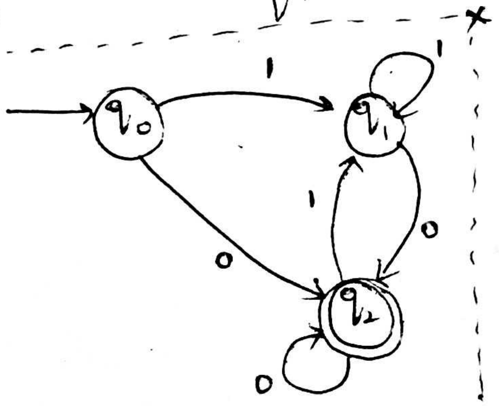


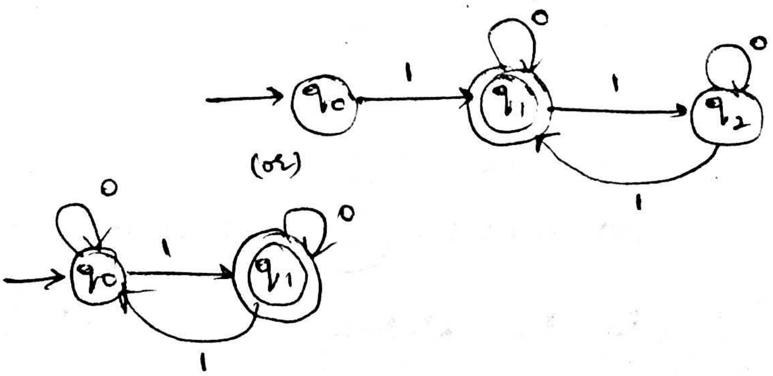
(or)



④ Design a FA which accept odd no. of 1's and any number of 0's.

soln





(or)

Take input string as 11100

$\Rightarrow 1q_1 1100$

$\Rightarrow 11q_1 100$

$\Rightarrow 111q_1 00$

$\Rightarrow 1110q_2 0$

$\Rightarrow 11100q_2$ ↳ It is the final state

So the string is accepted.

$\delta(q_0, 11100) \vdash \delta(q_1, 1100)$

$\vdash \delta(q_2, 100)$

$\vdash \delta(q_1, 00)$

$\vdash \delta(q_1, 0)$

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

$\vdash q_1$

which is final state.

Consider another string 11001

There are even no of 1s in this string.

Therefore it is not accepted

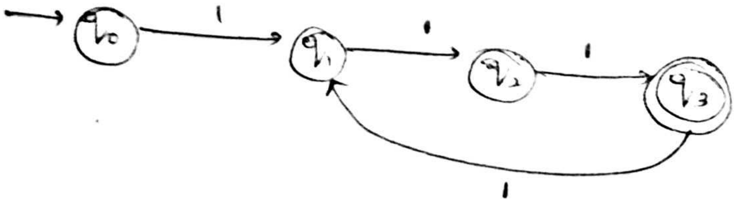
⑤ Design FA which checks whether the given unary no is divisible by 3.

soln Unary no is made up of 1's.

3 can be written as 111

5 can be written as 11111 → It is not divisible by 3.

6 can be written as 111111



⑥ Design a FA to check whether given decimal number is divisible by three.

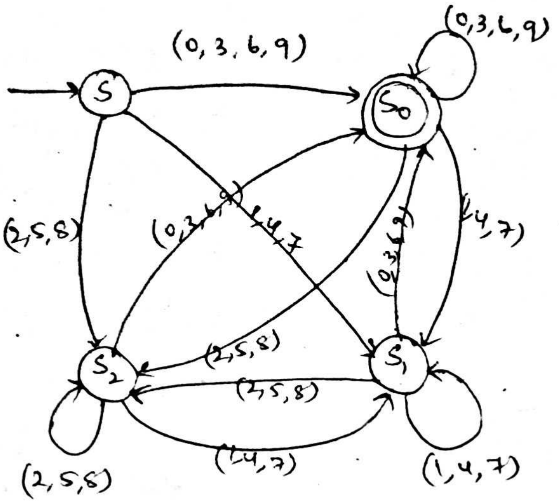
soln Here we will take the input number digit by digit.

While testing its divisibility by 3, we will get the remainder as either 0 or 1 or 2.

⇒ remainder 0 group - $S_0$ ⇒ 0, 3, 6, 9

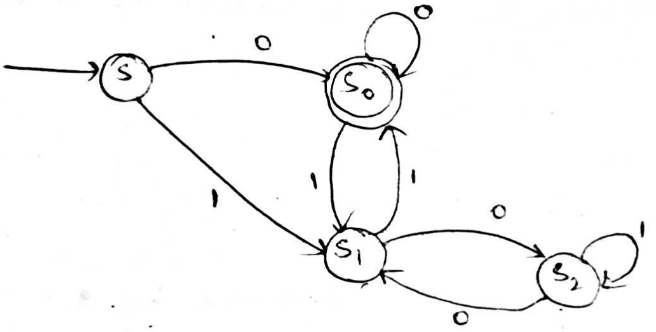
⇒ remainder 1 group - $S_1$ ⇒ 1, 4, 7

⇒ remainder 2 group - $S_2$ ⇒ 2, 5, 8.

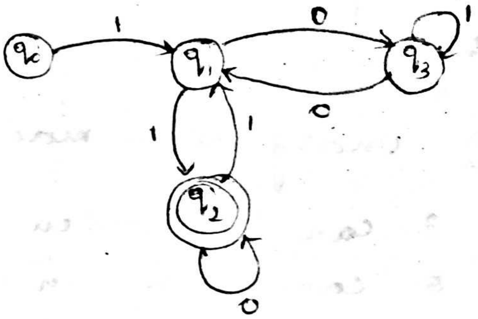


(7) Design finite Automata which checks whether a given binary no is divisible by three.

Sol<sup>n</sup>



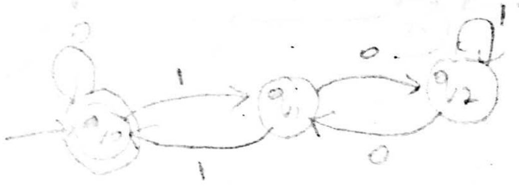
(or)



Let us take some input no 1001 → equivalent to 9.

So this is divisible by 3.

So this string should be accepted.



$1001 \Rightarrow 1\underline{S_1}001$

$\Rightarrow 10\underline{S_2}01$

$\Rightarrow 100\underline{S_1}1$

$\Rightarrow 1001S_0$ → It is the final state

– so the string is accepted.

8 4 2 1
1 0 0 0

Let us take another string 1000 → equivalent to 8.

This should not be accepted.

$1\underline{S_1}000$ $\Rightarrow 1000S_2$ → It is not the final state.

$\Rightarrow 10\underline{S_2}00$ So string is not accepted

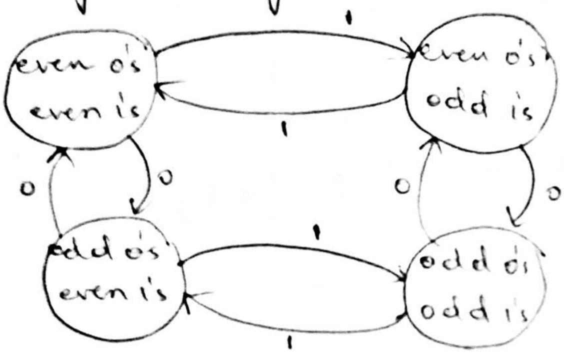
$\Rightarrow 100\underline{S_1}0$

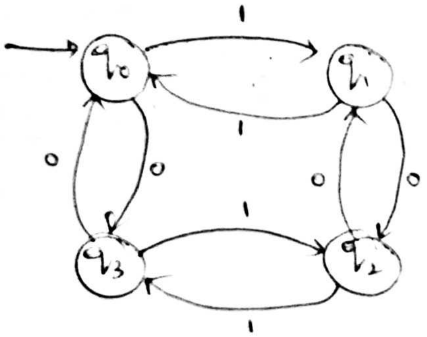
8. Design a FA which accepts even no of zeros and even no of 1's.

solⁿ In a string we may have four stages.

1) even no of 0's even no. of 1's.
2) even no of 0's odd no of 1's
3) odd no of 0's odd no of 1's
4) odd no of 0's even no of 1's.

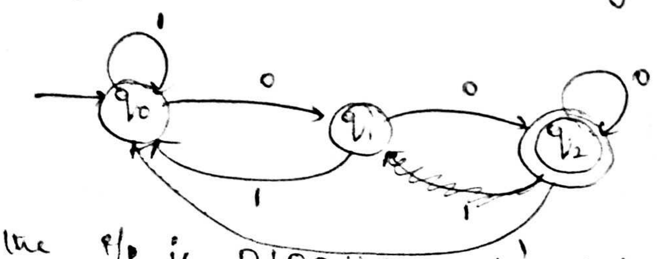
Let us try to design the machine



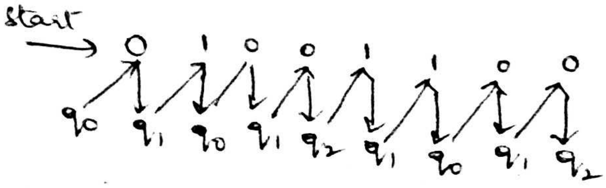


9 Design FA to accept the string that always ends with 00.

solⁿ



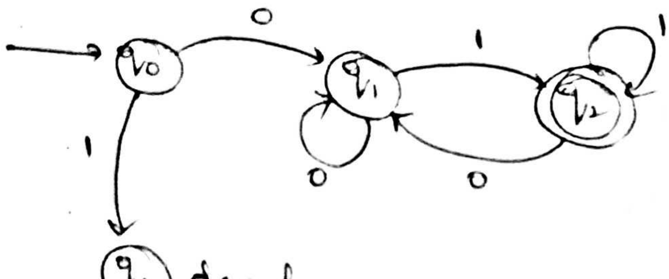
If the i/p is 01001100 it will be processed as.



$q_2$ is the final state, hence it is accepted.

(10) Construct the transition graph for a FA which accepts a language L over $\Sigma = \{0,1\}$ in which every string start with 0 and ends with 1

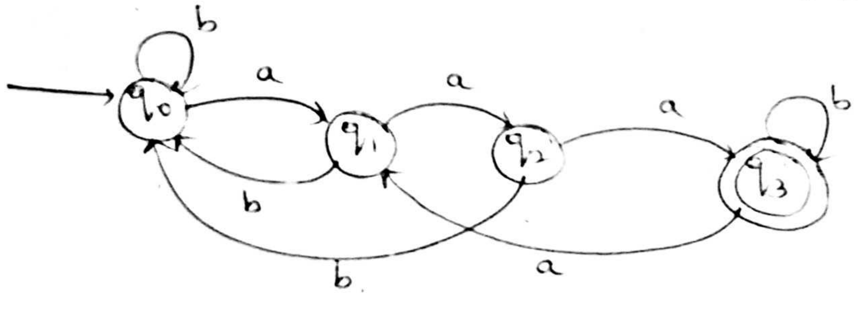
Sol^n



dead state. → Bcz the string should not start with 1.

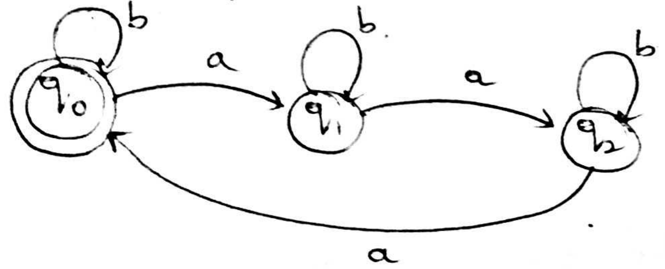
(11) Design FA to accept L where L = { strings in which a always appears trippled } over the set $\Sigma = \{a, b\}$.

Sol^n For this language the valid strings are aaab, baaaaaa, bbaaab ---- so on.



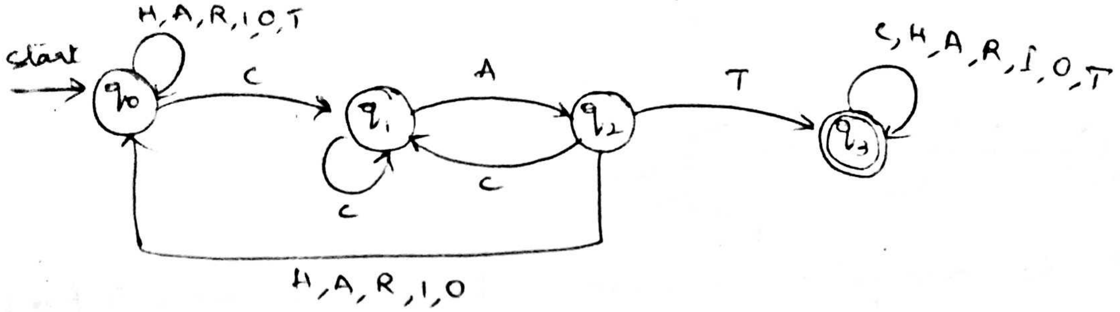
(12) Design a FA to accept L where all the strings in L are such that total no, of a's in them are divisible by 3.

Sol^n



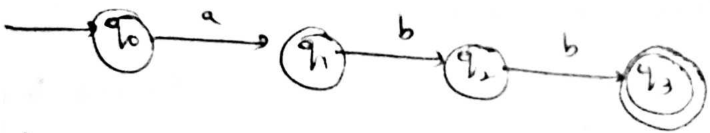
(13) Design a FA that reads strings made up of letters in the word CHARIOT and recognize those strings that contain the word CAT as a substring. (13)

Sol^n

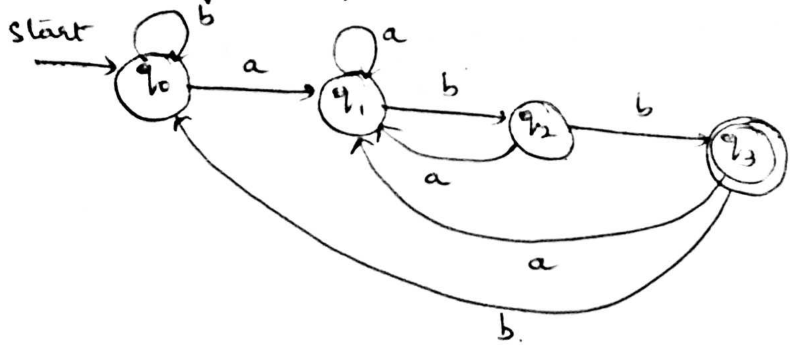


(14) Design a DFA to accept string of a's and b's ending with 'abb' over $\Sigma = \{a,b\}$

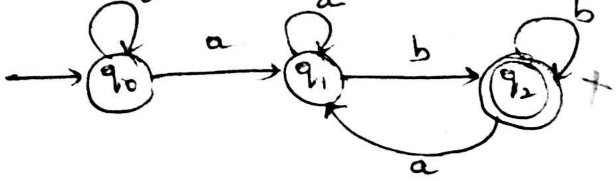
Sol^n :- Initially we will design DFA for the string abb as



We can design the complete DFA as



(or)

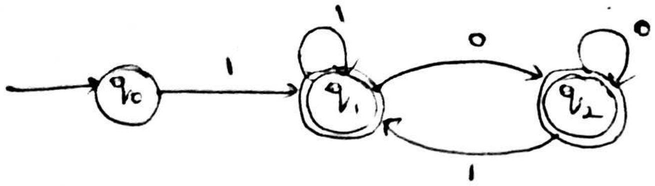


Input :- bbabb.

The string is accepted.

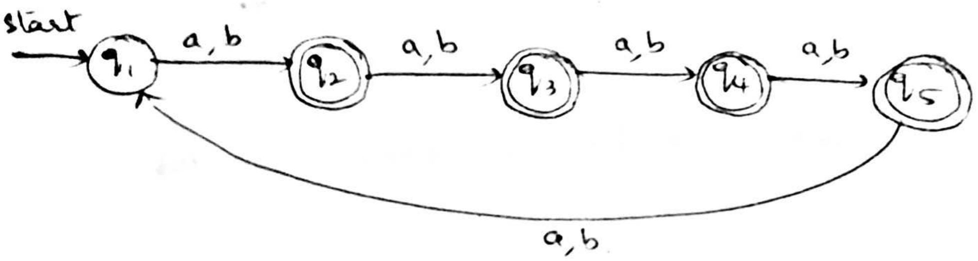
(15) Design DFA to accept odd and even numbers represented using binary notation.

solⁿ The binary no, that ends with 0 is even no, binary no, that ends with 1 is an odd no,



(16) write a DFA to accept the language $L = \{L : |w| \bmod 5 \neq 0\}$.

solⁿ The string which we obtain should not be divisible by 5. Hence the DFA will be



We can consider the string "abbb".

This string is a valid string as it is not divisible by 5. i.e abbb mod 5 ≠ 0.

- That means, on acceptance of this string, we should reach to final state.

$$\delta(q_1, abbb) \vdash \delta(q_2, bbb)$$
$$\vdash \delta(q_3, bb)$$
$$\vdash \delta(q_4, b)$$
$$\vdash \delta(q_5, \varepsilon)$$

Where $q_5$ is a final state. so the string is accepted.

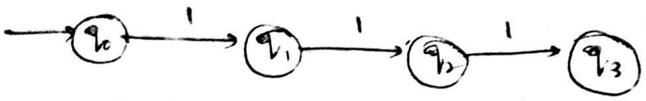
Consider another string, ababa

Since the string is divisible by 5

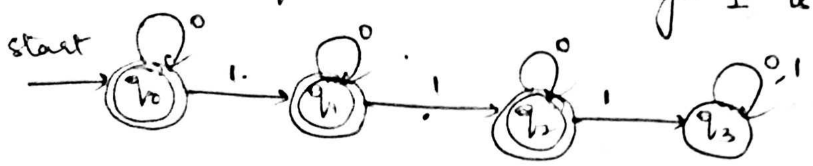
It should not be accepted.

(17) Design a DFA $L(M) = \{w \mid w \in \{0,1\}^*\}$ and w is a string that does not contain consequetive 1's.

Soln When three consecutive 1's occur, the DFA will be

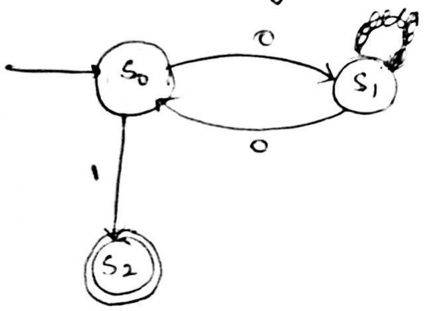


Here two consequetive 1's or single 1 is acceptable, hence



(18) Design a DFA which accepts strings with even number of 0's followed by single 1 over $\Sigma = \{0,1\}$.

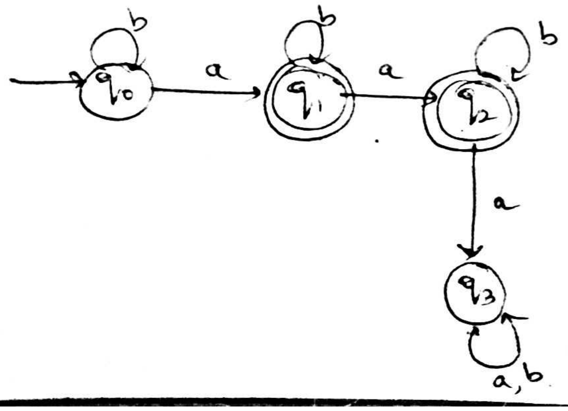
Soln



consider the input string 000101

(19) Design DFA which accepts all the strings not having more than two a's over $\Sigma = \{a,b\}$

Soln In this DFA maximum two a's are accepted.
If we try to accept third a then it should not lead to final state.
Hence the DFA will be



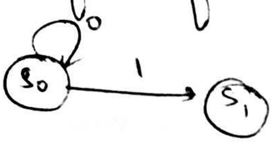
(20) Design a DFA for accepting all the strings of $\{ L = 0^m 1^n \mid m \geq 0 \text{ and } n \geq 1 \}$

solⁿ This a language in which all the o's followed by all 1's.

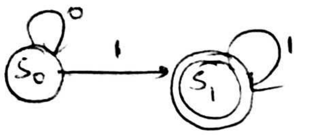
But no. of o's and no of 1's are different.

The language explicitly mentions that there should be atleast one 1.

Any no. of o's followed by only one 1 is,



But we have $1^n$ in the language, Hence the DFA is

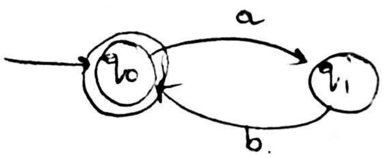


(21) Design DFA over $\Sigma = \{a, b\}$ for

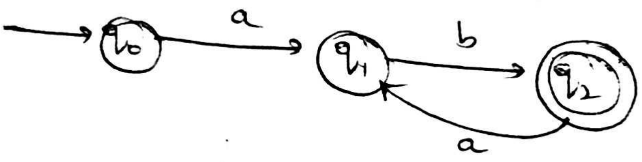
(i) $(ab)^n$ with $n \geq 0$

(ii) $(ab)^n$ with $n \geq 1$

solⁿ The DFA for (i) can be



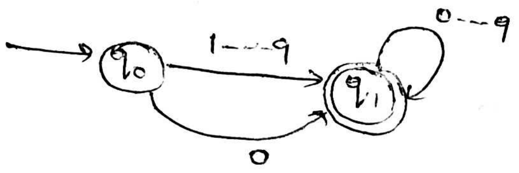
The DFA for (ii) can be



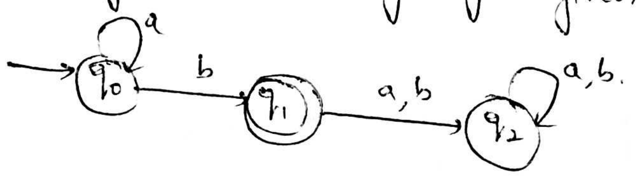
(22) Design DFA for accepting the set of integers.

Sol<sup>n</sup> For representing any integer the set {0, 1 ---- 9} is used. We can represent any integer value by taking appropriate combinations of symbols from {0 --- 9}

Hence the DFA will be



(23) Recognize the language given by following DFA.



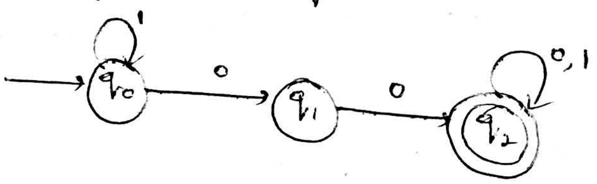
Input string ends with only one b.

b. can be followed by any no. of a's or any no of b's.

The language is denoted as $\{ L = a^n b \mid n \geq 0$

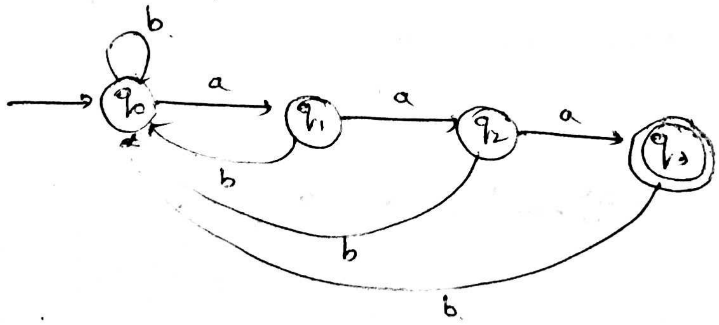
(24) Develop a DFA accepting a language given over the alphabet {0, 1}.

L = the set of all strings such that it contains atleast two 0's.



(25) Give DFA which reads strings from {a, b} and ends with aaa.

Sol<sup>n</sup>

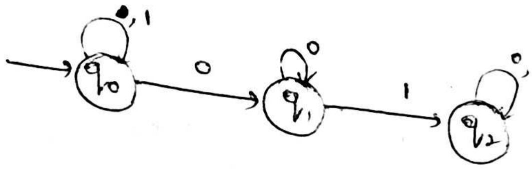


(26) Construct DFA for $L = \{w \mid w$ is in the form of '$x01y$' for some strings $x$ and $y$ consisting of 0's and 1's$\}$.

Sol$^n$ Let $L = w \in x01y$

Here $x$ and $y$ contains 0's and 1's but in b/w them there must be 01.

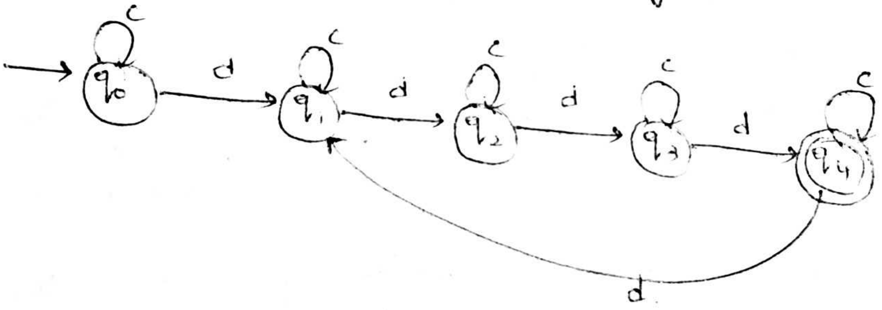
Hence DFA can be drawn as.



10101

(27) Design DFA to accept strings with c and d such that number d's are divisible by 4.

Sol$^n$

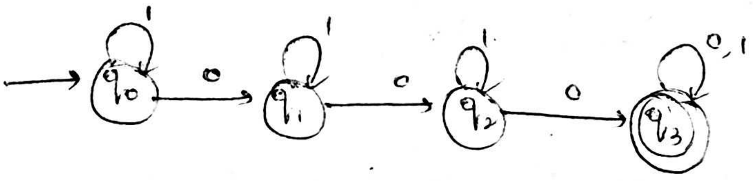


cddccdd.

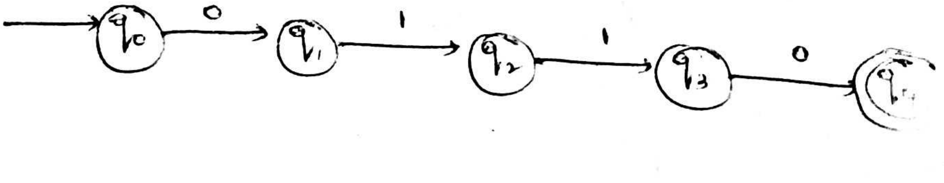
(28) Design DFA for the following over $\{0,1\}$

(a) All strings containing not more than three 0's.

Sol$^n$



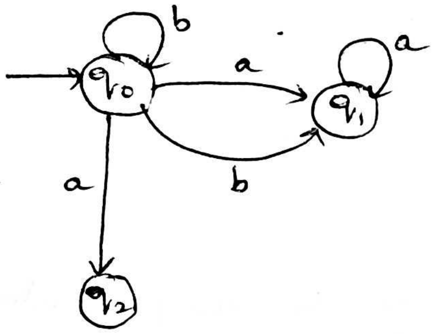
(b) All strings that has at least two occurrences of 1 between any two occurrences of 0.



# NON DETERMINISTIC FINITE AUTOMATA (NFA)

The concept of NFA is exactly reverse of Deterministic FA.
The FA is called NFA when there exists many paths for a specific i/p from current state to next state.

The NFA can be shown in fig:



Here the NFA shows from $q_0$ ~~for~~ input a there are two next states $q_1$ and $q_2$.
IIIrly from $q_0$ for input b the next states are $q_0$ and $q_1$.

- Thus it is not fixed or determined that with a particular input where to go next.

Hence this FA is called non-deterministic finite automata.

Definition of NFA:

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

1) Q = finite set of states
2) $\Sigma$ = finite set of inputs
3) $\delta$ = transition function
4) $q_0$ = initial state
5) F = final state where $F \subseteq Q$.

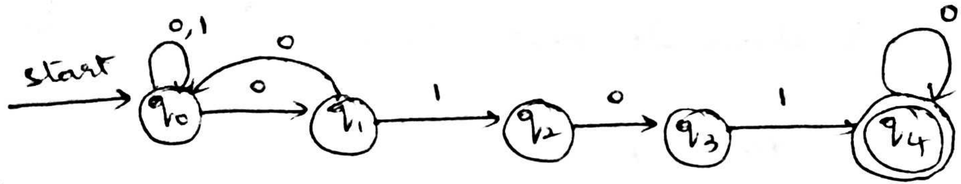
There can be multiple final states.
NFA is basically used in theory of computations, bcz they are more flexible & easier to use.

Ex 1 :- Construct a NFA for the language.

$L_1$ = { consisting a substring 0101 }

$L_2$ = { $a^n \cup b^n$ }.

Sol^n



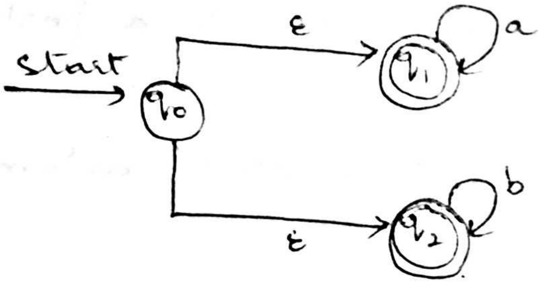
The string 00010101 is acceptable by above given NFA

Now we build NFA for $L_2$.

$L_2$ is a language in which there be any number of a's or any number of b's.

It accepts { a, b, aa, bb, aaa, bbb ---- }

Hence the NFA will be



The NFA shows two diff^t states $q_1$ and $q_2$ for the input ε from $q_0$ state.

Here ε is basically a null move i.e a move carrying no symbol from input set Σ.

Ex 2: Design the NFA transition diagram for the transition table as given below -

| | 0 | 1 |
|---|---|---|
| $q_0$ | $\{q_0\ q_1\}$ | $\{q_0\ q_2\}$ |
| $q_1$ | $\{q_3\}$ | |
| $q_2$ | $\{q_2\ q_3\}$ | $\{q_3\}$ |
| $q_3$ | $\{q_3\}$ | $\{q_3\}$ |

Here the NFA is $M = (\{q_0, q_1, q_2, q_3\}, \{0,1\}, \delta, q_0, \{q_3\})$.

Solⁿ. The transition diagram can be drawn by using the mapping function as given in table.

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

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

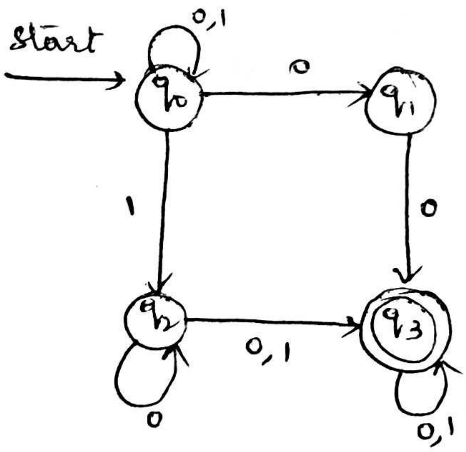
$\delta(q_1, 0) = \{q_3\}$

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

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

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

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

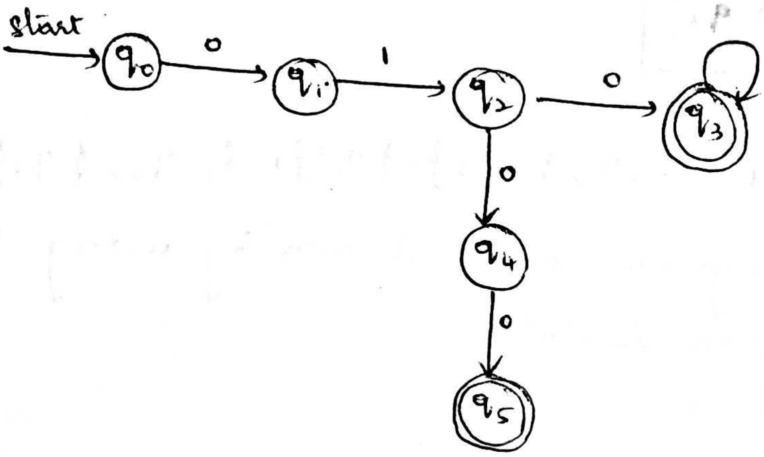


Ex : 3 Construct NFA for the language

$L = \{ 0101^n \cup 0100 \mid n \geq 0 \}$ .

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

sol<sup></sup>ⁿ Here in language L first three symbols are common i,e 010. Hence the NFA can be drawn as



The states $q_3$ and $q_5$ are final states accepting $0101^n$ and $0100$ respectively.

The NFA can be denoted by

$M = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{0, 1\}, \delta, q_0, \{q_3, q_5\})$.

Ex 4 : construct a transition diagram for the NDFA

$M = (\{q_1, q_2, q_3\}, \delta, \{q_1, \{q_3\})$ where $\delta$ is given by

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

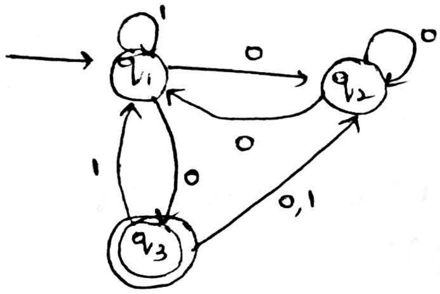
$\delta(q_2, 0) = \{q_1, q_2\}$ $\quad$ $\delta(q_2, 1) = \phi$

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

sol<sup></sup>ⁿ Firstly we will design a transition table using the given mapping function $\delta$.

| states \ Input | 0 | 1 |
|---|---|---|
| → $q_1$ | $\{q_2, q_3\}$ | $q_1$ |
| $q_2$ | $\{q_1, q_2\}$ | $\phi$ |
| $q_3$ | $q_2$ | $\{q_1, q_2\}$ |

The NDFA will be



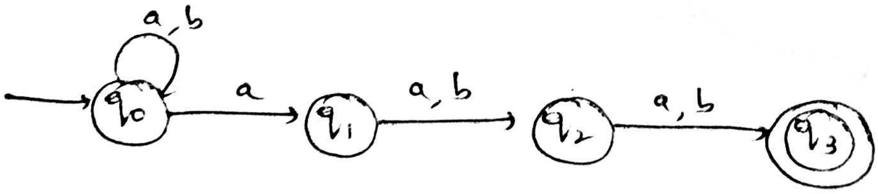
Ex : 5 Construct a NFA for a language L which accepts all the strings in which the third symbol from right end is always a. over $\Sigma = \{a, b\}$

Sol^n The strings in which such, a language are of the form.

Anything either a or b | a | a or b | a or b

Thus we get third symbol from right end as 'a' always.

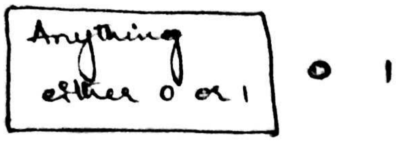
The NFA then can be



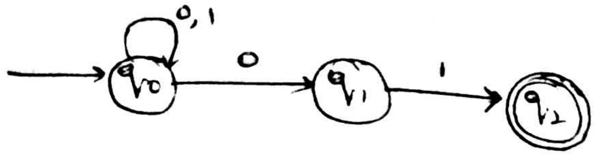
The above fig is a NFA because in state $q_0$ with i/p a we can go to either state $q_0$ or state $q_1$.

Ex-6: Design NFA accepting all strings ending with 01. over $\Sigma = \{0,1\}$.

Soln

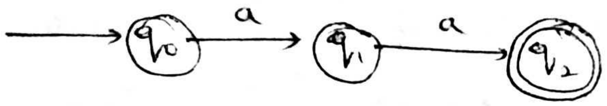


Hence, NFA would be

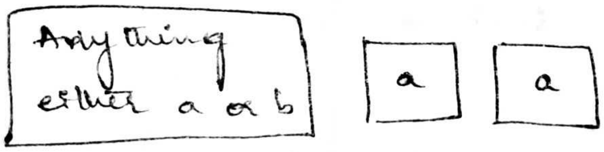


Ex-7: Design NFA to accept strings with a's and b's such that the string end with 'aa'.

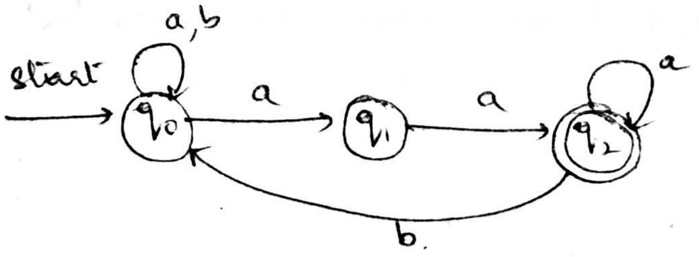
Soln The simple FA which accepts a string with 'aa' is



Now there can be a situation where in



Hence we can design the required NFA as



It can be denoted by

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

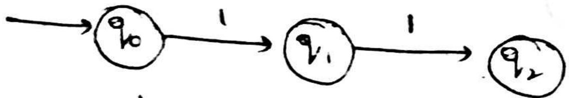
Input string:- aaa

$$\delta(q_0, aaa) \vdash \delta(q_0, aa)$$
$$\vdash \delta(q_1, a)$$
$$\vdash \delta(q_2, \varepsilon)$$

It is accepted.

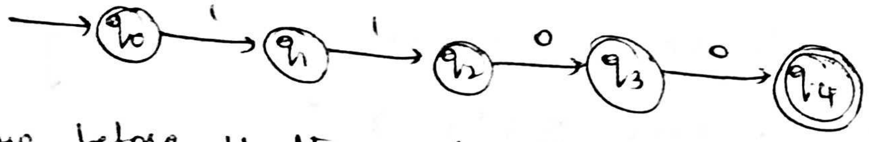
**Ex - 8:** construct a NFA in which double '1' is followed by double '0'. over $\Sigma = \{0,1\}$

**sol$^n$** The FA with double 1 is as drawn below.



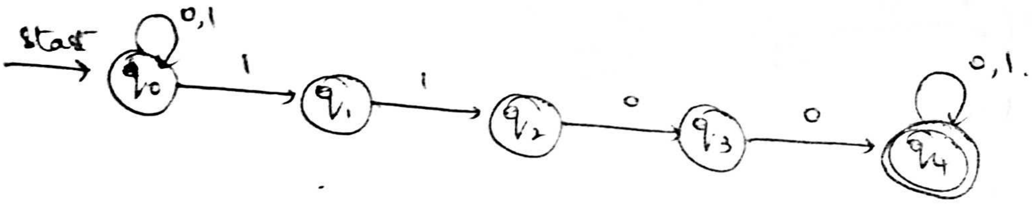
It should be immediately followed by double 0.

Then,



Now before 11 there can be any string of 0 and 1. similarly after double 0 there can be any string of 0 and 1.

Hence, the NFA becomes.



The transition table for above transition diagram can be given below.

| states\Input | 1 | 0 |
|---|---|---|
| $q_0$ | $\{q_0\, q_1\}$ | $q_0$ |
| $q_1$ | $q_2$ | - |
| $q_2$ | - | $q_3$ |
| $q_3$ | - | $q_4$ |
| $q_4$ | $q_4$ | $q_4$ |

consider string 11100,

$$\delta(q_0, 11100) \vdash \delta(q_0, 1100)$$
$$\vdash \delta(q_0, 100)$$
$$\vdash \delta(q_1, 00)$$

Got stuck! As there is no path from $q_1$ for input symbol 0.

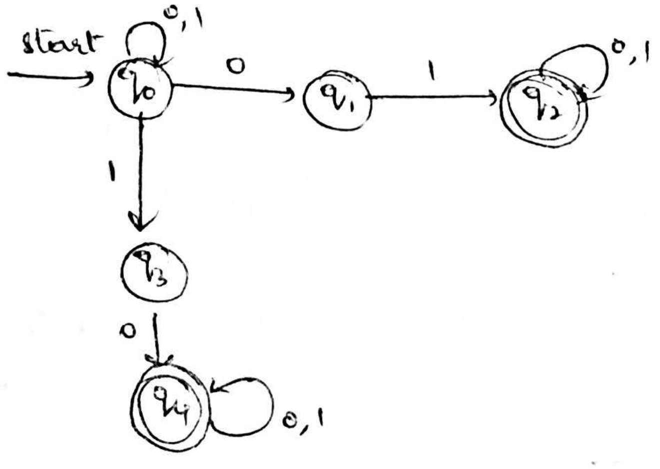
We can process string 11100 in another way.

$$\begin{aligned}
\delta(q_0, 11100) &\vdash \delta(q_0, 1100)\\
&\vdash \delta(q_1, 100)\\
&\vdash \delta(q_2, 00)\\
&\vdash \delta(q_3, 0)\\
&\vdash \delta(q_4, \varepsilon)
\end{aligned}$$

Ex-09: Design NFA which accepts the string containing either 01 or 10. over $\Sigma = \{0, 1\}$.

Soln The NFA can be drawn in stages as follows.

Consider a FA with '01' or '10'



The transition table for above transition diagram can be as given below -

| states \ Input | 0 | 1 |
|---|---|---|
| $\rightarrow q_0$ | $\{q_0\, q_1\}$ | $\{q_0\, q_3\}$ |
| $q_1$ | - | $q_2$ |
| $q_2$ | $q_2$ | $q_2$ |
| $q_3$ | $q_4$ | - |
| $q_4$ | $q_4$ | $q_4$. |

**Ex-10 :** Draw the transition diagram for below FA.

$M = \{ \{A, B, C, D\}, \{0, 1\}, \delta, C, \{A, C\} \}$

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

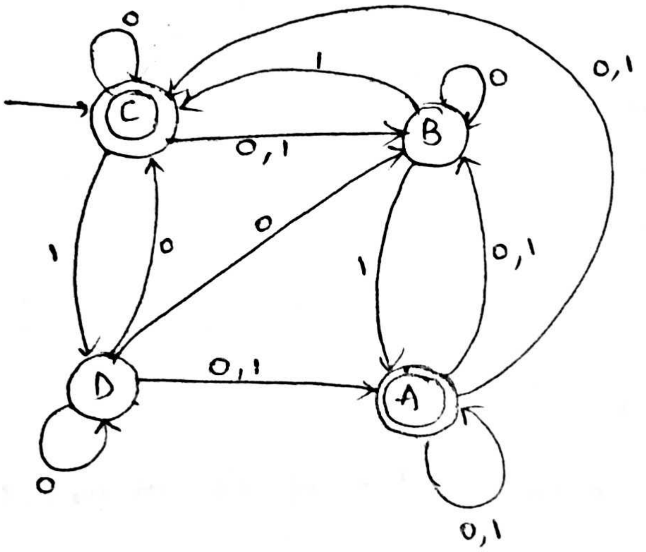
$\delta(B, 0) = B$ , $\delta(B, 1) = \{A, C\}$

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

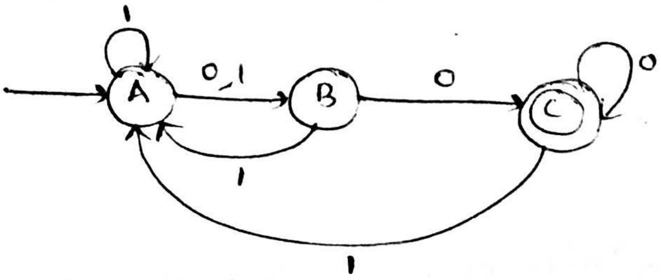
$\delta(D, 0) = \{A, B, C, D\}$

$\delta(D, 1) = \{A\}$

Sol$^n$



**Ex-11 :** Represent all five tuples for below transition and decide whether it is DFA or NFA.



Sol$^n$ The tuples can be represented by transition table -

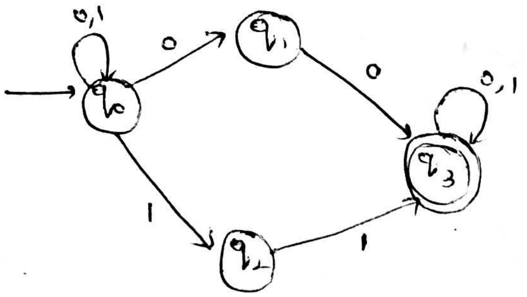
| states \ Input | 0 | 1 |
|---|---|---|
| A | B | A, B |
| B | C | A |
| C | C | A |

- As we can see that $\delta(A,1) = \{A, B\}$ that means for the input 1 there are two next states from state A.

- Hence given transitions diagram is NFA.

**Ex-12 :** Draw the transition diagram for a NFA which accepts all strings with either two consecutive 0's or two consecutive 1's.

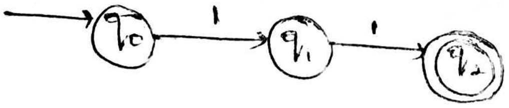
**sol^n :-**



**Ex-13 :** Provide an NFA such that

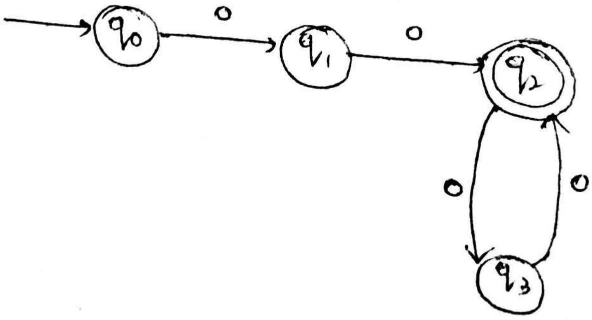
$L = \{w \mid w$ contains an even number of 0's or exactly two 1's$\}$

**sol^n** The language may contain exactly two 1's.

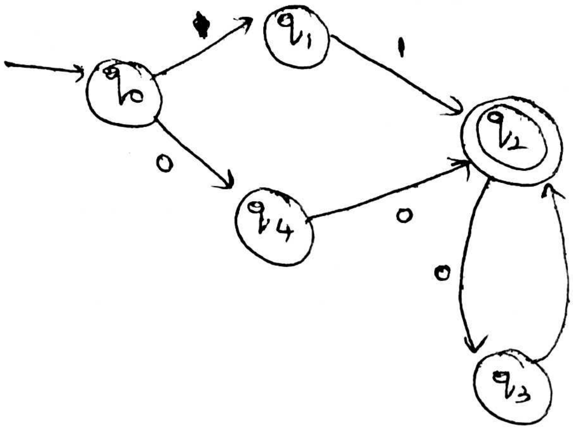


(or)

the language may contain even number of 0's.

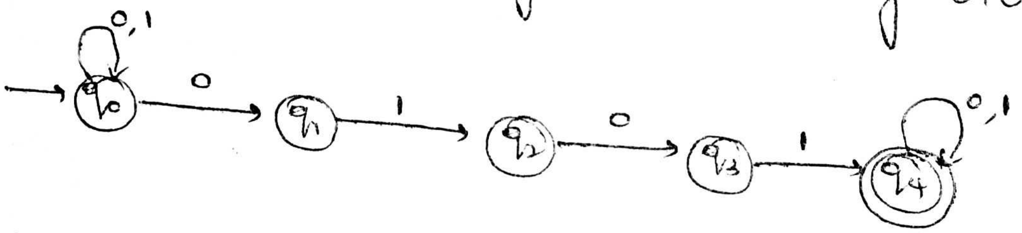


If we combine above transition diagrams then,
The finite automata will be,

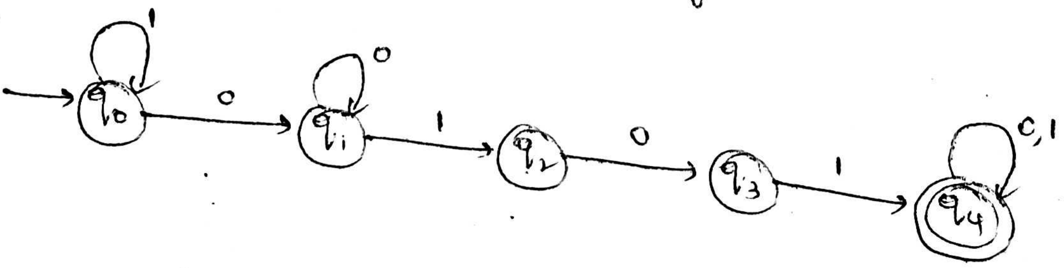


**Ex-14 :** Construct DFA and NFA for L = { w ∈ {0,1}* | w contains the substring 0101.

**sol^n** the NFA containing the substring 0101 will be



DFA containing the substring 0101 will be



10010

1001101

# An Application : Text search

(1) Finding Strings in Text :

Common problem in the age of the web and other on-line text repositories is the following.

Given a set of words, find all documents that contain one (or all) of those words. A search engine is a popular example of this process. the search engine uses a particular technology, called inverted indexes, where for each word appearing on the web a list of all the places where that word occurs is stored.

- Machines with very large amounts of main memory keep th most common of these lists available, allowing many peopl to search for documents at once.
- Inverted Index techniques do not make use of finite autom

The characteristics that make an application suitable for Searches that use automata are:

(1) The repository on which the search is conducted is rapidly changing. for example:

(a) Every day, news analysts want to search the day's online news articles for relevant topics. For example, a financial analyst might search for certain stock ticker symbols or names of companies.

(b). A "shopping robot" wants to search for the current prices charged for the items that its clients request.

The robot will retrieve current catalog pages from the Web and then search those pages for words that suggest a price for a particular item.

(2) The documents to be searched cannot be cataloged.

For eg: Amazon.com does not make it easy for crawlers to find all the pages for all the books that the company sells. Rather, these pages are generated "on the fly" in response to queries.

However, we could send a query for books on a certain topic, say "finite automata", and then search the pages retrieved for certain words eg. "Excellent" in a review portion.

NFA for Text Search:

Suppose we are given a set of words, which we shall call the keywords, and we want to find occurrences of any of these words. In applications such as these, a useful way to proceed is to design a non deterministic finite automaton, which signals, by entering an accepting state, that it has seen one of the keywords. The text of a document is fed one character at a time to this NFA, which then recognizes occurences of the keywords in this text.

There is a simple form to an NFA that recognizes a set of keywords.

1. There is a start state with a transition to itself on every input symbol. eg. every printable ASCII character if we are examining text. Intuitively, the start state represents a "guess" that we have not yet begun to see one of the keywords, even if we have seen some letters of one of these words.

2. For each keyword $a_1 a_2 --- a_k$, there are $k$ states, say $q_1, q_2, ---- q_k$. There is a transition from the start state to $q_1$ on symbol $a_1$, a transition from $q_1$ to $q_2$ on symbol $a_2$ and so on. The state $q_k$ is an accepting state and indicates that the keyword $a_1, a_2 .... a_k$ has been found.

## DFA to recognize a set of keywords:

The rules for constructing the set of DFA states is as follows:

a) If $q_0$ is the start state of the NFA, then $\{q_0\}$ is one of the states of the DFA.

b) Suppose $p$ is one of the NFA states, and it is reached from start state along a path whose symbols are $a_1 a_2 --- a_m$. Then one of the DFA states is the set of NFA states consisting of:

1. $q_0$
2. $p$.
3. Every other state of the NFA that is reachable from $q_0$ by following a path whose labels are a suffix of $a_1 a_2 --- a_m$ that is, any sequence of symbols of the form $a_j \, a_{j+1} ---- a_m$.

# UNIT - I

# FINITE - AUTOMATA

## Introduction:

- An Automaton is an abstract model of digital computer.
- An automaton has a mechanism to read input from input tape
- Any language is recognised by some automata
- Hence these automaton are basically language acceptors or language recognizers.

- The study of Theory of Automata starts with finite Automata

(I) Deterministic FA

(II) Non Deterministic FA.

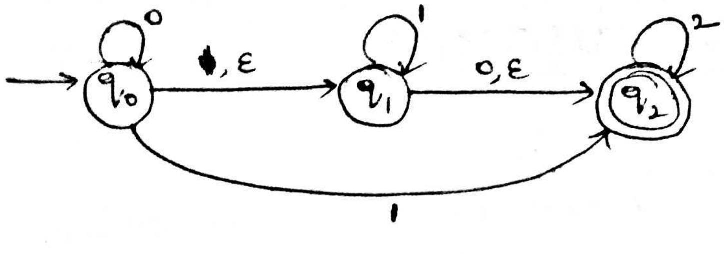
## Significance of NFA:

- Computers are basically deterministic machines
- That means on giving certain i/p, We get certain o/p either desirable or undesirable.
- It is difficult to construct deterministic machines.
- In such a case We construct a NFA or DFA.
- This NFA can be easily constructed to DFA.
- NFA serves as a bridge b/w input given and DFA constructed.

## NFA with ε transitions:

- The ε-transitions in NFA are given in order to move from one state to another without having any symbol from input set Σ.

- Consider the NFA With ε as follows.



- In this NFA with ε, $q_0$ is a start state with i/p 0.
- Here with i/p 0, we can be either in state $q_0$ or in state $q_1$
- So with ε we can change the state from $q_0$ to $q_1$.
- On the other hand, from start state $q_0$, with input 1 we can reach to state $q_2$.
- Thus it is not definite that on i/p 1 whether we will be in state $q_1$ or $q_2$.
- Hence it is called non deterministic finite automata (NFA).
- ε-moves are used to change the states from one state to another. Hence it is called NFA with ε.

Significance of NFA with ε :

- Construction of DFA is very difficult for certain i/p set. So we construct NFA.
- This NFA can be converted into DFA.
- ε is an empty string
- ε-transitions are used to change one state to another
- Some times to reach to final state we do not get proper state from start state. In such case we simply want to reach to a certain state which leads to final-state

- Such a transition to that specific state should be without any i/p symbol.
- Hence we require ε-moves by which a proper state can be obtained for reaching to final state.
- Thus ε-moves plays an imp$^{t}$ role in NFA.

## Acceptance of Language:

The language L accepted by NFA with ε, denoted by $M = (Q, \Sigma, \delta, q_0, F)$ can be defined as

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a NFA with ε

where

$Q =$ finite set of states

$\Sigma =$ input set

$\delta =$ a transition function or a mapping function. for transitions from $Q \times \Sigma$

$\Rightarrow Q \times \{\Sigma \cup \varepsilon\}$

$Q \times \{\Sigma \cup \varepsilon\} \longrightarrow 2^{Q}$.

$q_0 =$ start state

$F =$ set of final states, that $F \in Q$.

The string W in L accepted by NFA can be represented as $L(M) = \{ W \mid W \in \Sigma^*$ and $\delta$ transition for W from $q_0$ reaches to $F \}$.

ε-closure : A string w in $\Sigma^*$ will be accepted by NFA with ε-moves if there exists atleast one path - corresponding w, which starts in an initial state and ends in one of the final states, but since this path may be formed by ε-transitions as well as non ε-transitions, to find out whether w is accepted or not by the NFA with ε-moves, we require to define a function ε-closure(q), where q is the transition state.

- The function ε-closure(q) is defined as.

ε-closure(q) = set of all those states of automata which can be reached from q on a path labelled by ε.

Such that

(i) ε-closure(q) = q, where $q \in Q$

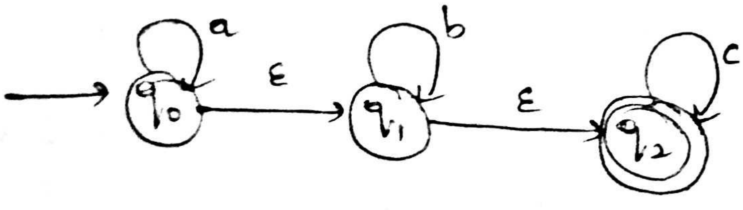
(ii) If there exists ε-closure(q) = {r} and
$\delta(r, \varepsilon) = s$ then
ε-closure(q) = {r, s}.

Example 1: construct NFA with ε which accepts a language consisting the strings of any no. of a's followed by any no. of b's followed by any number of c's.

Solⁿ: Here any no. of a's or b's or c's means zero or more than 1.

That means there can be 0 or more a's followed by zero or more b's followed by 0 or more c's.

Hence NFA with ε can be



Normally ε's are not shown in the i/p string.

Now, the transition table can be,

| state \ I/p | a | b | c | ε |
|---|---|---|---|---|
| $q_0$ | $q_0$ | $\phi$ | $\phi$ | $q_1$ |
| $q_1$ | $\phi$ | $q_1$ | $\phi$ | $q_2$ |
| $q_2$ | $\phi$ | $\phi$ | $q_2$ | $\phi$ |

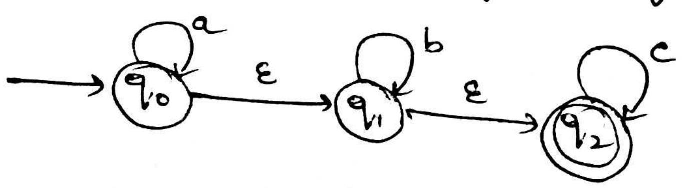
We can parse the string aabbcc as follows:

$$\delta(q_0, aabbcc) \vdash \delta(q_0, abbcc)$$
$$\vdash \delta(q_0, bbcc$$
$$\vdash \delta(q_0, \varepsilon bbcc)$$
$$\vdash \delta(q_1, bbcc)$$
$$\vdash \delta(q_1, bbc)$$
$$\vdash \delta(q_1, cc)$$
$$\vdash \delta(q_1, \varepsilon cc)$$
$$\vdash \delta(q_2, cc)$$
$$\vdash \delta(q_2, c)$$
$$\vdash \delta(q_2, \varepsilon$$

$q_2$ is the final state, which is the accept state after scanning the complete i/p string.

∴ The string is accepted.

**Example 2:** Find ε-closure for the following NFA with ε.



Solⁿ: In the NFA with ε-moves given in the fig:

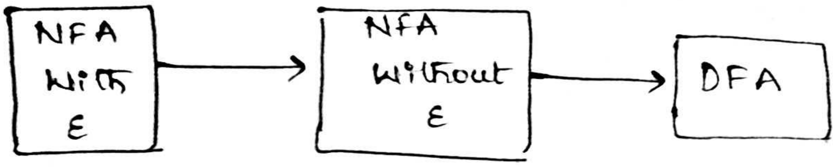
$\varepsilon$-closure($q_0$) = $\{q_0, q_1, q_2\}$ means selfstate + ε-reachable states.

$\varepsilon$-closure($q_1$) = $\{q_1, q_2\}$ $q_1$ is selfstate and $q_2$ is state obtained from $q_1$ with ε input.

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

## Conversions and Equivalence :

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



### Conversion from NFA with ε to NFA without ε.

Let $N = (Q, \Sigma, \delta, q_0, F)$ is a NFA with ε-moves, then there exists $N' = (Q, \varepsilon, \hat{\delta}, q_0, F')$ with ε moves.

1) Find ε-closure of all states in the design.

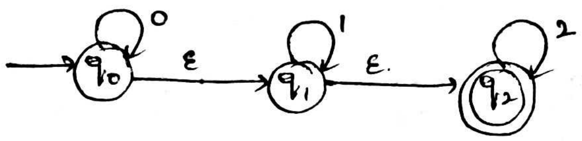
2) Calculate extended translates function using following converssion formulae.

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

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

3) F' is a set of all states whose ε-closure contains a final state in F.

Example 1 : convert the given NFA with ε to NFA without ε.



Sol$^n$ : We will first obtain ε-closure of each state i.e we will find out ε-reachable states from current state.

Hence

ε-closure $(q_0) = \{q_0, q_1, q_2\}$

ε-closure $(q_1) = \{q_1, q_2\}$

ε- closure $(q_2) = \{q_2\}$

Now we will obtain $\delta'$ transitions for each state on each i/p symbol.

$\delta'(q_0, 0) =$ ε-closure $(\delta(\hat{\delta}(q_0, \varepsilon), 0))$

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

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

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

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

= ε-closure $(q_0)$

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

$\delta'(q_0, 1) =$ ε-closure $(\delta(\hat{\delta}(q_0, \varepsilon), 1))$

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

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

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

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

= ε-closure $(q_1)$

= $\{q_1, q_2\}$.

$\delta'(q_1, 0) =$ ε-closure $(\delta(\hat{\delta}(q_1, \varepsilon), 0))$

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

= ε-closure $(\delta(q_1, q_2), 0))$

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

= ε-closure $(\phi \cup \phi)$

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

$= \phi.$

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

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

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

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

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

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

$= \{q_1, q_2\}.$

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

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

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

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

$= \phi.$

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

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

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

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

$= \phi.$

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

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

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

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

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

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

$= \{q_2\}$

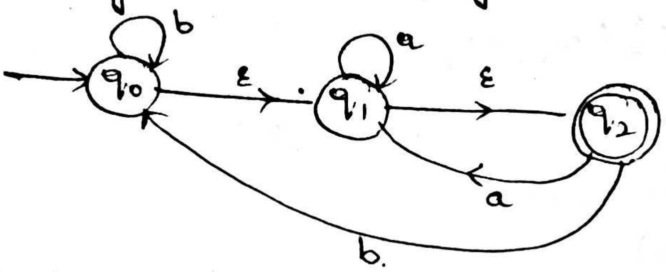
$\delta'(q_1, 2) = \varepsilon\text{-closure}(\delta(\hat{\delta}(q_1, \varepsilon), 2))$

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

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

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

ple 2: Explain the procedure to convert NFA with ε o ordinary NFA, and apply the same for the following.



First we will find ε-closure of each state.

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

ε-closure($q_1$) = {$q_1$, $q_2$}

ε-closure($q_2$) = {$q_2$}.

low δ' transitions for each state on each input

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

$= \{q_1, q_2\}$.

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

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

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

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

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

ne transition table can be

| | a | b |
|---|---|---|
| $q_0$ | {$q_1$, $q_2$} | {$q_0$, $q_1$, $q_2$} |
| $q_1$ | {$q_1$, $q_2$} | {$q_0$, $q_1$, $q_2$} |
| $q_2$ | {$q_1$, $q_2$} | {$q_0$, $q_1$, $q_2$} |

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

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

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

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

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

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

$= \{q_2\}.$

Now we will summarize all the computed $\delta'$ transitions.

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

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

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

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

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

$\delta'(q_1, 2) = \{q_2\}$

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

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

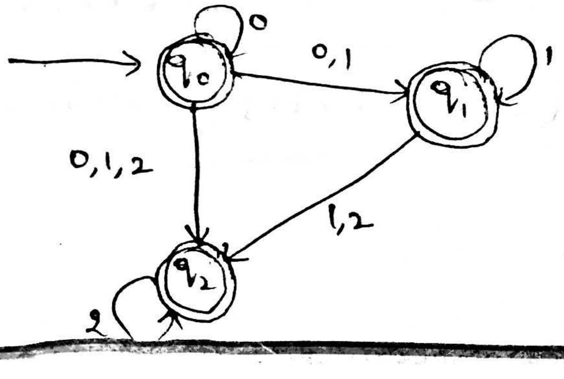
$\delta'(q_2, 2) = \{q_2\}.$

From this we can write the transition table as -

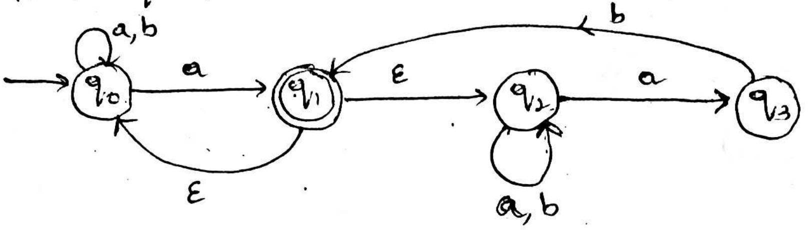
| state \ Input | 0 | 1 | 2 |
|---|---|---|---|
| $q_0$ | $\{q_0, q_1, q_2\}$ | $\{q_1, q_2\}$ | $\{q_2\}$ ✓ |
| $q_1$ | $\phi$ | $\{q_1, q_2\}$ | $\{q_2\}$ ✓ |
| $q_2$ | $\phi$ | $\phi$ | $\{q_2\}$ ✓ |

Here $q_0, q_1$ and is a final sta bcz ε-closure ε-closure$(q_1)$ a. ε-closure$(q_2)$ contains fina st $q_2$.

The NFA will be



Example 3: For the NFA - ε given check whether the string aannanan is accepted or not, if accepted write the transition path. Find equivalent NFA without epsilon transitions.



Sol$^n$: The string aannanan is not accepted by given NFA, because there is no path for input n.

The given NFA with ε can be translated to NFA without ε.

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

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

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

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

Now we will apply δ transitions for state

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

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

$= \varepsilon\text{-closure}(\delta^*(\varepsilon\text{-closure}(q_0), b))$

$= \varepsilon\text{-closure}(\delta^*(q_0, b))$

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

$= \{q_0\}$ ✓

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

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

$= \varepsilon\text{-closure}(\phi \cup q_2 \cup q_0)$

$= \varepsilon\text{-closure}(q_2) \cup \varepsilon\text{-closure}(q_0)$

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

$= \{q_2, q_0\}$

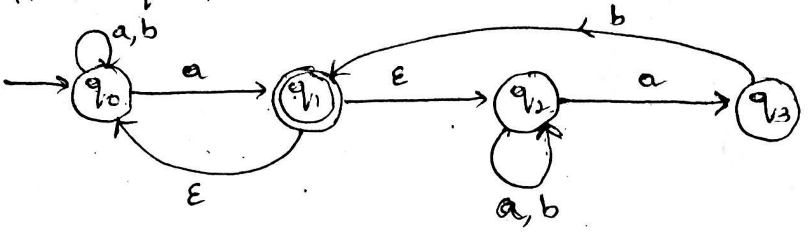
$\delta'(q_1, b) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_1), b))$

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

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

$= \varepsilon\text{-closure}(\phi \cup q_2 \cup q_0)$

**Example 3:** For the NFA - ε given check whether the string aannanan is accepted or not, if accepted write the transition path. Find equivalent NFA without epsilon transitions.



Sol$^n$: The string aannanan is not accepted by given NFA, because there is no path for input n.

The given NFA with ε can be translated to NFA without ε.

$\varepsilon$-closure$(q_0) = \{q_0\}$

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

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

$\varepsilon$-closure$(q_3) = \{q_3\}$

Now we will apply δ transitions for state

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

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

$= \varepsilon\text{-closure}(\delta^*(\varepsilon\text{-closure}(q_0), b))$

$= \varepsilon\text{-closure}(\delta^*(q_0, b))$

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

$= \{q_0\}$ ✓

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

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

$= \varepsilon\text{-closure}(\phi \cup q_2 \cup q_0)$

$= \varepsilon\text{-closure}(q_2) \cup \varepsilon\text{-closure}(q_0)$

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

$= \{q_2, q_0\}$

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

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

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

$= \varepsilon\text{-closure}(\phi \cup q_2 \cup q_0)$

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

$= \varepsilon\text{-closure}(q_0) \cup \varepsilon\text{-closure}(q_2)$

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

$= \{q_0, q_2\}.$

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

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

$= \{q_2\}.$

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

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

$= \{q_2\}.$

$\delta'(q_3, a) = \varepsilon\text{-closure}(\delta(q_3, a))$

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

$= \phi.$

$\delta'(q_3, b) = \varepsilon\text{-closure}(\delta(q_3, b))$

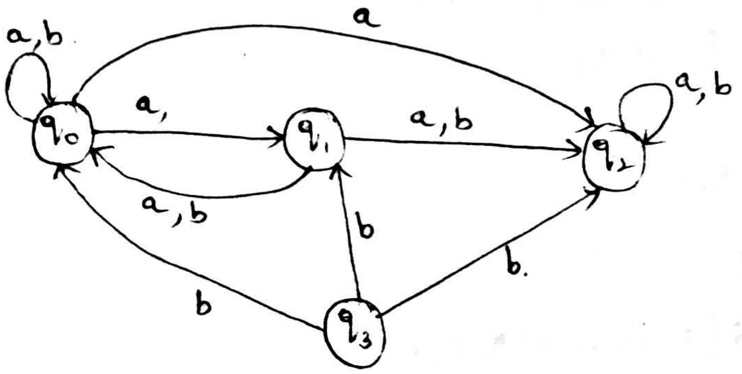
$= \varepsilon\text{-closure}(q_1)$

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

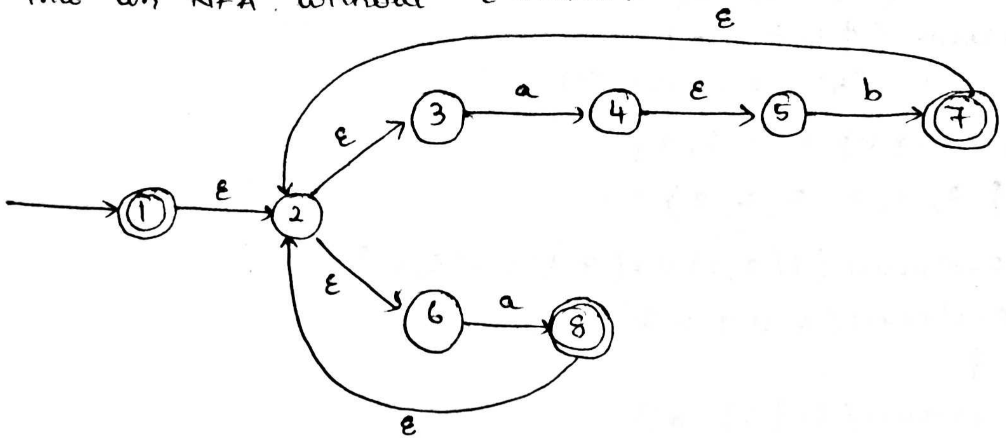
The transition table will be

| state \ Input | a | b |
|---|---|---|
| $q_0$ | $\{q_1, q_2, q_0\}$ | $\{q_0\}$ |
| $q_1$ | $\{q_0, q_2\}$ | $\{q_0, q_2\}$ |
| $q_2$ | $\{q_2\}$ | $\{q_2\}$ |
| $q_3$ | $\phi$ | $\{q_0, q_1, q_2\}$ |

Now NFA, without ε-transitions is



**Example 4 :** For the following NFA with ε-moves convert it into an NFA without ε-moves.



**Solⁿ** :- We will first find out ε closure of the states

ε-closure(1) = {1, 2, 3, 6}
ε-closure(2) = {2, 3, 6}
ε-closure(3) = {3}
ε-closure(4) = {4, 5}
ε-closure(5) = {5}
ε-closure(6) = {6}
ε-closure(7) = {2, 3, 6, 7}
ε-closure(8) = {2, 3, 6, 8}

Now we will apply input transitions on these states.

$$\delta'(1, a) = \varepsilon\text{-closure}(\delta(\hat{\delta}(1, \varepsilon), a))$$
$$= \varepsilon\text{-closure}(\delta(\{1, 2, 3, 6\}, a))$$
$$= \varepsilon\text{-closure}(\delta(1,a) \cup \delta(2,a) \cup \delta(3,a) \cup \delta(6,a))$$
$$= \varepsilon\text{-closure}(\phi \cup \phi \cup 4 \cup 8)$$
$$= \varepsilon\text{-closure}(4, 8)$$
$$= \varepsilon\text{-closure}(4) \cup \varepsilon\text{-closure}(8)$$
$$= \{4, 5\} \cup \{2, 3, 6, 8\}$$
$$= \{2, 3, 4, 5, 6, 8\}$$

$$\delta'(1, b) = \varepsilon\text{-closure}(\delta(\{1, 2, 3, 6\}, b))$$
$$= \varepsilon\text{-closure}(\delta(1,b) \cup \delta(2,b) \cup \delta(3,b) \cup \delta(6,b))$$
$$= \varepsilon\text{-closure}(\phi \cup \phi \cup \phi \cup \phi)$$
$$= \varepsilon\text{-closure}(\phi)$$
$$= \phi.$$

$$\delta'(2,a) = \varepsilon\text{-closure}(\delta(\{2,3,6\},a))$$
$$= \varepsilon\text{-closure}(\delta(2,a) \cup \delta(3,a) \cup \delta(6,a))$$
$$= \varepsilon\text{-closure}(\phi \cup 4 \cup 8)$$
$$= \varepsilon\text{-closure}(4) \cup \varepsilon\text{-closure}(8)$$
$$= \{4,5\} \cup \{2,3,6,8\}$$
$$= \{2,3,4,5,6,8\}$$

$$\delta'(2,b) = \varepsilon\text{-closure}(\delta(2,b) \cup \delta(3,b) \cup \delta(6,b))$$
$$= \varepsilon\text{-closure}(\phi \cup \phi \cup \phi)$$
$$= \phi.$$

$$\delta'(3,a) = \varepsilon\text{-closure}(\delta(\{3\},a))$$
$$= \varepsilon\text{-closure}(4)$$
$$= \{4,5\}$$

$$\delta'(3,b) = \varepsilon\text{-closure}(\phi)$$
$$= \phi.$$

$$\delta'(4,a) = \varepsilon\text{-closure}(\delta(4,a) \cup \delta(5,a))$$
$$= \varepsilon\text{-closure}(\phi \cup \phi)$$
$$= \phi.$$

$$\delta'(4,b) = \varepsilon\text{-closure}(\delta(4,b) \cup \delta(5,b))$$
$$= \varepsilon\text{-closure}(\phi \cup 7)$$
$$= \varepsilon\text{-closure}(7)$$
$$= \{2,3,6,7\}$$

$$\delta'(5,a) = \varepsilon\text{-closure}(\phi)$$
$$= \phi$$

$$\delta'(5,b) = \varepsilon\text{-closure}(7)$$
$$= \{2,3,6,7\}$$

$$\delta'(6,a) = \varepsilon\text{-closure}(8)$$
$$= \{2,3,6,8\}$$

$$\delta'(6,b) = \varepsilon\text{-closure}(\phi)$$
$$= \phi$$

$$\delta'(7,a) = \varepsilon\text{-closure}(\phi \cup 4 \cup 8 \cup \phi)$$
$$= \varepsilon\text{-closure}\{4,8\}$$
$$= \varepsilon\text{-closure}\{4\} \cup \varepsilon\text{-closure}\{8\}$$
$$= \{4,5,2,3,6,8\}$$
$$= \{2,3,4,5,6,8\}$$

$\delta'(7, b) = \phi$

$\delta'(8, a) = \{2, 3, 4, 5, 6, 8\}$

$\delta'(8, b) = \{\phi\}$.

The transition table will be

| state \ Input | a | b |
|---|---|---|
| 1 | $\{2,3,4,5,6,8\}$ | $\phi$ |
| 2 | $\{2,3,4,5,6,8\}$ | $\phi$ |
| 3 | $\{4,5\}$ | $\phi$ |
| 4 | $\phi$ | $\{2,3,6,7\}$ |
| 5 | $\phi$ | $\{2,3,6,7\}$ |
| 6 | $\{2,3,6,8\}$ | $\phi$ |
| 7 | $\{2,3,4,5,6,8\}$ | $\phi$ |
| 8 | $\{2,3,4,5,6,8\}$ | $\phi$ |

## NFA TO DFA CONVERSION

The following theorem tells that any NFA can be converted to its Equivalent DFA.

i,e any lang L acceptable by NFA can be acceptable by its equivalent DFA

Theorem: Let L be a set accepted by non-deterministic finite automaton. Then There exists a deterministic finite automaton that accepts L.

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

We construct DFA M' as follows.

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

where

1) $Q' = 2^Q$ (any state in $Q'$ is denoted by $[q_1, q_2, \dots\dots q_j]$ where $q_1, q_2 \dots\dots q_j \in Q$)

2) $q_0' = [q_0]$

$q_0$ is the initial state in NFA. It is denoted as $q_0'$ in DFA

3) $F'$ is the set of all subsets of $Q$ containing an element of $F$

4) We define,

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

if and only if

$$\delta([q_1, q_2 ---- q_i], a) = [p_1, p_2 ---- p_i]$$

* This means that whenever in NFA, at the current states $\{q_1, q_2, q_3 --- q_i\}$ if we get input a, it goes to next states $[P_1, P_2 --- P_i]$ then

* While constructing DFA for it the current state is assumed to be $[q_1, q_2, q_3 ----- q_i]$. At this state, if i/p is a, the next state is assumed to be $[P_1, P_2 ---- P_i]$

* on applying $\delta$ function on each of the states $q_1, q_2, q_3 ----- q_i$ the new states may be any of the states from $[P_1, P_2 ---- P_i]$

<u>Basis:</u> If length of input string is 0

ie $|x| = 0$, that means $x$ is $\varepsilon$ Then

$$q_0' = [q_0]$$

<u>Induction:</u> If we assume that the hypothesis is true for the ~~input~~ input string of length m or less than m.

Then if xa is a string of length m+1.

Then the $\delta'$ function could be written as

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

By the induction hypothesis,

$$\delta'(q_0', x) = [P_1, P_2 --- P_i]$$

if and only if.

$$\delta(q_0, x) = [P_1, P_2 ---- P_i]$$

By definition of $\delta'$

$\delta'([p_1, p_2, p_3 ---- p_i], a) = [r_1 r_2 ----- r_k]$

if and only if,

$\delta(\{p_1, p_2 ---- p_i\}, a) = \{r_1, r_2 ---- r_k\}$

Thus.

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

if and only if

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

is shown by inductive hypothesis.

Thus $L(M) = L(M')$

## NFA to DFA conversion procedure:

$M = (Q, \Sigma, \delta, q_0, F)$ is a NFA.

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

step 1 : The start state of NFA will be the start state for DFA. Hence add $q_0$ of NFA to $Q'$. Then find the transitions from this start state.

step 2 : For each state $[q_1, q_2 ---- q_i]$ in $Q'$ the transitions for each input symbol $\Sigma$ can be obtained as,

(i) $\delta'([q_1, q_2 ---- q_i], a) = \delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_i, a)$

$= [q_1, q_2 --- q_k]$ may be some state.

(ii) Add the state $[q_1, q_2 ---- q_k]$ to DFA if it is not already added in $Q'$

(iii) Then find the transitions for every i/p symbol from $\Sigma$ for state $[q_1, q_2 ---- q_k]$.

If we get some state $[q_1, q_2 ---- q_n]$ which is not in $Q'$ of DFA then add this state to $Q'$.

iv) If there is no new state generating, then stop the process after finding all the transitions.

③ for the state $[q_1, q_2 --- q_n] \in Q'$ of DFA if any one state $q_i$ is a final state of NFA then $[q_1, q_2 ---- q_n]$ becomes a final state.

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

Now, the $\delta'$ function will be computed as follows -

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

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

In NFA the initial state is $q_0$, so the DFA will also contain the initial state $[q_0]$

Let us draw the transition table for $\delta$ function for a given NFA.

| state \ I/P | 0 | 1 |
|---|---|---|
| $\rightarrow q_0$ | $\{q_0, q_1\}$ | $\{q_1\}$ |
| $\textcircled{q_1}$ | $\phi$ | $\{q_0, q_1\}$ |

$\delta$ Function for NFA.

→ From the transition table, we can compute that there are $[q_0], [q_1], [q_0, q_1]$ states for its equivalent DFA.

→ We need to compute the transition from state $[q_0, q_1]$

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

$= \{q_0, q_1\} \cup \phi$

$= \{q_0, q_1\}$

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

Similarly

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

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

$= \{q_0, q_1\}$

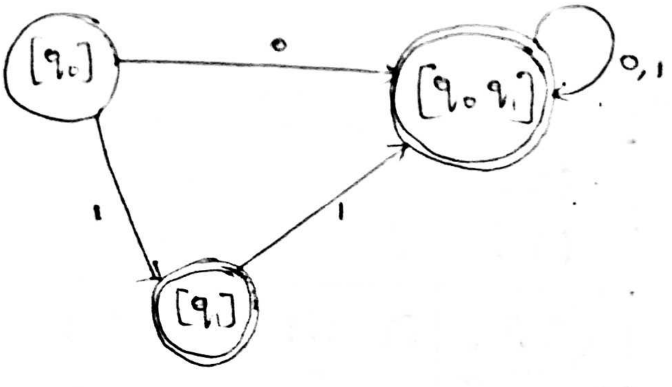
so, $\delta'([q_0, q_1], 1) = [q_0, q_1]$

- In the given NFA $q_1$ is a final state, so in DFA wherever $q_1$ exists that state becomes a final state.
- Hence in the DFA final states are $[q_1]$ and $[q_0, q_1]$
- Therefore set of final states $F = \{[q_1], [q_0, q_1]\}$

The equivalent DFA is

| state \ i/p | 0 | 1 |
|---|---|---|
| $\rightarrow [q_0]$ | $[q_0 q_1]$ | $\{q_1\}$ |
| $[q_1]$ | $\phi$ | $\{q_0 q_1\}$ |
| $[q_0 q_1]$ | $[q_0, q_1]$ | $[q_0 q_1]$ |

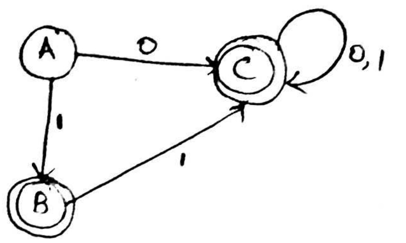
Transition table for equivalent DFA



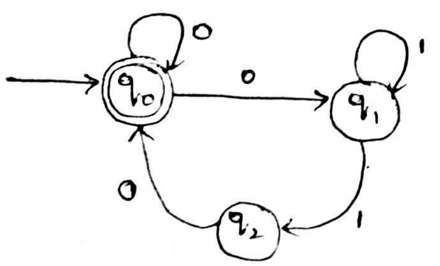
Even we can change the names of the states of DFA

$A = [q_0]$, $B = [q_1]$, $C = [q_0, q_1]$

With these new names the DFA will be as follows:



Problem 2: Convert the given finite automaton into its deterministic equivalence.



| | 0 | 1 |
|---|---|---|
| → $q_0$ | $\{q_0, q_1\}$ | $\phi$ |
| $q_1$ | $\phi$ | $\{q_1\ q_2\}$ |
| $q_2$ | $q_0$ | $\phi$. |

(1) Initial state is $q_0$, so the initial state in DFA is also $[q_0]$

(2) $Q' = 2^Q$

$Q' = \{\phi, \underline{q_0}, q_1, q_2, \{\underline{q_0, q_1}\}, \{q_1, q_2\}\}$ Here $q_0, [q_0\ q_1]$ are final states.

Applying i/p transitions on these states.

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

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

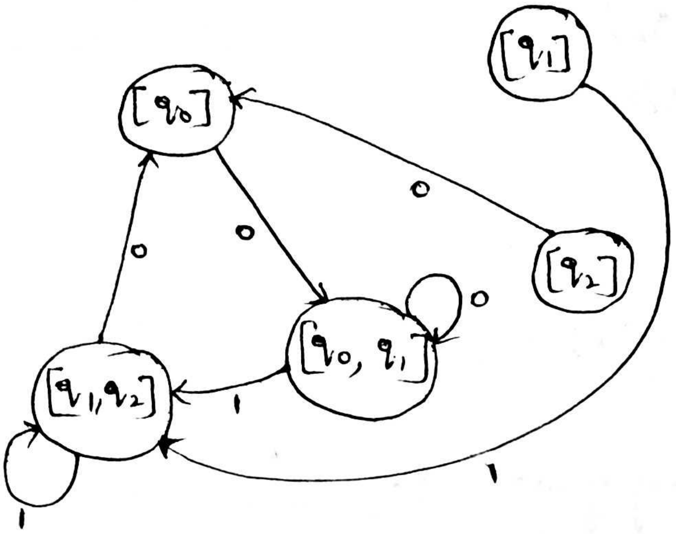
$\delta([q_1, q_2], 0) = [q_0]$

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

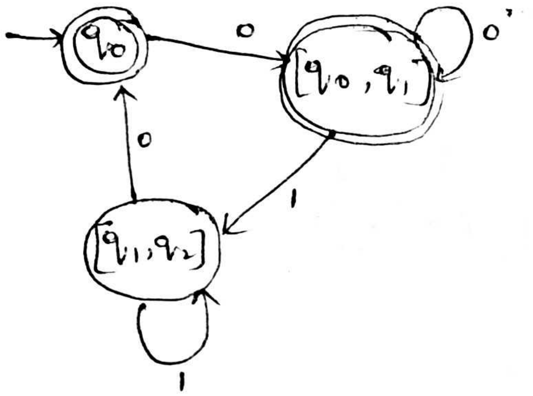
Now the transition table will be.

| state \ i/p | 0 | 1 |
|---|---|---|
| → $[q_0]$ | $[q_0, q_1]$ | $\phi$ |
| $[q_1]$ | $\phi$ | $[q_1, q_2]$ |
| $[q_2]$ | $[q_0]$ | $\phi$ |
| $[q_0, q_1]$ | $[q_0, q_1]$ | $[q_1, q_2]$ |
| $[q_1, q_2]$ | $[q_0]$ | $[q_1, q_2]$ |

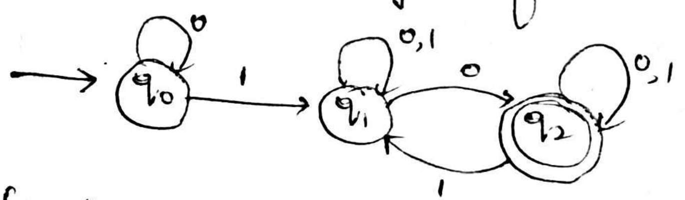
The transition diagram will be



The state $q_1$ and $q_2$ is a dead state and those can be eliminated. Hence DFA will be



PROBLEM: 3: convert the given finite automaton into its deterministic.



sol^n: for the given state diagram, the transition table is

|  | 0 | 1 |
|---|---|---|
| $q_0$ | $q_0$ | $q_1$ |
| $q_1$ | $q_1,q_2$ | $q_1$ |
| $q_2$ | $q_2$ | $q_1\ q_2$ |

$\delta(q_0, 0) = q_0$✓ $\delta(q_1, 0) = q_1\ q_2$ $\delta(q_2, 0) = q_2$✓

$\delta(q_0, 1) = q_1$✓ $\delta(q_1, 1) = q_1$✓ $\delta(q_2, 1) = q_1\ q_2$ → New state.

We will find δ' transitions for the new state $q_1 q_2$

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

$= [q_1,q_2] \cup [q_2]$

$= [q_1,q_2]$

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

$= [q_1] \cup [q_1,q_2]$

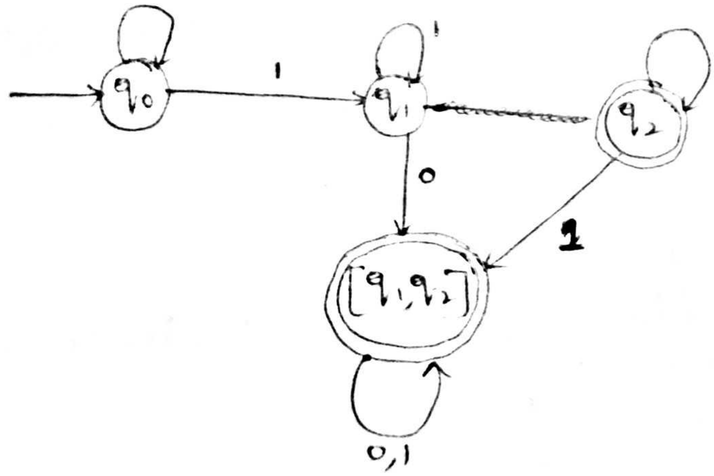
$= [q_1,q_2]$

There is no new state getting generated.

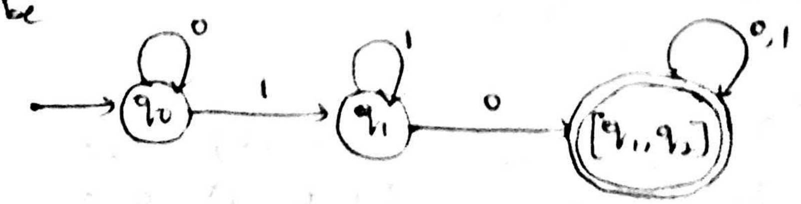
The transition table will be -

| | 0 | 1 |
|---|---|---|
| $[q_0]$ | $[q_0]$ | $[q_1]$ |
| $[q_1]$ | $[q_1,q_2]$ | $[q_1]$ |
| $[q_2]$ | $[q_2]$ | $[q_1,q_2]$ |
| $[q_1,q_2]$ | $[q_1,q_2]$ | $[q_1,q_2]$ |

The transition diagram will be



We cannot reach from start state to $q_2$ ie final state so the state $q_2$ is eliminated. The minimized DFA will be



## CONVERSION OF NFA WITH ε TO DFA :

Method for converting NFA with ε to DFA

step 1 : Consider $M = (Q, \Sigma, \delta, q_0, F)$ is a NFA with ε.

We have to convert this NFA with ε to equivalent DFA denoted by

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

Then obtain

ε-closure $(q_0) = \{P_1, P_2, P_3 \dots P_n\}$ then

$[P_1, P_2 \dots P_n]$ becomes a start state of DFA.

Now $[P_1 P_2 \dots P_n] \in Q_D$

step 2 : We obtain δ transitions on $[P_1 P_2 \dots P_n]$ for each i/p.

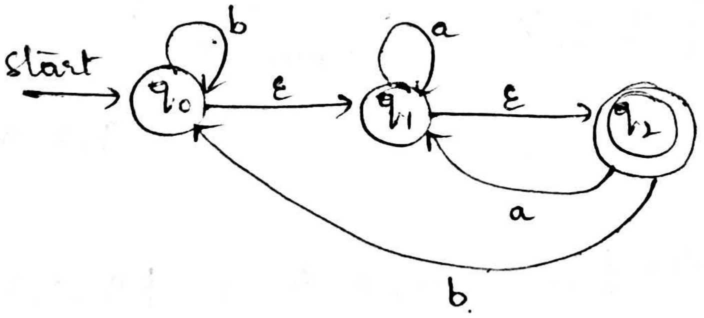
$$\delta_D([P_1 P_2 \dots P_n], a) = \epsilon\text{-closure}(\delta(P_1; a) \cup \delta(P_2, a) \cup \dots \delta(P_n, a))$$

$$= \bigcup_{i=1}^{n} \epsilon\text{-closure}\, d(P_i, a)$$

step 3 : The states obtained $[P_1 P_2 \dots P_n] \in Q_D$

The states containing final state in $P_i$ is a final state in DFA.

**PROBLEM 1 :** Convert the following NFA with ε to equivalent DFA.



sol$^n$ : To convert this NFA, first we find ε-closures.

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

ε-closure $\{q_1\} = \{q_1, q_2\}$

ε-closure $\{q_2\} = \{q_2\}$.

Let us start from ε-closure of start state (i.e $q_0 = \{q_0, q_1, q_2\}$)

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

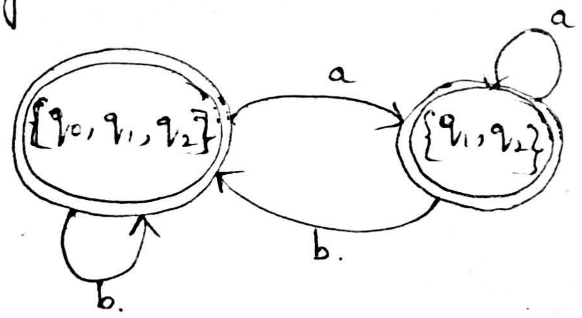
$$\delta'(\{q_0, q_1, q_2\}, b) = \varepsilon\text{-closure}(\delta(\{q_0, q_1, q_2\}, b))$$
$$= \varepsilon\text{-closure}(q_0 \cup \phi \cup q_0)$$
$$= \varepsilon\text{-closure}(q_0)$$
$$= \{q_0, q_1, q_2\}$$

~~δ'(~~

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

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

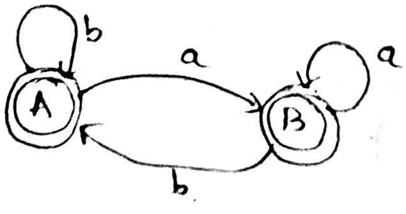
The generated DFA is



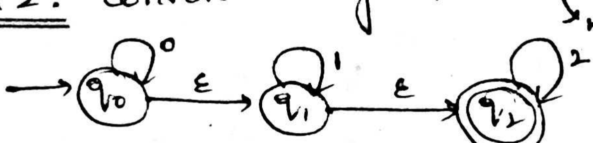
Since we hv $q_2$ in both the states. Both $\{q_0, q_1, q_2\}$ & $\{q_1, q_2\}$ are final states.

Let us assume the states $\{q_0, q_1, q_2\} = A$ & $\{q_1, q_2\} = B$.

Now the minimized DFA will be.



PROBLEM 2: convert the given NFA with ε into its equivalent DFA.



Sol^n First find ε-closures of each state.

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

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

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

Now let us start with ε-closure of start state i.e

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

find δ' transitions for the state $\{q_0, q_1, q_2\}$

$$\delta'(\{q_0,q_1,q_2\},0) = \cancel{\delta'}(\ \varepsilon\text{-closure}\{\delta((q_0,q_1,q_2),0)\}$$
$$= \varepsilon\text{-closure}\{q_0 \cup \phi \cup \phi\}$$
$$= \{q_0, q_1, q_2\}$$

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

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

Now find δ' transitions for the state $\{q_1, q_2\}$

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

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

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

Consider $\{q_0, q_1, q_2\}$ to be state A and $\{q_1, q_2\}$ to be state B. and $\{q_2\}$ to be state C

The transitions are

$\delta'(A,0) = A$ $\quad$ $\delta'(B,0) = \phi$

$\delta'(A,1) = B$ $\quad$ $\delta'(B,1) = B$

$\delta'(A,2) = C$ $\quad$ $\delta'(B,2) = C$

δ' transitions for C are

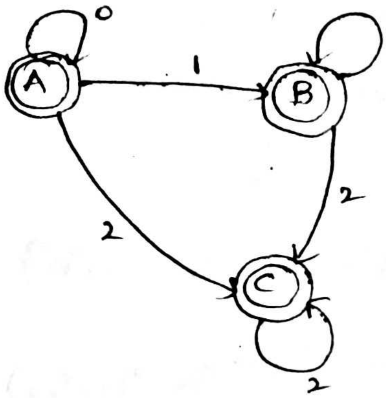
$\delta'(C,0) = \varepsilon\text{-closure}(\delta(q_2,0))$
$= \phi.$

$\delta'(C,1) = \varepsilon\text{-closure}(\phi)$
$= \phi$

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

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

Hence the DFA is



$q_2$ is final state in NFA

$q_2$ is there in states $(q_0, q_1, q_2)$, $\{q_1, q_2\}$

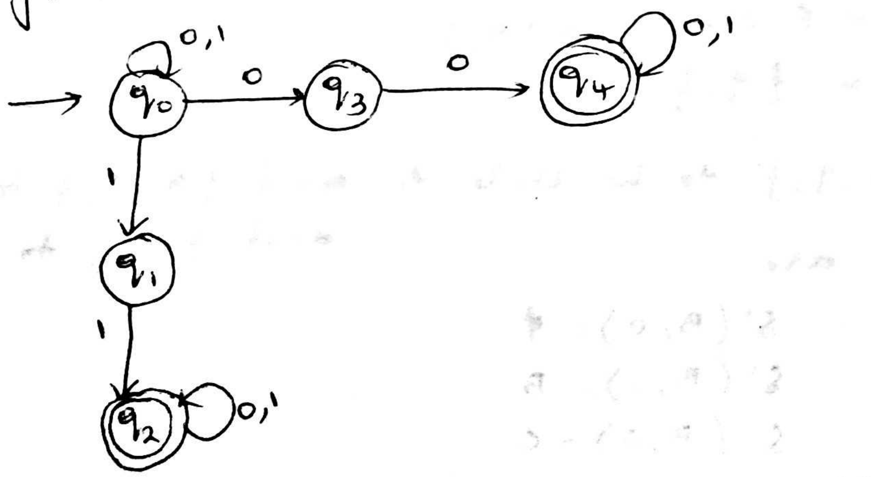
∴ The states A, B and C are final states.

NFA TO DFA

xtra: construct a DFA eqvt to $M = (\{q_0 q_1 q_2 q_3\}, \{0,1\}, \delta, q_0, \{q_3\}\}$ where δ is given by the fig.

| | 0 | 1 |
|---|---|---|
| → $q_0$ | $\{q_0 q_1\}$ | $\{q_0\}$ |
| $q_1$ | $\{q_2\}$ | $\{q_1\}$ |
| $q_2$ | $\{q_3\}$ | $\{q_3\}$ |
| ⓠ$_3$ | – | $\{q_2\}$ |

or the given NFA. construct its equivalent DFA



# FINITE AUTOMATA WITH OUTPUT

- The finite automata is a collection of $(Q, \Sigma, \delta, q_0, F)$ where Q is a set of states including $q_0$ as a start state.
- In finite automata reading the input string if we get final state then the string is said to be "acceptable."
- If we do not get final state then it is said that the string is "rejected".
- That means there is no need of o/p for the FA.
- But if there is a need for specifying output, other than accepted or rejected, then in such a case we require FA along with output.

There are two types of FA with output.

(1) Moore Machine

(2) Mealy Machine.

## 1// Moore Machine

* Moore Machine is a finite state machine in which the next state is decided by current ~~state~~ input symbol.

* The o/p symbol at a given time depends only on the present state of the machine.

* The formal definition of Moore machine is,

"Moore machine is a six tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where

Q is finite set of states.

$\Sigma$ is finite set of i/p symbols.

$\Delta$ is an o/p alphabet.

$\delta$ is an transition function such that $Q \times \Sigma \rightarrow Q$.

This is also known as state function.

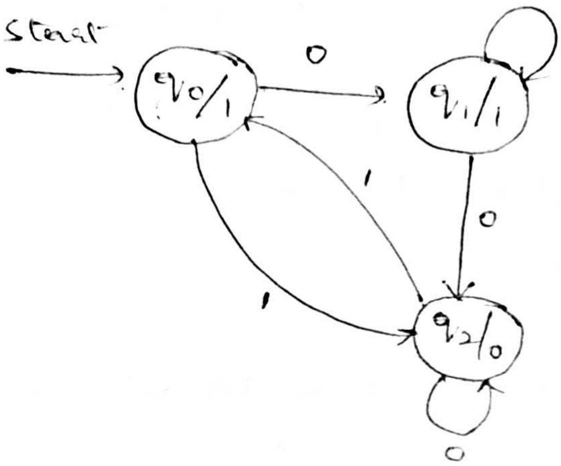
$\lambda$ is output function $Q \rightarrow \Delta$.

This function is also known as machine function.

$q_0$ is the initial state of machine.

For eg:

Consider the Moore machine given below.

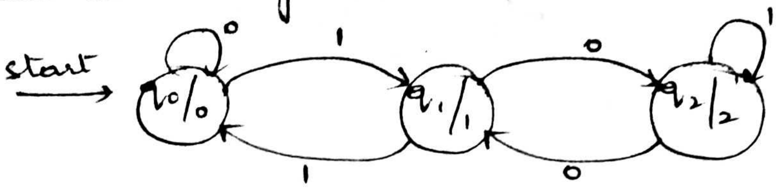


The transition table will be -

| Current state | Next state ($\delta$) | | output ($\lambda$) |
|---|---|---|---|
| | 0 | 1 | |
| $q_0$ | $q_1$ | $q_2$ | 1 |
| $q_1$ | $q_2$ | $q_1$ | 1 |
| $q_2$ | $q_2$ | $q_0$ | 0 |

- In Moore machine output is associated with every state.
- In the above given Moore machine, when machine is in $q_0$ state the output will be 1.
- For moore machine if the length of i/p string is n Then o/p string has length n+1.

Consider another Eg:-

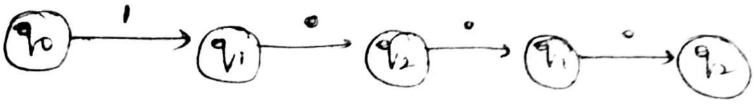


Moore machine.

Moore machine: An automaton in which the output depends only on states of the machine is called a Moore machine.

- In Moore machine o/p depends upon present state and not on present i/p.

- for the i/p string 1000 for the above machine, the transition states is given by



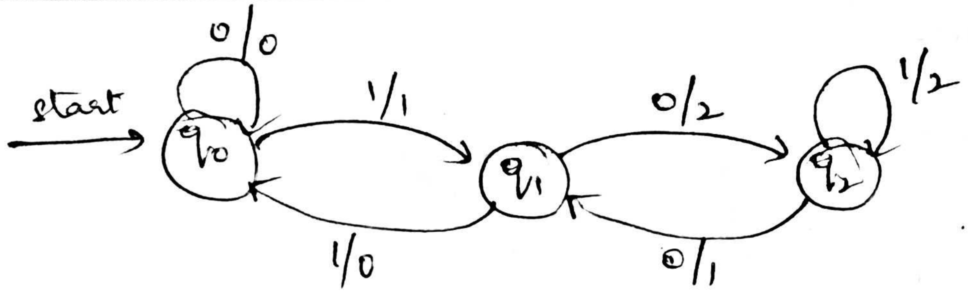
For q0 output is 0 1 2 1 2

so the o/p string is 01212.

Mealy machine: An automaton in which the output depends on the state and the input at any instant of time is called Mealy machine.

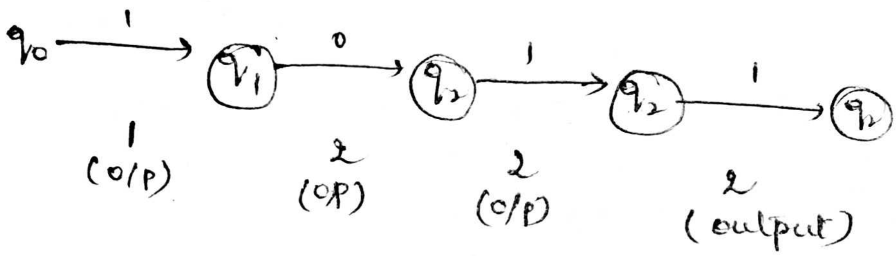
Example:

| Present state | Next state |  |  |  |
|---|---|---|---|---|
|  | Input a = 0 |  | Input a = 1 |  |
|  | state | o/p | state | o/p. |
| → $q_0$ | $q_0$ | 0 | $q_1$ | 1 |
| $q_1$ | $q_2$ | 2 | $q_0$ | 0 |
| $q_2$ | $q_1$ | 1 | $q_2$ | 2 |



In this machine the output depends upon the present state and present input

for the i/p string 1011. the transitions are,



That means.

Initial state is $q_0$.

$q_0$ by 1 it reaches $q_1$ and its output is 1.
$q_1$ by 0 it " $q_2$ " " " " 2
$q_2$ by 1 " " $q_2$ " " " " 2
$q_2$ by 1 " " $q_2$ " " " " 2

Thus the output string is 1222.

## (2) Mealy Machine

Mealy machine is a machine in which output symbol depends upon the present input symbol and present state of the machine.

- The mealy machine can be defined as.

a six tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where

$Q$ is a finite set of states

$\Sigma$ is finite set of input symbols.

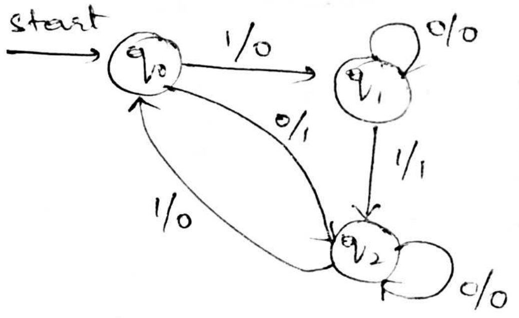
$\Delta$ is an output alphabet.

$\delta$ is state Transition function such that $Q \times \Sigma \rightarrow Q$

$\lambda$ is machine function such that $Q \times \Sigma \rightarrow \Delta$.

$q_0$ is initial state of machine.

For eg:-



- For the input string 1001 the output will be 0001.
- In mealy machine the length of input string is equal to length of output string.

**Problem 1 :** Design a Moore machine to generate 1's complement of given binary number.

Sol^n To generate 1's complement of given binary number the simple logic which we will apply is that

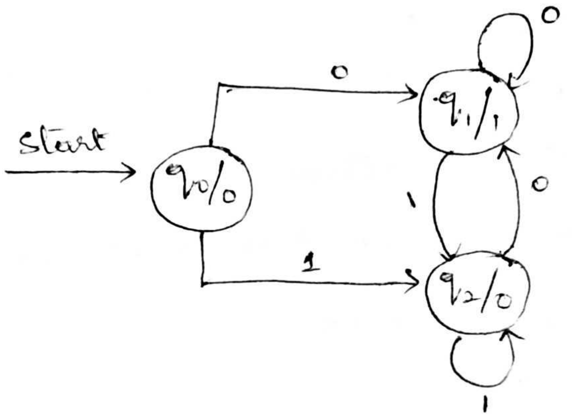
if i/p is 0 then o/p will be 1

if i/p is 1 then o/p will be 0.

That means there are 3 states,

1) one - start state
2) second state is for taking 0's as i/p and produces o/p as 1.
3) The third state is for taking 1's as i/p and producing o/p as 0.

Hence moore machine will be,



For instance, take one binary number 1011 Then

| | | | | | |
|---|---|---|---|---|---|
| Input : | | 1 | 0 | 1 | 1 |
| State : | $q_0$ | $q_2$ | $q_1$ | $q_2$ | $q_2$ |
| Output ; | 0. | 0 | 1. | 0 | 0 |

- Thus We get 00100 as 1's complement of 1011.
- We can neglect the initial 0. and the o/p which we get is 0100 which is 1's complement of 1011.

- The transition table can be drawn as below -

| | $\leftarrow \delta \rightarrow$ | | $\leftarrow \lambda \rightarrow$ |
|---|---|---|---|
| current state | Next state | | output |
| | 0 | 1 | |
| $\rightarrow q_0$ | $q_1$ | $q_2$ | 0 |
| $q_1$ | $q_1$ | $q_2$ | 1 |
| $q_2$ | $q_1$ | $q_1$ | 0. |

Thus the moore machine M = $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

where

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

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

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

The transition table shows the $\delta$ and $\lambda$ functions.

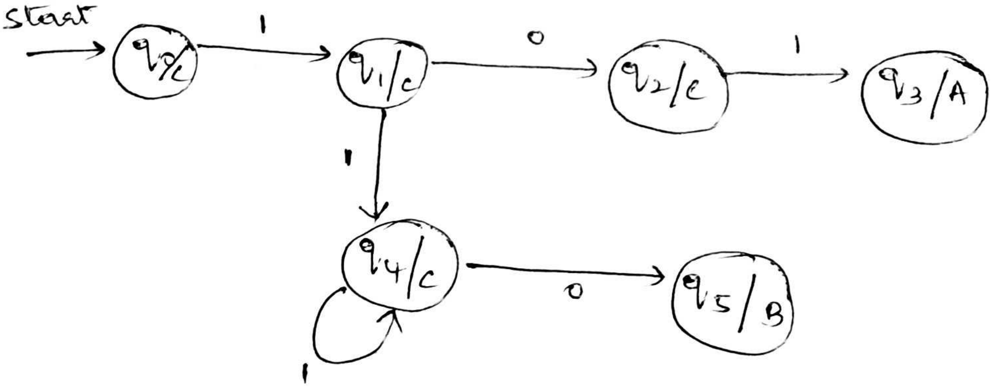
Problem 2 : Design a Moore and Mealy machine for a binary i/p sequence such that if it has a substring 101 the machine outputs A if input has substring 110 it outputs B otherwise it outputs c.

Soln :- for designing this kind of machine we need to take care of two conditions

(1) for checking 101

(2) for checking 110.

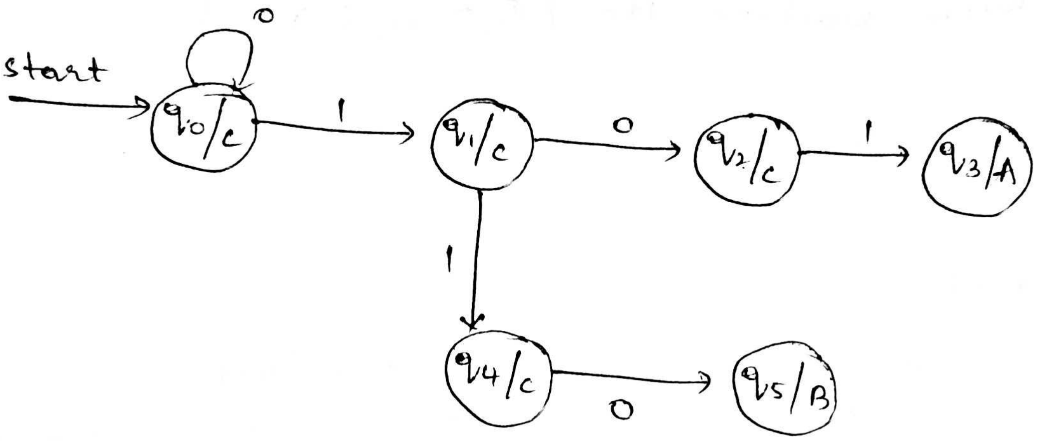
- If we get 101 the output will be A.
- If we get 110 the output will be B.
- For other strings the output will be c.



Now, we insert the possibilities of 1's and 0's for each state.

The the moore machine becomes.

(P.T.O)


start
0
q0/c
1
q1/c
0
q2/c
1
q3/A
1
q4/c
0
q5/B

Problem 3: Design a Mealy machine to find 2's complement of a given binary number.

Soln :- for designing 2's complement of a binary number we assume that input is read from LSB to MSB.

We will keep the binary number as it is until we read first 1. Keep that 1 as it is then change remaining 1's by 0's and 0's by 1's,

for eg:-

1011

←

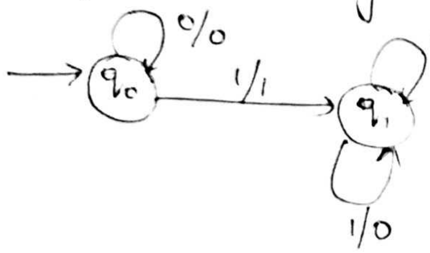
read from LSB

Keep the first 1 from LSB as it is and toggle the remaining bits we will get

0101

Thus 2's complement of 1011 is 0101.

The required Mealy machine will be -



- In the start state if we get either 0 we will be in the same state.
- If we get 1 then we will move to next state i.e q1 and o/p will be 1.

- After 1st 1 which ever the values comes either 0 or 1 they should be change by its complements.

for eg:- 101010

First find 1's complement

i.e 010101

Now add 1

$$\begin{array}{r} 010101 \\ +\ 1 \\ \hline 010110 \end{array}$$

By our logic 101010 (1st 1) (1st zero)

010110

←

Now the 2's complement of 101010 is

010110

Problem 4: Design a Moore machine which will increment the given binary number by 1.

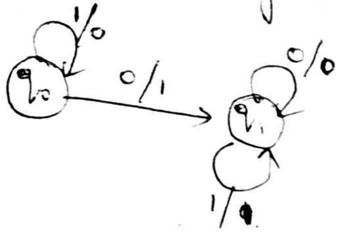
Sol^n We will read the binary number from LSB one bit at a time. We will replace each 1 by 0 until we get first 0. Once we get first 0 we will replace it by 1 and then keep remaining bits as it is.

for eg:-

```
              1 0 1 1
      as it is  ↓ ↓ ↓
        ←    1  1 0 0
             1
```

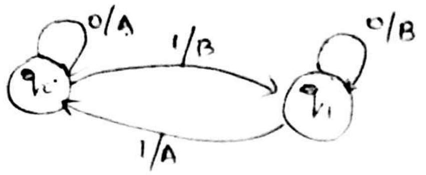
so the increment of 1 to the i/p string 1011 is 1100.

— Now whenever we get a zero, we move to the next state i.e $q_1$



After zero if we get 0 or 1 we will keep them as it is.

**Problem 1:** Convert the following Mealy machine into eqvt Moore machine.



**sol^n^** : The states for Moore machine will be $[q_0, A], [q_0, B]$, $[q_1, A], [q_1, B]$.

Then we will calculate $\delta'$ and $\lambda'$ as follows.

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

$$\lambda'([q_0, A]) = A$$

$$\delta'([q_1, A], 0) = [\delta(q_1, 0), \lambda(q_1, 0)]$$
$$= [q_1, B]$$

$$\lambda'([q_1, A]) = A$$

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

$$\lambda'([q_1, A]) = A$$

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

$$\lambda'([q_1, B]) = B.$$

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

$$\lambda'([q_1, B]) = B.$$

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

$$\lambda'([q_0, A]) = A$$

$$\delta'([q_0, B], 0) = [\delta(q_0, 0), \lambda(q_0, 0)] = [q_0, A]$$

$$\lambda'([q_0, B]) = A$$

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

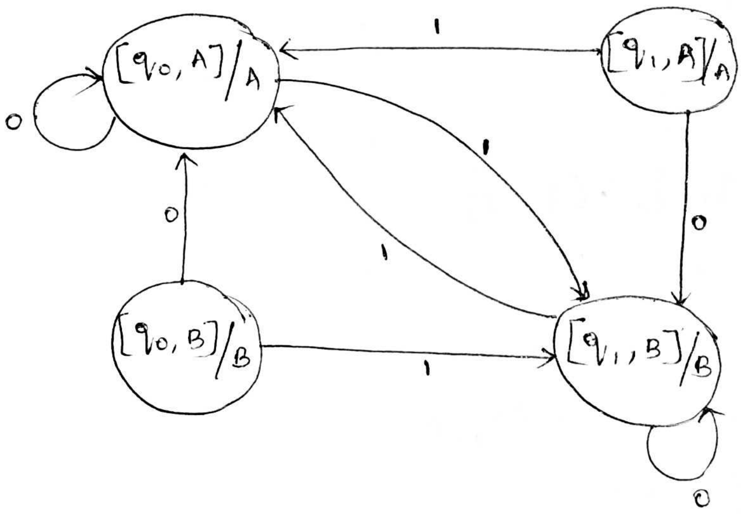
$= [q_1, B]$

$\lambda'([q_0, B]) = B.$

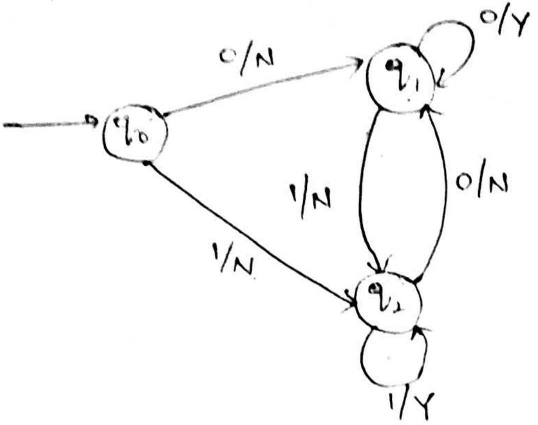
Hence the transition table will be.

| | 0 | 1 | output. |
|---|---|---|---|
| $[q_0, A]$ | $[q_0, A]$ | $[q_1, B]$ | A |
| $[q_0, B]$ | $[q_0, A]$ | $[q_1, B]$ | B. |
| $[q_1, A]$ | $[q_1, B]$ | $[q_0, A]$ | A |
| $[q_1, B]$ | $[q_1, B]$ | $[q_0, A]$ | B. |

The transition diagram will be,



Problem 2 : convert the following Mealy machine into equ Moore machine.



Solⁿ : We will write the transition table for given transition ~~table~~ diagm

|  | Input 0 | output | Input 1 | output |
|---|---|---|---|---|
| $q_0$ | $q_1$ | N | $q_2$ | N |
| $q_1$ | $q_1$ | Y | $q_2$ | N |
| $q_2$ | $q_1$ | N | $q_2$ | Y |

Now we will find out the states and corresponding outputs for moore machine.

The states for moore machine will be -

$[q_0, N], [q_0, Y], [q_1, N], [q_1, Y], [q_2, N], [q_2, Y]$

Now let us calculate $\delta'$ and $\lambda'$ for all the above given states.

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

$\delta'([q_0, N], 1) = [\delta(q_0, 1), \lambda(q_0, 1)] = [q_2, N]$ $\lambda'[q_0, N] = N$

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

$\delta'([q_0, Y], 1) = [\delta(q_0, 1), \lambda(q_0, 1)] = [q_2, N]$ $\lambda'[q_0, Y] = Y$

$\delta'([q_1, N], 0) = [\delta(q_1, 0), \lambda(q_1, 0)] = [q_1, Y]$ $\lambda'[q_1, N] = N$

$\delta'([q_1, N], 1) = [\delta(q_1, 1), \lambda(q_1, 1)] = [q_2, N]$ $\lambda'[q_0, N] = N.$

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

$\delta'([q_1, Y], 1) = [\delta(q_1, 1), \lambda(q_1, 1)] = [q_2, N]$ $\lambda'[q_0, Y] = Y.$

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

$\delta'([q_2, N], 1) = [\delta(q_2, 1), \lambda(q_2, 1)] = [q_2, Y]$

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

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

$\lambda'[q_2, N] = N$

$\lambda'[q_2, N] = N$

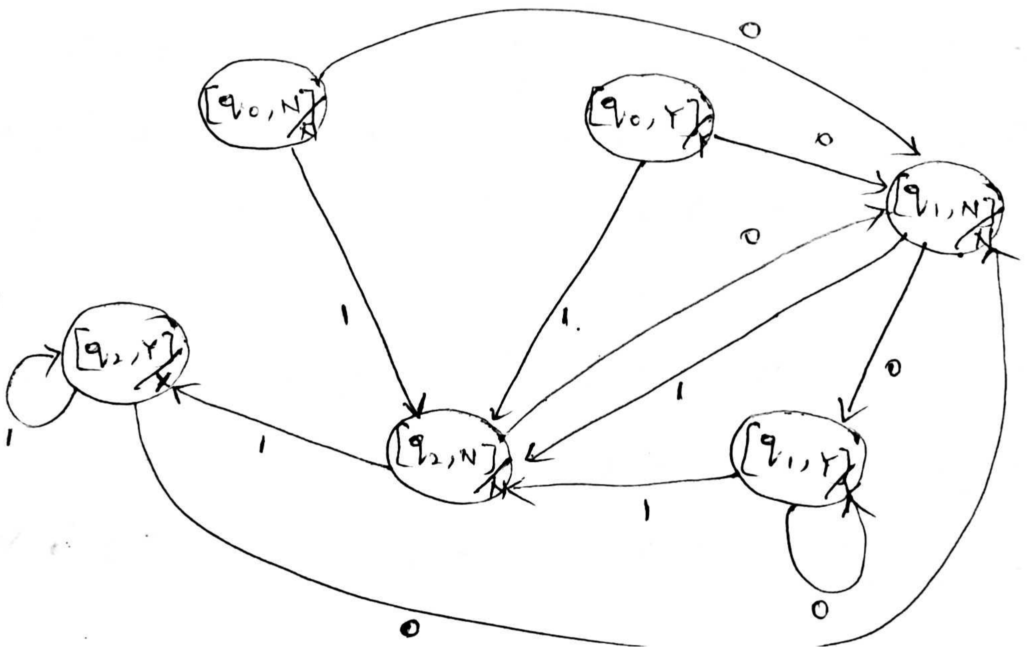
$\lambda'[q_2, Y] = Y$

$\lambda'[q_2, Y] = Y$

Now the transition table for Moore machine will be.

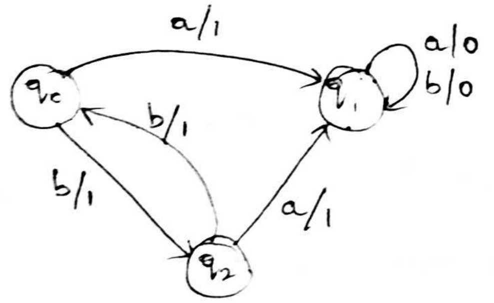
| | 0 | 1 | output |
|---|---|---|---|
| $[q_0, N]$ | $[q_1, N]$ | $[q_2, N]$ | N. |
| $[q_0, Y]$ | $[q_1, N]$ | $[q_2, N]$ | Y. |
| $[q_1, N]$ | $[q_1, Y]$ | $[q_2, N]$ | N. |
| $[q_1, Y]$ | $[q_1, Y]$ | $[q_2, N]$ | Y |
| $[q_2, N]$ | $[q_1, N]$ | $[q_2, Y]$ | N. |
| $[q_2, Y]$ | $[q_1, N]$ | $[q_2, Y]$ | Y |

Now Moore Machine will be.

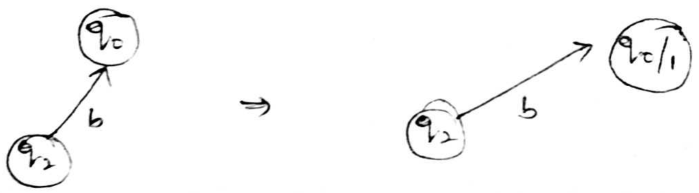


Problem 3 : construct the Moore machine for given Mealymachine

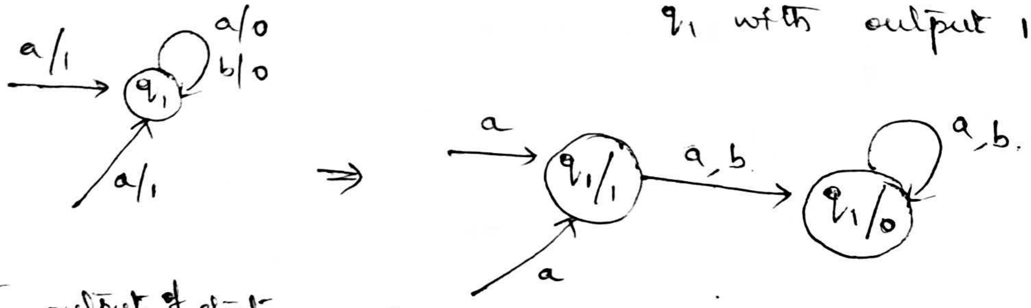
| state / I/p | a | b | o/p. |
|---|---|---|---|
| $q_0$ | $q_1$ | $q_2$ | 1 |
| $q_1$ | $q_1$ | $q_1$ | 0 |
| $q_2$ | $q_1$ | $q_0$ | 1. |



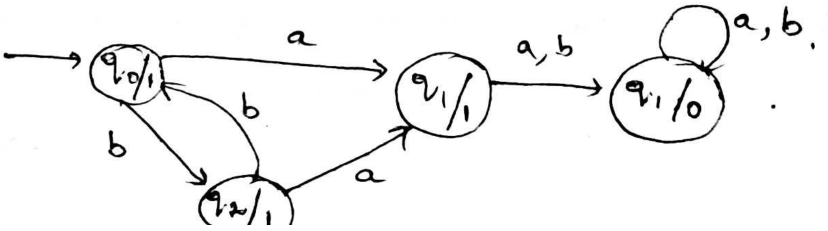
state $q_0$ : Only one incoming edge is carrying output 1. Hence output of $q_0$ state is 1.



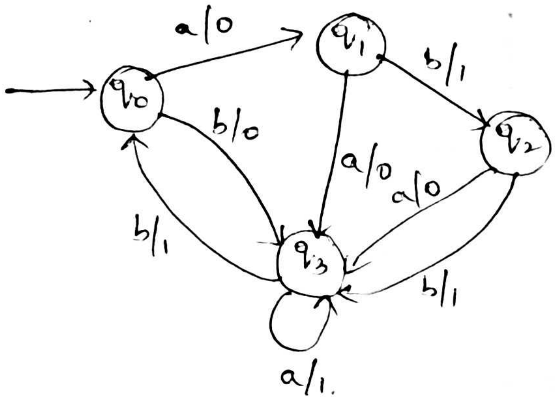
state $q_1$ : The incoming edge to state $q_1$ carries the output 1, from $q_0$ and $q_2$. And the self loop to $q_1$ state shows that output of state $q_1$ should be 0. Hence to adjust both the o/p symbols we will split state $q_1$ as $q_1$ with output 0 and $q_1$ with output 1



IIIrly output of state $q_2$ is 1.



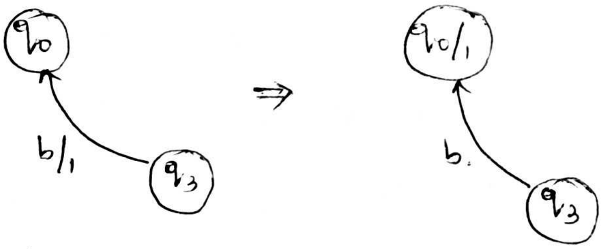
Problem 4: construct Moore Machine for Mealy machine.



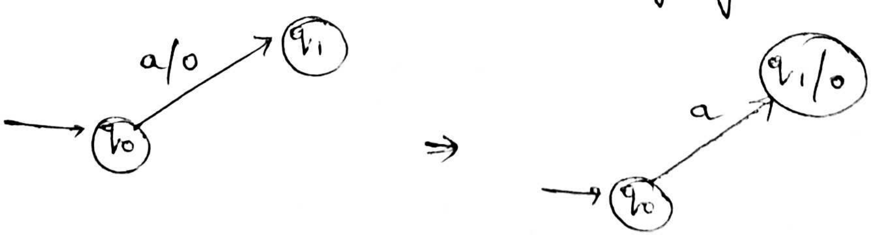
solⁿ. In Mealy machine output is given along the edge with input symbol. In moore machine each state has the o/p. while designing the moore machine we first observe the incoming edges of a state. The output carried by each incoming edge defines, the output of that corresponding state.

In the given Mealy machine

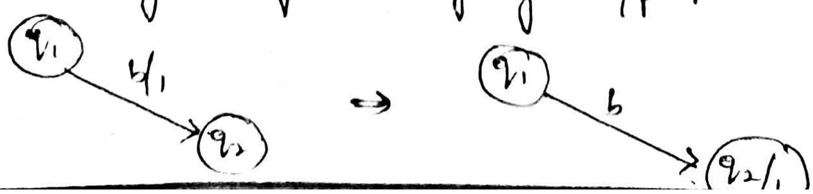
State q0 : The only one incoming edge carrying the o/p 1. Hence o/p of state q0 is 1.



State q1 : The only one incoming edge carrying the o/p 0.

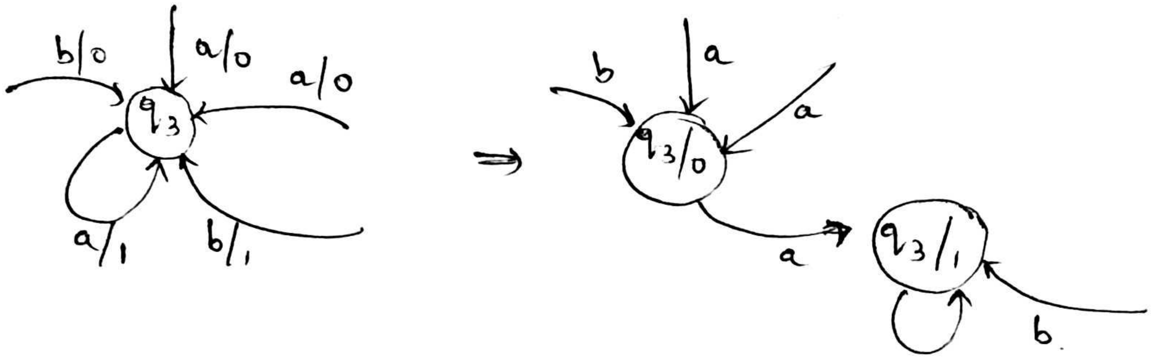


State q2 : The only incoming edge carrying o/p 1



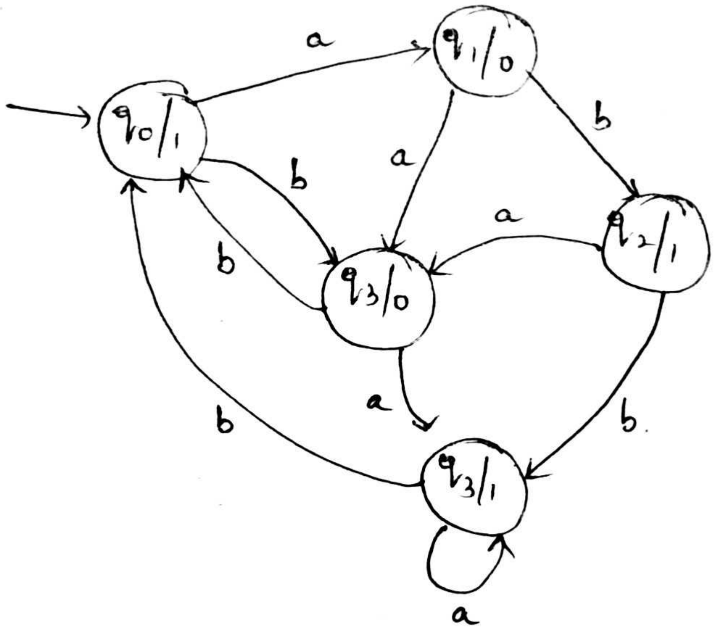
State $q_3$: Now for state $q_3$, there are 5 incoming edges. Three of which are carrying the output symbol 0 and remaining two are carrying output symbol 1.

We have to split up state $q_3$ as - state $q_3$ with output 0 and state $q_3$ as output 1.



We have to adjust the outgoing edges from $q_3$ accordingly

The converted Moore machine will be.



# EQUIVALENCE OF MOORE AND MEALY MACHINES

- The meaning of word equivalence is the two machines which accept the same language.
- Hence equivalence of Moore and Mealy machine means both the machines generate the same output string for same input string.
- We cannot convert moore machine to its equivalent Mealy machine directly because length of Moore machine is one longer than Mealy machine for the given input.

## Conversion of Moore Machine to Mealy Machine :

Let $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ be a Moore machine. The equivalent Mealy machine can be represented by

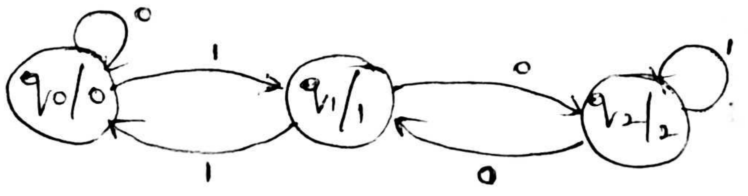
$M' = (Q, \Sigma, \Delta, \delta, \lambda', q_0)$. The output function $\lambda'$ can be obtained as -

$$\lambda'(q, a) = \lambda(\delta(q, a))$$

Ex (1): The Moore machine to determine residue mod 3 for binary number is given below. Convert it into Mealy equivalent machine.

| $Q \backslash \Sigma$ | 0 | 1 | output ($\lambda$) |
|---|---|---|---|
| $q_0$ | $q_0$ | $q_1$ | 0 |
| $q_1$ | $q_2$ | $q_0$ | 1 |
| $q_2$ | $q_1$ | $q_2$ | 2 |

Soln: The transition diagram for the given problem can be drawn as -



The output function λ' can be obtained using following rule,

$$\lambda'(q,a) = \lambda(\delta(q,a))$$

Hence we will obtain output for every transition corresponding to input symbol.

$$\lambda'(q_0,0) = \lambda(\delta(q_0,0))$$
$$= \lambda(q_0)$$ i.e output of $q_0$

$$\boxed{\lambda'(q_0,0) = 0}$$

$$\lambda'(q_0,1) = \lambda(\delta(q_0,1))$$
$$= \lambda(q_1)$$ i.e o/p of $q_1$

$$\boxed{\lambda'(q_0,1) = 1}$$

$$\lambda'(q_1,0) = \lambda(\delta(q_1,0))$$
$$= \lambda(q_2)$$

$$\boxed{\lambda'(q_1,0) = 2}$$

$$\lambda'(q_1,1) = \lambda(\delta(q_1,1))$$
$$= \lambda(q_0)$$

$$\boxed{\lambda'(q_1,1) = 0}$$

$$\lambda'(q_2,0) = \lambda(\delta(q_2,0))$$
$$= \lambda(q_1)$$
$$= 1$$

$$\boxed{\lambda'(q_2,0) = 1}$$

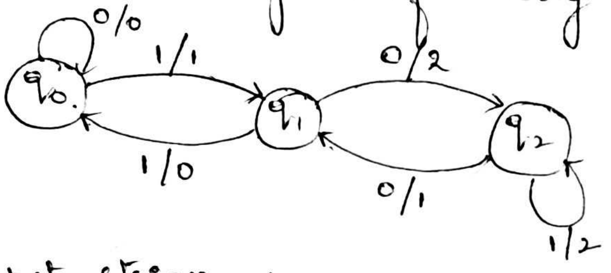
$$\lambda'(q_2,1) = \lambda(\delta(q_2,1))$$
$$= \lambda(q_2)$$

$$\boxed{\lambda'(q_2,1) = 2}$$

Hence the transition table can be drawn as follows:-

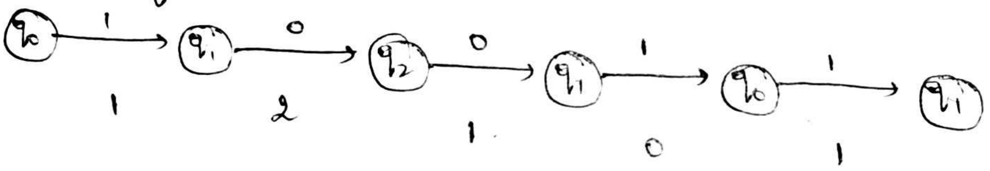
| Q\Σ | Input 0 | | Input 1 | |
|---|---|---|---|---|
| | state | o/p. | state | o/p. |
| $q_0$ | $q_0$ | 0 | $q_1$ | 1 |
| $q_1$ | $q_2$ | 2 | $q_0$ | 0 |
| $q_2$ | $q_1$ | 1 | $q_2$ | 2 |

The transition diagram of mealy machine is,

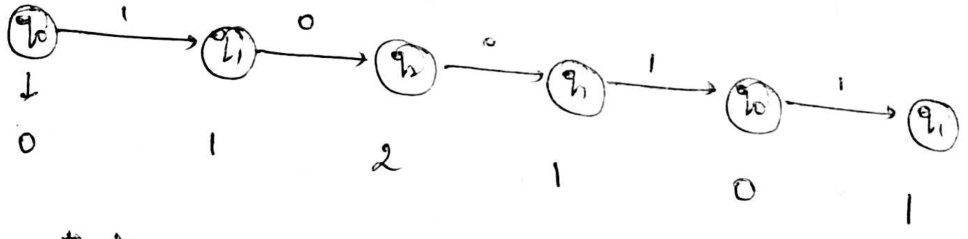


The input string 10011

The output for Mealy machine will be



The out-put sequence for Moore machine will be.



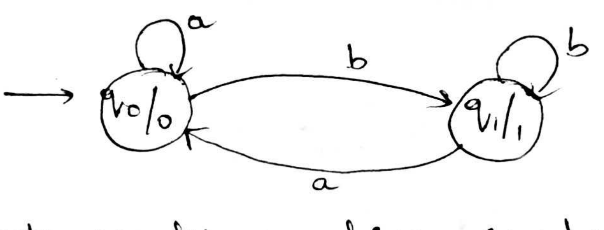
The output sequence in mealy machine is $n$

The output sequence in Moore machine is $n+1$.

Ex: 2: Convert the following Moore machine into equt Mealy machine $M = (\{q_0, q_1\}, \{a, b\}, \{0, 1\}, \delta, \lambda, q_0)$

| δ | a | b | output (λ) |
|---|---|---|---|
| $q_0$ | $q_0$ | $q_1$ | 0 |
| $q_1$ | $q_0$ | $q_1$ | 1 |

Soln :-



The eqvt mealy machine can be obtained as follows :-

$$\lambda'(q_0,a) = \lambda(\delta(q_0,a))$$
$$= \lambda(q_0)$$
$$\lambda'(q_0,a) = 0$$

$$\lambda'(q_0,b) = \lambda(\delta(q_0,b))$$
$$= \lambda(q_1)$$
$$\lambda'(q_0,b) = 1$$

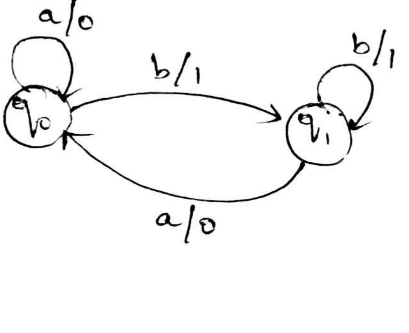
$$\lambda'(q_1,a) = \lambda(\delta(q_1,a))$$
$$= \lambda(q_0)$$
$$\lambda'(q_1,a) = 0$$

$$\lambda'(q_1,b) = \lambda(\delta(q_1,b))$$
$$= \lambda(q_1)$$
$$\lambda'(q_1,b) = 1$$

Hence the transition table can be drawn as follows :-

| Q \ Σ | Input a | | Input b | |
|---|---|---|---|---|
| | state | o/p | state | o/p |
| $q_0$ | $q_0$ | 0 | $q_1$ | 1 |
| $q_1$ | $q_0$ | 0 | $q_1$ | 1 |

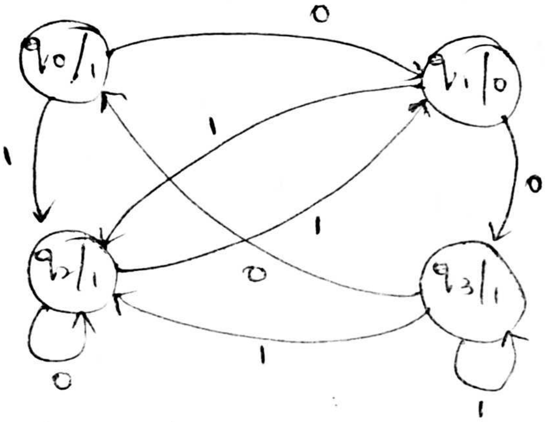
The Mealy machine will be,



Ex:3: Construct a Mealy machine equivalent to the Moore machine given by the following table.

| | a=0 | a=1 | output |
|---|---|---|---|
| $q_0$ | $q_1$ | $q_2$ | 1 |
| $q_1$ | $q_3$ | $q_2$ | 0 |
| $q_2$ | $q_2$ | $q_1$ | 1 |
| $q_3$ | $q_0$ | $q_3$ | 1 |

Sol$^n$ We first draw the transition diagram from given transition diagram.



$\lambda'(q_0, 0) = \lambda(\delta(q_0, 0)) = \lambda(q_1) = 0$

$\lambda'(q_0, 1) = \lambda(\delta(q_0, 1)) = \lambda(q_2) = 1$

$\lambda'(q_1, 0) = \lambda(\delta(q_1, 0)) = \lambda(q_3) = 1$

$\lambda'(q_1, 1) = \lambda(\delta(q_1, 1)) = \lambda(q_2) = 1$

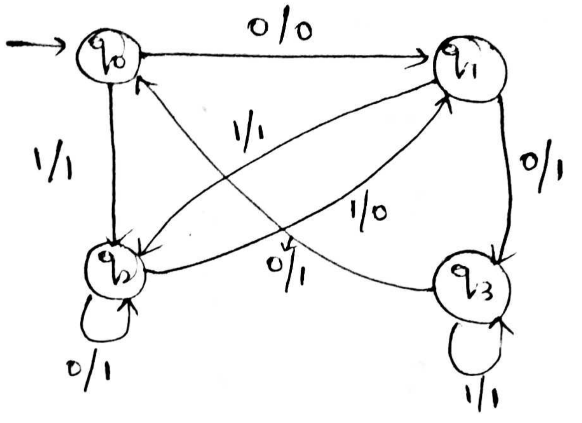
$\lambda'(q_2, 0) = \lambda(\delta(q_2, 0)) = \lambda(q_2) = 1$

$\lambda'(q_2, 1) = \lambda(q_1) = 0$

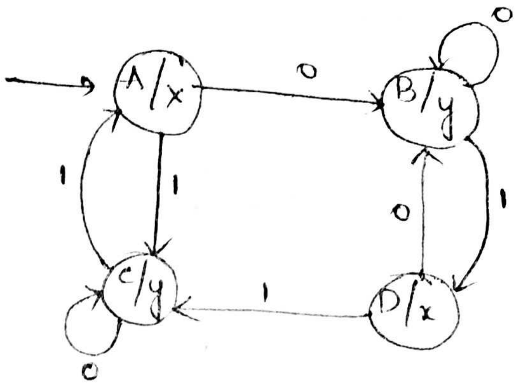
$\lambda'(q_3, 0) = \lambda(q_0) = 1$

$\lambda'(q_3, 1) = \lambda(q_3) = 1$

Now the mealy machine will be



Ex : 4 :- Reduce the Moore machine



Sol<sup>n</sup> We will design a transition table for given transition diagram.

| state \ input | 0 | 1 | output ($\lambda$) |
|---|---|---|---|
| → A | B | C | x |
| B | B | D | y |
| C | C | A | y |
| D | B | C | x. |

The o/p function $\lambda'$ can be obtained using following rule,

$$\lambda'(q,a) = \lambda(\delta(q,a))$$

$\lambda'(A,0) = \lambda(\delta(A,0)) = \lambda(B) = y.$

$\lambda'(A,1) = \lambda(\delta(A,1)) = \lambda(C) = y$

$\lambda'(B,0) = \lambda(\delta(B,0)) = \lambda(B) = y$

$\lambda'(B,1) = \lambda(\delta(B,1)) = \lambda(D) = x$

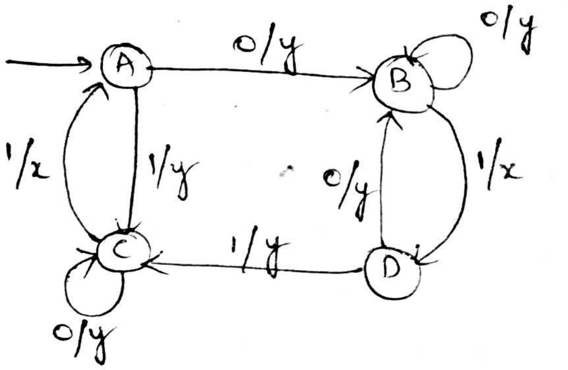
$\lambda'(C,0) = \lambda(\delta(C,0)) = \lambda(C) = y$

$\lambda'(C,1) = \lambda(\delta(C,1)) = \lambda(A) = x$

$\lambda'(D,0) = \lambda(\delta(D,0)) = \lambda(B) = y$

$\lambda'(D,1) = \lambda(\delta(D,1)) = \lambda(C) = y$

Now the transition diagram of mealy machine is



## TO CONVERT A MEALY MACHINE INTO A MOORE MACHINE

- Consider the transition table for Mealy machine is given in fig:

| Present state | Next state |  |  |  |
|---|---|---|---|---|
|  | Input a=0 |  | Input a=1 |  |
|  | state | o/p | state | o/p. |
| → $q_0$ | $q_0$ | 0 | $q_1$ | 1 |
| $q_1$ | $q_2$ | 2 | $q_0$ | 0 |
| $q_2$ | $q_1$ | 1 | $q_2$ | 2 |

(1) At first stage, develop the procedure, both machines accepts exactly the same set of input sequences.

) Look into the next state column for any state say $q_i$

and determine the no of diff+ output states, say $q_j$ and determine the no. of diff+ outputs.

3) split $q_i$ into several diff+ states, the no. of such states being equal to the no. of diff+ outputs associated with $q_i$.

In the above transition table,

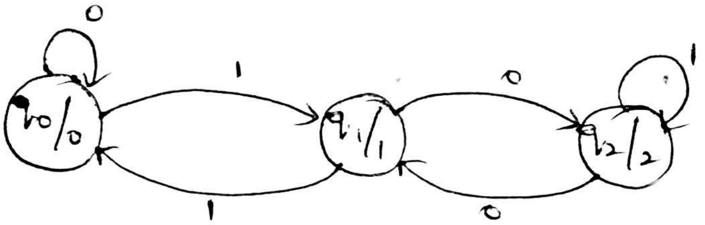
$q_0$ is associated with one output 0.
$q_1$ is " " " " 1
$q_2$ " " " " " 2

| Present state | Next state a=0 | Next state a=1 | output |
|---|---|---|---|
| → $q_0$ | $q_0$ | $q_1$ | 0 |
| $q_1$ | $q_2$ | $q_0$ | 1 |
| $q_2$ | $q_1$ | $q_2$ | 2 |

Revised state table.

The above fig gives moore machine, initial state $q_0$ is associated with output 0.

- Moore machine accepts zero-length sequence which is not accepted by mealy machine.



Moore machine.

# AUTOMATA THEORY AND FORMAL LANGUAGES

## UNIT-2

(Lecture Notes)

III CSE
R-18
JNTUH

By
G M Subhani, Asst.Professor
CSE, MRIET.

# UNIT - II
# REGULAR LANGUAGES

## Introduction:

- The language accepted by finite automata are easily described by simple expressions called regular expressions.
- The Regular expression is the most effective way to represent any language.
- The language accepted by some regular expression is known as a regular language.

## REGULAR SET:

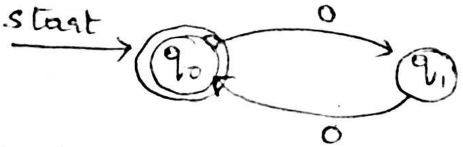
Regular sets are the sets which are accepted by finite automata

for example : $L = \{ \varepsilon, 00, 0000, 000000, \ldots \}$

i,e the set of even number of zeros.

We can represent this set by a finite automata as shown in fig:

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



The given set L is said to be a regular set because it is represented by finite automata.

## REGULAR EXPRESSIONS:

Let Σ be an alphabet which is used to denote the input set.

The regular expression over Σ can be defined as follows:-

1) ϕ is a regular expression which denotes the empty set.
2) ε is a regular expression and denotes the set {ε} and it is a null string.

3). For each 'a' in $\Sigma$, a is a regular expression and denotes the set $\{a\}$.

4). If r and s are regular expressions denoting the language $L_1$ and $L_2$ reespectively, then

$r+s$ is equivalent to $L_1 \cup L_2$ i,e Union.

$rs$ is equivalent to $L_1 L_2$ i,e concatenation

$r^*$ is equivalent to $L_1^*$ i,e closure.

- The $r^*$ is known as Kleen closure or closure which indicates occurence of r for $\infty$ no. of times.

- For example if $\Sigma = \{a\}$ and we have regular expression $R = a^*$, then R is a set denoted by

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

i,e R includes any no. of a's as well as empty string which includes zero number of a's appearing - denoted by $\varepsilon$ character.

- IIIrdly there is a positive closure of L which can be shown as $L^+$.

- The $L^+$ denotes set of all the strings except the $\varepsilon$ or null string.

- The null string can be denoted by $\varepsilon$ or $\wedge$

If $\Sigma = \{a\}$ and if we have regular expression $R = a^+$ then R is a set denoted by

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

We can construct $L^*$ as.

$$\boxed{L^* = \varepsilon . L^+}.$$

**Problem 1:** write the regular expression for the language accepting all combinations of a's over the set $\Sigma = \{a\}$.

**solⁿ:** All combinations of a's means

a may be single, double, triple and so on.

- There may be the case that a is appearing for zero times, which means a null string.
- We expect the set of $\{\varepsilon, a, aa, aaa \ldots\}$

So we can give regular expression for this as.

$R = a^*$ i.e Kleen closure of a.

**Problem 2:** Design the regular expression (r.e) for the language accepting all combinations of a's except the null string over $\Sigma = \{a\}$.

**solⁿ:** The regular expression has to be built for the language.

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

This set indicates that there is no null string.

So we can denote r.e as

$R = a^+$

**Problem 3:** Design a R.E for the language containing all the strings containing any number of a's and b's.

**solⁿ:** The regular expression will be,

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

This will give us the set as

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

(4) construct the R.E for the language containing all strings having any number of a's and b's except the null string.

Solⁿ :- r.e = $(a+b)^+$

(5) construct the r.e for the language accepting all strings which are ending with 00 over the set $\Sigma = \{0, 1\}$

Solⁿ :- The r.e. has to be formed in which at the end, there should be 00. That means.

r.e = (any combination of 0's and 1's) 00

r.e = $(0+1)^* 00$

Thus the valid strings are 100, 0100, 1000, 10100

(6) write r.e. for the language accepting the strings which are starting with 1 and ending with 0, over the set $\Sigma = \{0, 1\}$.

Solⁿ: The first symbol in r.e should be 1 and the last symbol should be 0.

So, $R = 1(0+1)^* 0$

(7) If L = { The language starting and ending with a and having any combination of b's in b/w ; Then what is r ?

Solⁿ :- The R.E

$r = a\, b^*\, a$

**Ex: 8** Describe in simple English the language represented by the following R.E

$r = (a + ab)^*$

**solⁿ :** We will first try to find out the set of strings, which can be possible by this r,

$L(r) = \{a, aba, abab, aab, aaa \dots\}$

The language is begining with a but not having consecutive b's.

**Ex: 9** Write r.e. to denote the language L over $\Sigma^*$, where $\Sigma = \{a,b,c\}$ in which every string will be such that any number of a's is followed by any no. of b's is followed by no. of c's.

**solⁿ:** Any no. of a's means $a^*$

Any no. of b's means $b^*$

Any no. of c's means $c^*$

$r = a^* b^* c^*$

**Ex: 10** write a R.E to denote the language L over $\Sigma^*$, where $\Sigma = \{a,b,c\}$ such that every string will have atleast one a followed by atleast one b followed by atleast one c.

**solⁿ:** Here we are using atleast, means null string is not allowed at all.

So we can write

$R = a^+ b^+ c^+$

**Ex: 11** Write a r.e to denote a language L which accepts all the strings which begin or end with either 00 or 11.

**Sol$^n$** R.E can be categorized into two subparts.

$R = L_1 + L_2$

$L_1$ = strings which begin with 00 or 11

$L_2$ = strings which ~~begin~~ end with 00 or 11.

Let us find out $L_1$ and $L_2$

$L_1 = (00+11)$(any no. of 0's and 1's)

$L_1 = (00+11)(0+1)^*$

$\text{III}^{rly}$

$L_2 =$ (any no. of 0's and 1's) $(00+11)$

$L_2 = (0+1)^*(00+11)$

Hence

$$R = [(00+11)(0+1)^*] + [(0+1)^*(00+11)]$$

**Ex: 12** Write a r.e to denote a language over $\Sigma^*$, where $\Sigma = \{a, b\}$ such that the 3rd character from right end of the string is always a.

**Sol$^n$** The r.e contains the third symbol from right end as a

r =

| any no. of characters of a's and b's | a | either a or b | either a or b |
|---|---|---|---|
|  | 3rd | 2nd | 1st |

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

The valid strings are babb, baaa, baba, aaab and so on.

Ex:13 construct r.e for the language L which accepts all strings with atleast two b's over $\Sigma = \{a,b\}$

Soln $R = (a+b)^* b (a+b)^* b (a+b)^*$.

Ex:14: construct r.e for the language which consists of exactly two b's over the set $\Sigma = \{a,b\}$

Soln $R = a^* b a^* b a^*$.

Ex:15 write r.e which contains L having strings which should have atleast one 0 and atleast one 1.

Soln The required expression will be

$R = [(0+1)^* 0 (0+1)^* 1 (0+1)^*] + [(0+1)^* 1 (0+1)^* 0 (0+1)^*]$

Ex:16 construct r.e which denotes a language L over the set $\Sigma = \{0\}$ having even length of string.

Soln $L = \{\varepsilon, 00, 0000, 000000 ------\}$.

We can give r.e as $R = (00)^*$.

Ex:17: write r.e which denotes a language L over the set $\Sigma = \{1\}$ having odd length of strings.

Soln :- $R = 1(11)^*$.

Ex:18 construct r.e for the language L over the set $\Sigma = \{a,b\}$ in which the total number of a's are divisible by 3.

Soln $L = \{baaa, bababa, ababab -----\}$.

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

Ex : 19 write the r.e to denote the language L over $\Sigma = \{a, b\}$ such that all the strings do not contain the sub string "ab"

Sol$^n$

$L = \{\epsilon, a, b, bb, aa, ba, baa \dots\}$

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

Ex : 20 find a r.e corresponding to each of the following subsets over $\{0,1\}^*$

a) The 10$^{th}$ symbol from the right end is 1.

$r.e = (0+1)^* 1 (0+1)(0+1)(0+1)(0+1)(0+1)(0+1)(0+1)(0+1)(0+1)$
)

b). no. of occurences are divisible by 3. $\Sigma = \{0,1\}$.

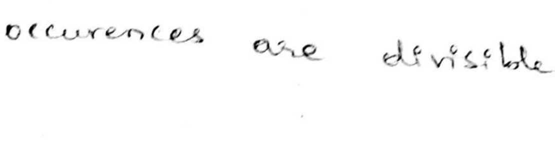
i) If number of 0's are divisible by 3.

$r.e = (1^* 0 1^* 0 1^* 0 1^*)^*$

ii) If number of 1's are divisible by 3.

$r.e = (0^* 1 0^* 1 0^* 1 0^*)^*$

Ex : 21 Conclude what type of strings will be accepted by the below f.A



Sol$^n$

The $r.e = (0+1)^* 00 (0+1)^*$.

(22): Define regular expression and find regular expression for the following:

$L = \{ w /$ every odd position of $w$ is $1 \}$ defined over $\Sigma = \{0, 1\}$.

Regular expression for given language L :

$$\underset{\text{odd}}{\boxed{1}} \quad 0 \quad \underset{\text{odd.}}{\boxed{1}} \quad 0 \quad - - - - - - - - - -$$

Hence $r.e = (10)^*$

(23) Describe in English language, the sets represented by the following regular expressions.

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

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

soln (a) This is a language containing all the strings that are beginning with a and ending with ab.

(b). This is a language in which either any no of a's are followed by single b or any no, of b's are followed by single a.

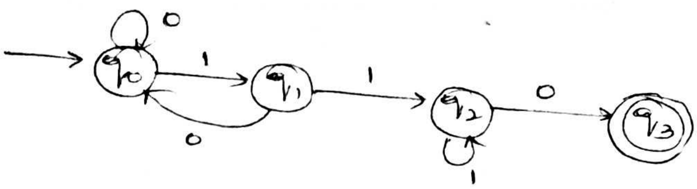
(24). Write down regular expression corresponding to each of the following language over $\{0, 1\}$.

(i) The language of all strings in which every 0 is followed immediately by 11.

(ii) The language of all strings that do not contain substring 110.

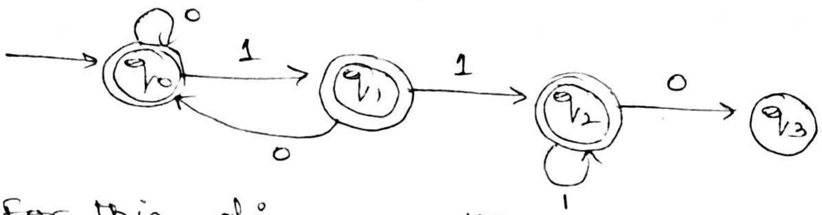
soln (i). The possible strings are $\{011, 1, 11, 111, 0111, 01111 \ldots\}$

Hence $r.e = (1 + 011)^*$.



(ii) Above is the transition diagram for the language that contains 110.

Now for the language that do not contain 110, we will change final states to non-final states and non-final states to final states.



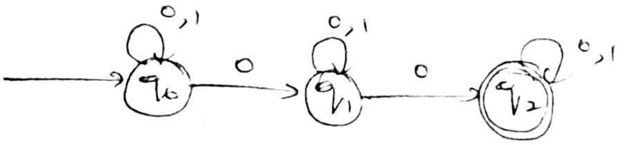
For this diagram, the r.e will be

$r.e = [0^*(10)^* + 1^*]$

(25) Find a regular expression corresponding to each of following subset of [0,1]

(i) The language of all strings containing atleast two 0's.

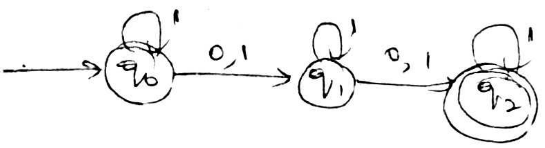
Soln



$r.e = (0+1)^* 0 (0+1)^* 0 (0+1)^*$

(ii) The language of all strings containing at the most two 0's.

Soln



$r.e = 1^* (0+1) 1^* (0+1) 1^*$.

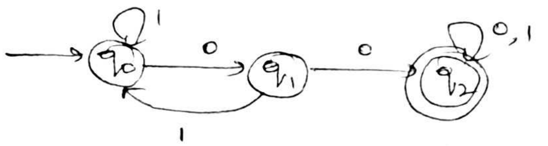
(iii) The language of all strings in which both the no of o's and 1's are odd.

solⁿ :- r.e = $1(11)^* + 0(00)^*$

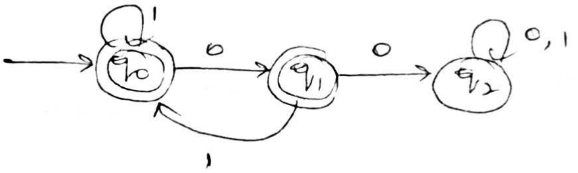
(26) write regular expressions for the languages.

(i) $\Sigma = \{0,1\}$ containing all possible combinations of o's and 1's but not having two consecutive o's.

Solⁿ :- FA for the language which contains consecutive o's.



New FA for the language which does not contain consecutive o's is



from This FA the required r.e would be

r.e = $[1^*(01)^*]^*$

(ii) The set of all strings of o's and 1's such that every pair of adjacent o's appears before any pair of adjacent 1's.

solⁿ r.e = $[(00)^*(11)^*]^*$

(iii) The set of all strings over $\Sigma = \{0,1\}$ without length two.

solⁿ r.e = $(0+1)(0+1)(0+1)^+$.

(27) Define regl expn. Find the r.e for the following languages on $\{a, b\}$

Soln (i) Language of all strings w such that w contains exactly one 1 and even no of o's.

$$r.e = [(00)^* 1 + 1(00)^*]$$

(ii) Set of strings over $\{0, 1, 2\}$ containing atleast one o and atleast one 1.

Soln

$$r.e = (0 + 1 + 2^*)^+$$

(28) Find the regular expression for the language $\{L = a^{2n} b^{2m} \mid n \geq 0, m \geq 0\}$

Soln $r.e = (aa)^*(bb)^*$

(29) Give regular Expn for representing the set L of strings in which every o is immediately followed by atleast two 1's.

$$r.e = (011 + 1)^*$$

(30) obtain the r.e to accept strings of a's, b's and c's such that fourth symbol from the right end is a.

## IDENTITY RULES :

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

- for showing this equivalence of regular expressions we need to show some identities of regular expressions.

- Let P, Q and R are regular expressions then the Identity rules are as given below.

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

2) $\varepsilon^* = \varepsilon$   $\varepsilon$ is null string

3) $(\phi)^* = \varepsilon$   $\phi$ is empty string.

4) $\phi R = R\phi = \phi$

5) $\phi + R = R$

6) $R + R = R$

7) $RR^* = R^*R = R^+$

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

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

10) $(P+Q)R = PR + QR$

11) $(P+Q)^* = (P^*Q^*) = (P^* + Q^*)^*$

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

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

14) $\varepsilon + R^* = R^*$

15) $(PQ)^*P = P(QP)^*$

16) $R^*R + R = R^*R$.

Let us see one important theorem named Arden's Theorem which helps in checking the equivalence of two regular expressions.

ARDEN'S THEOREM: Let P and Q be the two regular expressions over the input set $\Sigma$. The Regular expression R is given as

$$R = Q + RP$$

which has a unique solution as

$$R = QP^*.$$

Proof: Let P and Q are two regular expressions over The input string $\Sigma$.

- If P does not contain $\varepsilon$ Then, there exists R such that

$$R = Q + RP \text{ ——— } (1)$$

We will replace R by $QP^*$ in Equation (1)

consider R.H.S of Equation (1)

$$= Q + QP^*P$$

$$= Q(\varepsilon + P^*P) \qquad \because \varepsilon + R^*R = R^*$$

$$= QP^*$$

Thus $\boxed{R = QP^*}$ is proved.

To prove that $R = QP^*$ is a unique solution, We will now replace L.H.S of equation (1) by Q+RP. Then it becomes

$$LHS = Q + RP$$

But again R can be replaced by Q+RP.

$\therefore \quad Q+RP = Q+(Q+RP)P$

$= Q+QP+RP^2$

Again replace R by Q+RP

$= Q+QP+(Q+RP)P^2$

$= Q+QP+QP^2+RP^3$

Thus if we ~~go~~ go on replacing R by Q+RP then we get,

$Q+RP = Q+QP+QP^2+ \cdots\cdots QP^i + RP^{i+1}$

$= Q(\varepsilon + P+P^2+ \cdots\cdots P^i) + RP^{i+1}$

from Equation ①,

$R = Q(\varepsilon + P+P^2+ \cdots\cdots P^i) + RP^{i+1}$ —②

where $i \geq 0$

consider Equation 2,

$R = Q(\underbrace{\varepsilon + P+P^2+ \cdots +P^i}_{P^*}) + RP^{i+1}$

$\therefore \quad R = QP^* + RP^{i+1}$

& Let W be a string of length i.

In $RP^{i+1}$ has no string of less than i+1 length.

Hence W is not in set $RP^{i+1}$.

Hence R and $QP^*$ represent the same set.

Hence it is proved that

$R = Q+RP$ has a unique solution

$R = QP^*$.

# PROBLEMS FOR EQUIVALENCE OF R.E

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

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

Sol

$\text{sol}^n$ :- Let us solve L.H.S first.

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

We will take $(1+00^*1)$ as a common factor

$1+00^*1$

$1+00^*1(\varepsilon+(0+\underset{R}{\underset{\downarrow}{10^*1}})^*(0+\underset{R}{\underset{\downarrow}{10^*1}}))$

As we know, $\varepsilon+R^*R = \varepsilon+RR^* = R^*$

$\therefore \underbrace{(1+00^*1)}_{\downarrow}((0+10^*1)^*)$

Take 1 as a common factor

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

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

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

$=$ R.H.S

$\therefore$ The two regular expressions are equivalent.

Problem 2 : show that $(0^*1^*)^* = (0+1)^*$

$\text{sol}^n$ : consider L.H.S.

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

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

$= \{$ any combination of 0's,
any combination of 1's,
any combination of 0 and 1, $\varepsilon\}$

lllrly

R.H.S

$= (0+1)^*$

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

$= \{\varepsilon$, any combination of 0's and any combination of 1's, any combination of 0 and 1$\}$

Hence L.H.S = R.H.S is proved.

Problem 3: show that $(r+s)^* \neq r^* + s^*$

soln: Let us consider the L.H.S first

L.H.S $= (r+s)^*$

$= \{\varepsilon, r, rr, s, ss, rs, sr, rsrs, \ldots\}$

$= \{\varepsilon$, any combination of r and s$\}$

R.H.S $= \{\varepsilon, r, rr, rrr, s, ss, sss \ldots\}$

$= \{\varepsilon$, any combination of only r or any combination of only s$\}$

Note: In R.H.S there is no combination of r and s together.

Hence L.H.S ≠ R.H.S is proved.

Problem 4: prove $\varepsilon + 1^*(011)^*(1^*(011)^*)^* = (1+011)^*$

soln: Let, L.H.S is

$$\varepsilon + \underbrace{1^*(011)^*}\ \underbrace{(1^*(011)^*)^*}$$

If we consider $1^*(011)^*$ as P then

$\Rightarrow \varepsilon + PP^*$ $\quad (\because \varepsilon + PP^* = \varepsilon + P^*P = P^*)$

$\Rightarrow P^*$

We can put $P = 1^*(011)^*$ then,

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

Now consider

$P_1 = 1$ ; $P_2 = 011$ then it becomes

$\Rightarrow (P_1^* P_2^*)^*$

$\Rightarrow (P_1 + P_2)^*$ $\because (P^*Q^*)^* = (P+Q)^*$

$\Rightarrow (1 + 011)^*$

$=$ R.H.S

Thus L.H.S = R.H.S Hence

$\varepsilon + 1^*(011)^*(1^*(011)^*)^* = (1+011)^*$ is proved.

Constructing Finite Automata for Given R.Es.

- There is a close relationship between a finite automata and the regular expression We can show this relation in Fig below

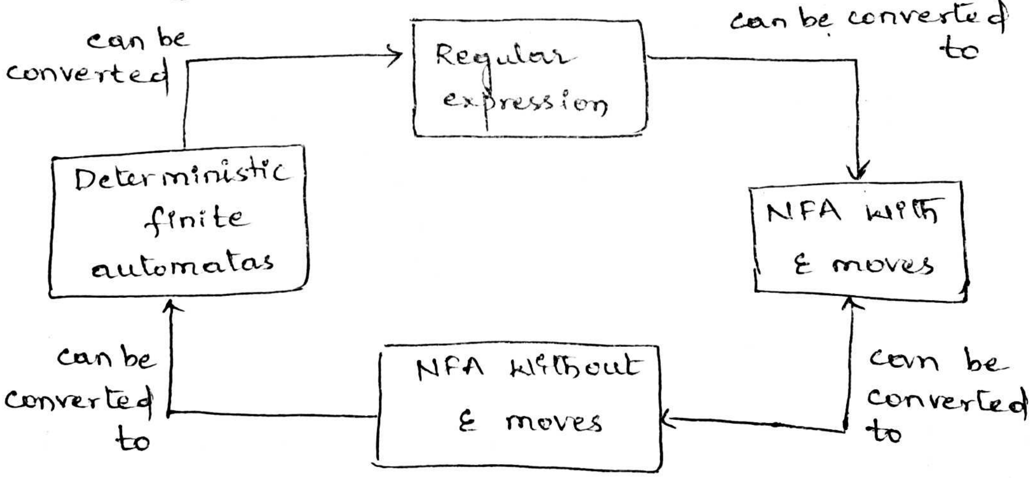


Fig: Relnship b/w F.A and R.E

The figure shows that it is convenient to convert the regular expression to NFA with ε moves.

# APPLICATIONS OF REGULAR EXPRESSIONS:

## (1) Regular Expressions in UNIX:

Before seeing the applications, we shall introduce the UNIX notation for extended regular expressions. This notation gives us a number of additional capabilities. In fact, the UNIX extensions include certain features, especially the ability to name and refer to previous strings that have matched a pattern, that actually allow non regular languages to be recognized.

- The first enhancement to the regular-expression notation concerns The fact that most real applications deal with the ASCII character set. our examples have typically used a small alphabet, such as {0, 1}. The existence of only two symbols allowed us to write succint expressions like 0+1 for "any character". If there were 128 characters, the same expression would involve listing them all, and would be highly inconvenient to write. Thus UNIX regular expressions allow us to write character classes to represent large sets of characters as succinctly as possible.

The rules for character classes are:

- The symbol . (dot) stands for "any character".
- The sequence $[a_1 a_2 \text{-----} a_k]$ stands for the regular exp^n.

$$a_1 + a_2 + \text{-----} + a_k.$$

This notation saves about half the characters, since we don't have to write the +- signs. For example, we could express the four characters used in c comparison operators by [<>=!].

- Between the square braces we can put a range of the form x-y to mean all the characters from x to y in the ASCII sequence. since the digits have codes in order as do the uppercase letters and the lowercase letters, we can express many of the classes of characters that we really care about with just a few keystrokes. For Eg:- the digits can be expressed [0-9], the upper case letters can be expressed [A-Z] and the set of all letters and digits can be expressed [A-Za-z0-9].

- square brackets, or other characters that have special meanings in UNIX regular expressions can be represented as characters by preceding them with a backslash (\).

- There are special notations for several of the most common classes of characters. for instance:

a) [:digit:] is the set of ten digits, the same as [0-9].

b) [:alpha:] stands for any alphabetic character, as does [A-Za-z]

c) [:alnum:] stands for the digits and letters (alphabetic and numeric characters), as does [A-Za-z0-9].

In addition there are several operators that are used in UNIX regular expressions that we have not encountered previously.

1. The operator | is used in place of + to denote union.
2. The operator ? means "zero or one of". Thus R? in UNIX is the same as $\epsilon + R$
3. The operator + means "one or more of". Thus R+ in UNIX is shorthand for RR* in our notation.
4. The operator {n} means "n copies of". Thus, R{5} in UNIX is shorthand for RRRRR.

(2) Lexical Analysis:

One of the oldest applications of regular expressions was in specifying the component of a compiler called a "lexical Analyzer". This component scans the source program and recognizes all tokens, Keywords and identifiers are common examples of tokens, but there are many others.

- The UNIX command lex and its GNU version flex, accept as input a list of regular expressions, in the UNIX style, each followed by a bracketed section of code that indicates what the lexical analyzer is to do when it finds an instance of that token.

such a facility is called a lexical-analyzer generator,

- commands such as lex and flex have been found extremely useful because the regular expression notation is exactly as powerful as we need to describe tokens.

Example: Given below is an example of partial input to the lex command, describing some of the tokens that are found in the language C.

(i) else {return (ELSE);}

This line handles the keyword else and the action is to return a symbolic constant (ELSE in this Example) to the parser for further processing.

(ii) [A-Za-z][A-Za-z0-9]* {code to enter the found identifier in the symbol table;
return (ID);
}

This line contains a regular expression describing identifiers: a letter followed by zero or more letters and/or digits. The action is first to enter that identifier in the symbol table.

(iii) >= {return (GE);}

This entry is for the sign >=, a two character operator.

The last example we show is for the sign =, a one character operator

(iv) = {return(EQ);}

There would be expressions describing each of the keywords each of the signs and punctuation symbols like commas and parenthesis, and families of constants such as numbers and strings.

(3) Finding Patterns in Text :

We introduced the notion that automata could be used to search efficiently for a set of words in a large repository such as the web. While the tools and technology for doing so are not so well developed as that for lexical analyzers, the regular expression notation is valuable for describing searches for interesting patterns.

- The general problem for which regular expression technology has been found useful is the description of a vaguely defined class of patterns in text.

- By using patterns regular expression notation, it becomes easy to describe the patterns at a high level with little effort and to modify the description quickly when things go wrong. A "compiler" for regular expressions is useful to turn the expressions we write into executable code.

Example: suppose we want to find the Restaurants within 10 minutes.

We shall focus on recognizing street addresses in -

The last example we show is for the sign =, a
one character operator

(iv) = {return(EQ);}

There would be expressions describing each of the keywords each of the signs and punctuation symbols like commas and parenthesis, and families of constants such as numbers and strings.

(3) Finding Patterns in Text :

We introduced the notion that automata could be used to search efficiently for a set of words in a large repository such as the web. While the tools and technology for doing so are not so well developed as that for lexical analyzers, the regular expression notation is valuable for describing searches for interesting patterns.

-The general problem for which regular expression technology has been found useful is the description of a vaguely defined class of patterns in text.

-By using patterns regular expression notation, it becomes easy to describe the patterns at a high level with little effort and to modify the description quickly when things go wrong. A "compiler" for regular expressions is useful to turn the expressions we write into executable code.

Example: suppose we want to find the Restaurants within 10 minutes.

We shall focus on recognizing street addresses in -

particular.

To begin, a street address will probably end in "street" or its abbreviation, "st."

However some people live on "Avenues" or "Roads", and these might me abbreviated in the address as well.

Thus, we might use as the ending for our regular expression something like:

```
Street | st\. | Avenue | Ave\. | Road | Rd\.
```

In the above expression, we have used UNIX-style notation, with the vertical bar '|' rather than +, as a Union operator.

- Here the dots are escaped with a preceding backslash. Since in this case we really want the dot to end the three abbreviations.

- Some streets have a name consisting of more than one word such as Rhode Island Avenue.

  Thus the notation can be

```
[A-Za-z]*([A-Z][a-z]*)*
```

→ we may include house no in the address.

eg:- "123A Main st"

```
[0-9]+[A-Z]? [A-Z][a-z]*([A-Z][a-z]*)*
```

+ means one or more

? means zero or one.

# CONVERSION OF F.A TO R.E.

Conversion steps :

1) Let $q_1$ be the initial state.

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

3) Let $\alpha_{ji}$ represents the transition from $q_j$ to $q_i$

4) Calculate $q_i$ such that

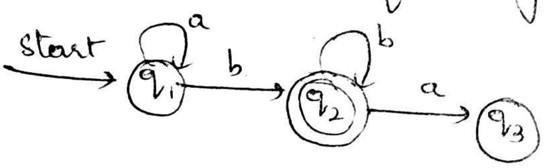
$$q_i = \alpha_{ji} \cdot q_j$$

If $q_i$ is start state

$$q_i = \alpha_{ji} q_j + \varepsilon.$$

5) IIIrly compute final state which ultimately gives the r.e r.

Ex :1 construct r.e from given DFA.



Let us write the Eqns.

$q_1 = q_1 a + \varepsilon.$ ($\because q_1$ is start state, we add $\varepsilon$)

IIIrly $q_2 = q_1 b + q_2 b.$ —— ②

Simplify $q_1$ first

$q_1 = \varepsilon + q_1 a$ —— ①

$R = Q + RP.$

$R = QP^*$

$q_1 = \varepsilon . a^*$

$q_1 = a^*$ ($\because \varepsilon R = R$)

Substitute $q_1 = a^*$ in Eq$^n$ (2)

$q_2 = q_1 b + q_2 b$

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

$R = Q + R P$

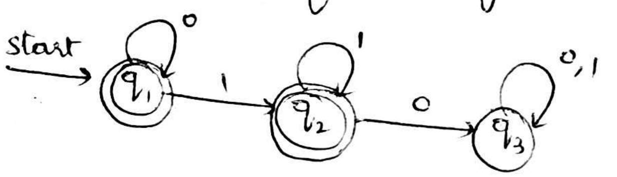
$R = Q P^*$

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

$q_2 = a^* b^+ \quad (\because b.b^* = b^+)$

$\therefore \boxed{r.e = a^* b^+}$

(2) construct r.e for the given DFA



sol$^n$ Let us build r.e for each state.

$q_1 = q_1 0 + \varepsilon$

$q_2 = q_1 1 + q_2 1$

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

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

Since $q_1$ and $q_2$ are final states, we are interested in solving $q_1$ and $q_2$ only

$q_1 = \varepsilon + q_1 0$

$R = Q + RP.$

$R = Q . P^*$

$q_1 = \varepsilon . 0^*$

$q_1 = 0^* \quad (\because \varepsilon R = R)$

Substituting $q_1$ in $q_2$

$q_2 = q_1 1 + q_2 1$

$q_2 = 0^* 1 + q_2 1$

$R = Q + RP.$

$R = QP^*$

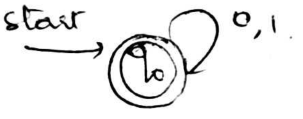
$q_2 = 0^* 1 . 1^*$

$q_2 = 0^* . 1^+ \quad (\because 1.1^* = 1^+)$

The r.e is given by $r = q_1 + q_2$

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

③ Represent the lang accepted by following DFA



soln

$q_0 = q_0 0 + q_0 1 + \varepsilon$

$q_0 = q_0 (0+1) + \varepsilon$

$R = RP + Q$

$R = QP^*$

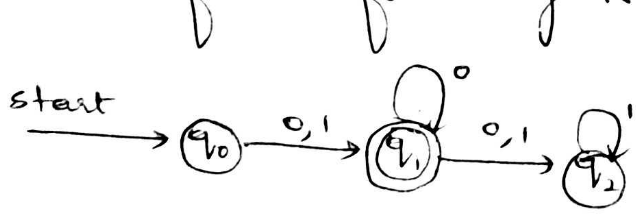
$q_0 = \varepsilon . (0+1)^* \quad (\because \varepsilon R = R)$

$q_0$ is final state.

$r = (0+1)^*.$

④ Find r.e for the following NFA's

(a)



$q_0 = \varepsilon \longrightarrow ①$

$q_1 = q_0 0 + q_0 1 + q_1 0 \longrightarrow ②$

$q_2 = q_1 0 + q_1 1 + q_2 1. \longrightarrow ③$

Substitute Eqn ① in Eqn ②

$q_1 = \varepsilon.0 + \varepsilon.1 + q_1 0$

$q_1 = \varepsilon(0+1) + q_1 0$

$q_1 = (0+1) + q_1 0$ $\quad (\because \varepsilon R = R)$

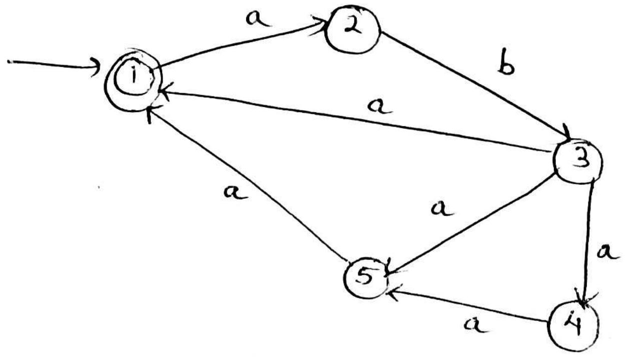
$R = Q + RP$

$R = QP^*$

$q_1 = (0+1).0^*$

$\boxed{r.e = (0+1).0^*}$

(b)



Let us write $Eq^{ns}$.

$1 = \varepsilon + 3a + 5a$ —— ①

$2 = 1a$ —— ②

$3 = 2b$ —— ③

$4 = 3a$ —— ④

$5 = 3a + 4a$ —— ⑤

state ① is final state, Hence $Eq^n$ for state 1 is r.e

Let

$3 = 2b.$

$4 = 3a'$

$4 = 2b.a$ $\quad (\because 3 = 2b)$

$4 = 1aba$ $\quad (\because 2 = 1a)$ —— ⑥

$5 = 4a + 3a$ $\quad (\because 4 = 3a)$

$5 = 3aa + 3a$

$5 = 3a(a + \varepsilon)$

$5 = 1aba(a+\varepsilon)$ —— ⑦

$1 = \varepsilon + 3a + 5a$

$1 = \varepsilon + 1aba + 1aba(a+\varepsilon).a$

$1 = 1aba + 1abaaa + \varepsilon.1abaa + \varepsilon$

$1 = 1aba + 1abaaa + \underline{1abaa} + \varepsilon$

$1 = \underbrace{1aba[\varepsilon + aa + a]} + \varepsilon$

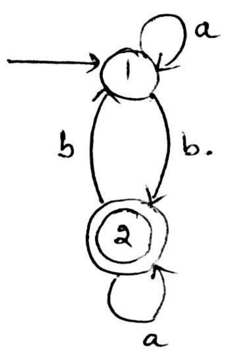
$R = R.P + Q$

$R = QP^*$

$1 = \varepsilon.(aba(\varepsilon+aa+a))^*$

$\boxed{r.e = (aba(\varepsilon+aa+a))^*}$ $(\because \varepsilon R = R)$

(5)



Soln

$1 = 1a + 2b + \varepsilon$ —— (1)

$2 = 1b + 2a$ —— (2)

state 2 is a final state , Hence Eqn of state 2 will give r.e.

$1 = 1a + \underbrace{2b + \varepsilon}$

$R = QP^*$

$1 = (2b+\varepsilon).a^*$ —— (3)

By substituting Eqn 3 in (2)

$2 = (2b+\varepsilon).a^*b + 2a$

$2 = 2ba^*b + \varepsilon.a^*b + 2a$

$2 = \underline{2ba^*b} + a^*b + \underline{2a}$ $(\because \varepsilon R = R)$

$2 = 2(a + ba^*b) + a^*b.$

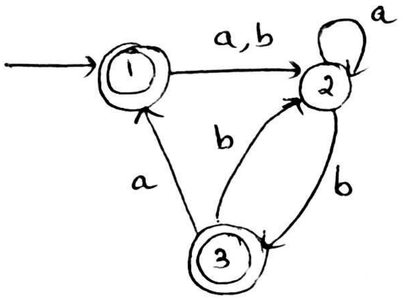
$R = R.P + Q.$

$R = Qp^*$

$2 = a^*b(ba^*b+a)^*$

$$\boxed{r.e = a^*b(ba^*b+a)^*}$$

6



Soln

$1 = 3a + \varepsilon$ —— (1)

$2 = 3b + 2a + 1a + 1b$ —— (2)

$3 = 2b$ —— (3)

Simplify Eqn (2)

$2 = 1(a+b) + 2a + 2b.b \quad (\because 3 = 2b)$

$$\underbrace{2}_{R} = \underbrace{1(a+b)}_{Q} + \underbrace{2}_{R}\underbrace{(a+bb)}_{P}$$

$2 = 1(a+b)(a+bb)^*$

Sub the above Eqn in Eqn (1)

$1 = 3a + \varepsilon.$

$1 = 2ba + \varepsilon$

$$\underbrace{1}_{R} = \underbrace{1}_{R}\underbrace{(a+b)(a+bb)^*.ba}_{P} + \underbrace{\varepsilon}_{Q}$$

$R = Qp^*$

$1 = \varepsilon.[(a+b)(a+bb)^*.ba]^*$

$1 = [(a+b)(a+bb)^*.ba]^* \quad (\because \varepsilon R = R)$

Sub above Eqn in Eqn (3)

$3 = 2b.$

$3 = [1(a+b)(a+bb)^*].b.$

$3 = [(a+b)(a+bb)^*.ba]^*.(a+b)(a+bb)^*.b.$

r.e = Eq$^n$ of state 1 + Eq$^n$ of state 3.

r.e $= [(a+b)(a+bb)^*.ba]^* + [(a+b)(a+bb)^*.ba]^*(a+b)(a+bb)^*.b.$

# ALGEBRAIC LAWS FOR REGULAR EXPRESSIONS

(1) Associativity and commutativity:

commutativity is the property of an operator that says we can switch the order of its operands and get the same result.

An eg: for arithmetic was given below:

$x+y=y+x$. where $x$ and $y$ are variables that can be replaced by any two numbers.

Associativity is the property of an operator that allows us to regroup the operands when the operator is applied twice.

For eg: the associative law of multiplication is $(x\times y)$

$(x\times y)\times z = x\times(y\times z)$

Here are three laws of these types that hold for regular expressions.

- $L+M = M+L$. This law, the commutative law for union, says that we may take the union of two languages in either order.

- $(L+M)+N = L+(M+N)$. This law, the associative la for union, says that we may take the Union o three languages either by taking the union of fir two initially or taking the union of the last two initially.

Note : Together with the commutative law for Union, we conclude that we can take the union of any collection of languages with any order and grouping and the result will be the same.

- $(LM)N = L(MN)$. This law, the associative law for concatenation, says that we can concatenate three languages by concatenating either the first two or the last two initially.

- $LM = ML$ — says that concatenation is commutative. This law is false in case of regular expressions.

Eg:- Consider the regular expressions 01 and 10. These expressions denote the languages $L_1$ for 01 and $L_2$ for 10.

Since the languages are different the general law $LM = ML$ cannot hold.

$LM = 0110$ ; $ML = 1001$.

(2) Identities and Annihilators :

An identity for an operator is a value such that when the operator is applied to the identity and some other value, the result is the other value.

for instance 0 is the identity for addition, since

$0 + x = x + 0 = x$.

And 1 is the identity for multiplication, since

$1 \times x = x \times 1 = x$

Annihilator : It is a value such that when the operator is applied to the annihilator and some other value, the result is the annihilator.

For instance 0 is an annihilator for multiplication, since result is the annihilator.

$0 \times x = x \times 0 = 0$

- There is no annihilator for addition.

There are three laws for regular expressions involving these concepts.

- $\phi + L = L + \phi = L$. This law says that $\phi$ is the identity for union.
- $\epsilon L = L\epsilon = L$. This law says that $\epsilon$ is the identity for concatenation.
- $\phi L = L\phi = \phi$. This laws says that $\phi$ is the annihilator for concatenation.

These laws are powerful tools in simplifications.

(3) Distributive Laws :

A distributive law involves two operators and asserts that one operator can be pushed down to be applied to each argument of the other operator individually.

- The most common example for distributive law is distributive law of multiplication over addition.

that is $x \times (y+z) = x \times y + x \times z$.

These laws we can state in two forms:

- $L(M+N) = LM + LN$. This law is the left distributive law of contatenation over union
- $(M+N)L = ML + NL$. This law is the right distributive law of concatenation over union.

## (4) The Idempotent Law:

An operator is said to be idempotent if the result of applying it to two of the same values as argments is that value. The common arithmetic operators are not idempotent; $x+x \neq x$ in general $x \times x \neq x$

- Union and intersection are common examples of idempotent operators.

Thus for regular expressions, we may assert the following law:

- $L+L=L$. This law, the idempotent law for union states that if we take the union of two identical expressions, we can replace them by one copy of the expression.

## (5) Laws Involving closures:

There are a number of laws involving the closure operators and its UNIX-style variants + and ?

(1) $(L^*)^* = L^*$.

(2) $\phi^* = \epsilon$

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

(4) $L^+ = LL^* = L^*L$

(5) $L^* = L^+ + \epsilon$

(6) $L? = \epsilon + L$ (? is for zero or ~~more~~ one).

## PROPERTIES OF REGULAR LANGUAGES:

### PUMPING LEMMA FOR REGULAR LANGUAGES:

This is a basic and important theorem used for checking whether given string is accepted by regular expression or not. In short, this lemma tells us whether given language is regular or not.

One key theme is that any language for which it is possible to design the finite automata is definitely the regular language.

Theorem: Let L be a regular set. Then there is a constant n such that if z is any word in L and $|z| \geq n$ We can write $z = uvw$ such that $|uv| \leq n$, $|v| \geq 1$ for all $i \geq 0$,

$uv^iw$ is in L.

Then n should not be greater than the number of states.

Proof: If the language L is regular it is accepted by a DFA. $M = (Q, \Sigma, \delta, q_0, F)$.

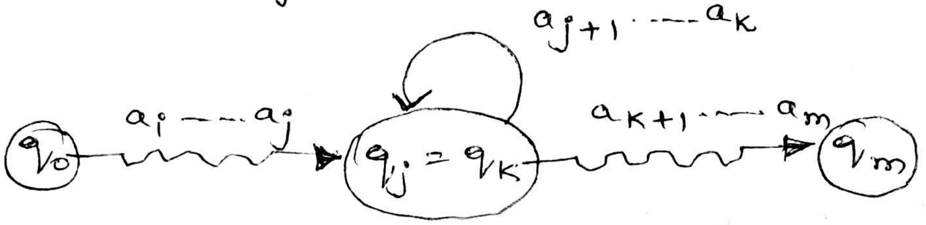
With some particular number of states say, n.

Consider the input can be $a_1, a_2, a_3 \text{------} a_m$, $m \geq n$.

The mapping function $\delta$ could be written as.

$\delta(q_0, q_1, q_2, q_3 \text{-----} q_i) = q_i$.

The transition diagram is shown below:



pumping lemma.

If $q_m$ is in F i,e $q_1, q_2, q_3 \text{-----} q_m$ is in L(M) then $a_1, a_2, a_3 \text{-----} a_j\ a_{K+1}\ a_{K+2} \text{-----} a_m$ is also in L(M).

- since there is a path from $q_0$ to $q_m$ that goes through $q_j$ but not around the loop labelled $a_{j+1} \text{----} a_K$.

Thus.

$$\delta(q_0, a_1, a_j\ a_{K+1} \text{----} a_m) = \delta(\delta(q_0, q_1, \text{--} q_j), a_{K+1} \text{--} a_m)$$
$$= \delta(q_j, q_{K+1} \text{----} q_m)$$
$$= \delta(q_K, q_{K+1} \text{----} q_m)$$
$$= q_m$$

That is what we have proved i,e given any long string can be accepted by F.A, We should be able to find a substring near the beginning of the string that may be pumped i,e repeated as many times as we like and resulting string may be accepted by F.A.

Pumping lemma is used to check whether given language

## Applications of Pumping Lemma:

Ex ①: Show that the set $L = \{ b^{i^2} \mid i \geq 1 \}$ is not regular.

Soln We have to prove that the language $L = b^{i^2}$ is not regular. This language is such that the no of b's is always a perfect square.

For example; if we take $i = 1$

$L = b^{1^2} = b$ the length $= 1$

$L = b^{2^2} = bbbb$ ; the length $= 4$. and so on.

Now, Let us consider

$L = b^{n^2}$ where length $= n^2$

it is denoted by $z$.

$|z| = n^2$

By pumping lemma, $z = uvw$

where $1 \leq v \leq n$

As $z = uv^iw$ where $i = 1$

Now we will pump $v$ i,e make $i = 2$

As we made $i = 2$ we have added one $n^2$.

$1 \leq v \leq n$

$n^2 + 1 \leq uvw \leq n + n^2$

i,e $n^2 + 1 + 2n \leq uvw \leq n^2 + 2n + n$.

$(n+1)^2 \leq uvw \leq n^2$

The string lies b/w two consecutive perfect squares.

But the string is not a perfect square.

Hence we can say the given language is not regular

for eg:-

$L = b^{i^2}$

$i = 2$

$L = bbbb$

$L = uvw$

Assume $uvw = b\underline{bb}b$

Take $u = b$
$v = bb$
$w = b.$

By pumping lemma, even if we pump v i.e increase v then language should show the length as perfect square.

$uvw$

$= uvvw$

$= bbbbbb$

= the length of b is not a perfect square.

Thus, the behavior of the language is not regular.

Ex-(2): Prove $L = \{ a^p$ is a prime $\}$ is not regular.

Sol^n Let us assume L is a regular and p is a prime number.

$L = a^p$

$|z| = uvw \quad i=1$

Now consider $L = uv^iw$ where $i=2$

$= uv.vw$

Adding 1 to p we get,

$p < |uvvw|$

$p < p+1$

But $p+1$ is not a prime number.

Hence what we have assumed becomes contradictory. Thus L behaves as it is not a regular language.

Ex: ③ show that $L = \{0^n 1^{n+1} \mid n > 0\}$ is not regular.

Soln: Let us assume that L is a regular language.

$|z| = |uvw|$

$= 0^n 1^{n+1}$

Length of string $|z| = n+n+1 = 2n+1$

That means length is always odd.

By pumping lemma

$= |uv.vw|$

That is if we add $2n+1$

$2n+1 < (2n+1) + 2n+1$

$2n+1 < 4n+2$

But if $n=1$ then we obtain $4n+2 = 6$ Which is no way odd.

Hence the language becomes irregular.

Even if We add 1 to the length of $|z|$, then

$|z| = 2n+1+1 = 2n+2$

$=$ even length of the string.

So this is not a regular language.

Ex: ④ Is $L = \{a^{2n} \mid n >= 1\}$ regular?

Soln We assume that $L = \{a^{2n}\}$ is regular language.

The string $w \in L$ and

$w = a^{2n}$

By pumping lemma if $w = xy^iz \in L$

We will map $w = a^{2n}$ with $xy^iz$

$$w = \underset{x}{\underset{\downarrow}{(aa)}}\ \underset{y}{\underset{\downarrow}{a}}\ \underset{z}{\underset{\downarrow}{a}}$$

By pumping lemma $w = xy^iz$.

Consider following cases

(i) $i=0$ then

$w = xy^iz$

$= xz$ $\qquad \because y^0 = 1$

$= aaa$

$w = a^3 \notin L$

(ii) $i = 2$ then

$w = xy^iz$

$= xyyz$

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

$= a^5 \notin L$

From the above two cases, It is clear that our assumption of L being regular is wrong.

Hence given language is not regular.

Ex: ⑤ Is $L = \{a^{2n} \mid n >= 1\}$ regular ?

Soln We assume that $L = \{a^{2n}\}$ is regular language.

The string $w \in L$ and

$w = a^{2n}$

By pumping lemma if $w = xy^iz \in L$

We will map $w = a^{2n}$ with $xy^iz$.

$w = \underbrace{(aa)}_{x}\underbrace{a}_{y}\underbrace{a}_{z}$

By pumping lemma $w = xy^iz$ consider following cases.

1) $i = 0$ then

$w = xy^iz$

$w = xy^0z$ $\qquad (\because y^0 = 1)$

$w = xz$

Ex (5): Find whether the following languages are regular or not.

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

Sol^n Consider $w = abab.$

$w^R = baba$

Hence

$L = ww^R.$

The string $z$ can be denoted by

$z = uvw$ where $|z| \geq n.$ and $|uv| \leq n.$

We assume $z = \underbrace{ab}_{n}\underbrace{ab}_{n}\underbrace{ba}_{n}\underbrace{ba}_{n}$

$|u| = n-1$

$|v| = 1$

i.e $|uv| = |u| + |v|$

$= n - 1 + 1$

$= n$

let $z = \underbrace{a}_{u}\ \underbrace{b}_{v}\ \underbrace{abbaba}_{w}$

According to pumping lemma,

When $z = uv^iw \in L$ then language is said to be regular.

We assume that $L = ww^R$ is regular.

We will find $uv^iw$

$\therefore$ $z = uv^iw$

$z = ababbaba$

Assume $i = 0$ i.e there will be

$z = uv^0w$

$z = uw$

then the string becomes

$z = aabbaba$

$z \neq ww^R.$

i.e $z = aabbaba \notin L$ —— (1)

Now consider $i=2$ then

$z = uv^2w$

$= abbabbaba$

$z \neq ww^R$

i.e $z = abbabbababa \notin L$ ——— ②

from equations ① & ② we can state that

$z = uv^iw \notin L$

Hence our assumption that $L = ww^R$ being regular is wrong.

Hence we can prove that

$z = ww^R$ is not regular.

(ii) consider $L = \{0^n 1^m 2^{n+m}, n, m \geq 1\}$

Assume that L is regular.

Sol^n $z = uvw$

$z = 0^n 1^m 2^{n+m}$ $t = n+m$

$z = 0^n 1^m 2^t$

According to pumping lemma, when for any language $z = uv^iw \in L$ then the lang is said to be regular.

Let, $z = uv^iw$ ; if $i=0$ then

$z = uv^0w$

$z = uw$ • For $n=1$;

$z = 00222$ where $t \neq n+m$ $z = 0^n \underline{1}^m 2^t$

Hence $z \notin L$ $z = 0^1 2^2$

$z = 02222$

Let $z = \underbrace{00}_{u}\underbrace{1}_{v}\underbrace{222}_{w}$

|||ly if $i = 2$ then

$z = uv^i w$

$z = uv^2 w$

$z = 0011\,222$ where $t \neq n+m$

again $z \notin L$

Hence our assumption of L being regular is wrong.

Hence $L = 0^n 1^m 2^{n+m}$ is not a regular language.

(iii) consider $L = \{1^K \mid K = n^2\}$ be a regular language

This language means that the number of 1's is always a perfect square.

If we take $n=1$ then

$L = 1^K$

$= 1^{n^2}$

$= 1^{1^2}$

$= 1$ the length is $1^2$.

Now consider, $n=2$

$L = 1^{2^2}$

$= 1$ the length is $2^2$

$z = 1^{2^2}$

$z = 1111$ Let $u = 1$
$v = 1$
$w = 11$

By pumping lemma if we pump v then also 1 should be perfect square.

$z = uv^i w$

$= (1)(1)(11)$ (for $i=1$)

$z = uv^2 w$

$= (1)(11)(11) = 11111$ (for $i=2$)

the length is not a perfect square.

As After pumping something onto the language, it is not showing the same property.

∴ The language is not regular.

(iv) Consider $L_1/L_2 = \{x \mid$ for some $y \in L_2$ and $xy \in L_1\}$

Now assume $x = 11$ and $y = 11$

Then $\frac{L_1}{L_2} = \frac{xy}{x}$

$= \frac{1111}{11} = 11$ which is $x$.

Now if we pump the value of $y$ we should get $x$ as a result of $L_1/L_2$.

Then only the language is said to be regular.

Consider $y^2 = 1111$ then

Here $x = 11$ and $y^2 = 1111$

$L_1/L_2 = 111111/1111$

$= 11$ which is again $x$.

Thus even after pumping some value of $x$ we get $L_1/L_2 = x$. This shows that the language is regular.

Ex- (6) Using pumping lemma. prove whether the following languages are regular.

(i) $L = \{a^n b^{2n} \mid n > 0\}$.

Sol^n Let $L = a^n b^{2n}$ be a regular language.

Then by pumping lemma,

$|z| = |uvw| \in L$

$z = a^n b^{2n}$

$|z| = n + 2n = 3n$

That means length is always multiple of 3.

By pumping lemma,

$|uvvw| = 3n+1$

We do not get the length of resultant string in multiple of 3.

This shows that $z \notin L$.

This is not a regular language.

(ii) Let us assume that

Given $L = a^n b^{2m} \mid 0<n<m$ is a regular language.

That means. $|z| = |uvw| \in L$

If we assume string z as

$z = a^n b^{2m}$ then after applying pumping lemma following cases are possible.

Case : 1

as $n<m$

when $n=2$ and $m=3$

$z = uv^iw$

$z = a^n b^{2m}$

$z = \underbrace{aa}_{u}\underbrace{bbb}_{v}\underbrace{bbb}_{w}$

If $z = uvvw$ then

$z = aa\,bbb\,bbb\,bbb$

$= a^2 b^9$

Here $9 \neq 2m$ That means this language $\notin L$.

Case 2:

Let

$z = aabbbbbb$ with $n=2$, $m=3$ as $n<m$

By pumping lemma

$$z = \underbrace{aa}_{u}\underbrace{bbb}_{v}\underbrace{bbb}_{w}$$

If $z = uv^iw$ with $i=0$ then

$z = uw$

i.e $z = aa\,bbb$

$= a^2b^3 \neq a^n b^{2m}$

That means given language is not regular.

from the above two cases it is proved that

$L = a^n b^{2m}$ is not regular.

(7) prove or disprove that the language L given by $L = \{a^m b^n \mid m \neq n$, m and n are positive integers$\}$ is regular.

Sol$^n$ Let $L = \{a^m b^n \mid m \neq n\}$ be a language

Assume L is regular language.

Now Consider

Case 1: Assume $z = aaabbbb \in L$

By pumping lemma if $z = uvw$ then if we pump some strings to make $z = uv^iw$ then if $z \in L$ then such a language is called regular.

Let,

$$z = \underbrace{a}_{u}\underbrace{aabb}_{v}\underbrace{bb}_{w}.$$

if $z = uv^iw$ and if $i = 2$ then $z = uvvw$.

$z = aaabbaabbbb$

$z = a^5b^6 \in L$.

<u>Case 2</u>: Assume $L \in z$ such that

$$z = \underbrace{a}_{u}\underbrace{aa}_{v}\underbrace{abbbbbb}_{w}.$$

By pumping lemma, $z = uv^iw \in L$.

If $i = 2$ then

$z = uvvw$

$= aaaaaabbbbbb.$

$z = a^6b^6 \in a^mb^n$

But $m \neq n$.

Thus our assumption of L being regular is wrong.

Hence given language $L = a^mb^n | m \neq n$ is not regular

## CLOSURE PROPERTIES OF REGULAR LANGUAGES :

If certain languages are regular and language L is formed ~~from~~ from them by certain operations (such as union or concatenation) then L is also regular. These properties are called closure properties of regular languages.

-The closure properties express the idea that when one or many languages are regular then certain related languages are also regular.

The closure properties of regular languages are as given below.

1. The Union of two regular languages is regular.
2. The intersection of two regular languages is regular.
3. The complement of a regular language is regular
4. The difference of two regular languages is regular
5. The reversal of a regular language is regular
6. The closure operation on a regular language is regular
7. The concatenation of regular language is regular.
8. A homomorphism of regular languages is regular.
9. The inverse homomorphism of regular language is regular.

## Closure of Regular Languages under Boolean operations:

(1) Let L and M be languages over alphabet $\Sigma$. Then $L \cup M$ is the language that contains all strings that are in either or both of L and M.

(2) Let L and M be languages over alphabet $\Sigma$. Then $L \cap M$ is the language that contains all strings that are in both L and M.

(3) Let L be a language over alphabet $\Sigma$. Then $\bar{L}$, the complement of L, is the set of strings in $\Sigma^*$ that are not in L.

### Closure under Union:

<u>Theorem</u>: If L and M are regular languages, then $L \cup M$ is regular

<u>proof</u>: This proof is simple.

Since L and M are regular, they have regular exp$^{ns}$:

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

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

$= L(R+S)$

### Closure under complementation:

Start with a regular expression, and find a regular exp$^{n}$ for its complement.

(1) Convert the regular expression to an $\epsilon$-NFA

(2) Convert that $\epsilon$-NFA to a DFA by the subset construction.

(3) Complement the accepting states of that DFA.

(4) Turn the complement DFA back into a regular expression

using

Theorem: If $L$ is a regular language over alphabet $\Sigma$, then $\bar{L} = \Sigma^* - L$ is also a regular language.

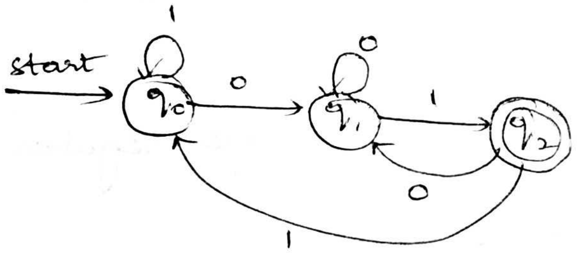
Proof: Let $L = L(A)$ for some DFA. $A = (Q, \Sigma, \delta, q_0, F)$.

Then $\bar{L} = L(B)$, where B is the DFA $(Q, \Sigma, \delta, q_0, Q-F)$

That is, B is exactly like A, but the accepting states of A have become non accepting states of B, and vice versa.

Then w is in $L(B)$ if and only if $\hat{\delta}(q_0, w)$ is in $Q-F$, which occurs if and only if w is in $L(A)$.

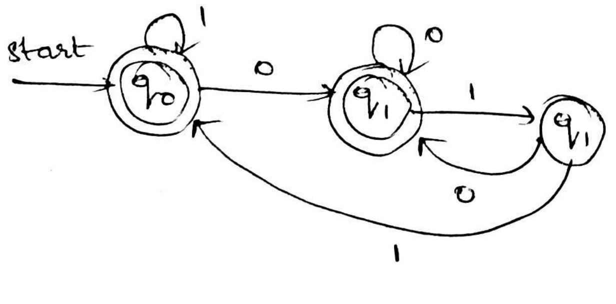
Example: A be the automaton given below.



The above DFA A accepts all and only the strings of 0's and 1's that end in 01; In regular expression terms, $L(A) = (0+1)^*01$.

- The complement of $L(A)$ is all strings of 0's and 1's that do not end in 01.

- The below figure shows the complement of the language $(0+1)^*01$.



$\therefore \quad \bar{L} = \Sigma^* - L(A)$

$\bar{L} = \{0,1\}^* - L(A)$

closure under Intersection:

We can obtain the intersection of languages L and M by the identity $L \cap M = \overline{\bar{L} \cup \bar{M}}$

In general the intersection of two sets is the set of elements that are not in the complement of either set. It is one of De-morgan's Law.

The other law is the same with union and that is $L \cup M = \overline{\bar{L} \cap \bar{M}}$.

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

Proof: Let L and M be the languages of automata

$A_L = (Q_L, \Sigma, \delta_L, q_L, F_L)$ and

$A_M = (Q_M, \Sigma, \delta_M, q_M, F_M)$.

Notice that we are assuming that the alphabets of both automata are the same. that is $\Sigma$ is the union of the alphabets of L and M.

We assume $A_L$ and $A_M$ are DFA's.

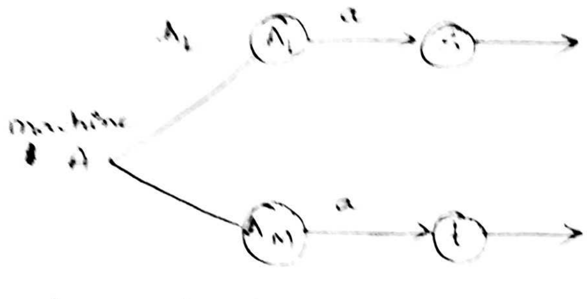
- For $L \cap M$ we shall construct an Automaton A that simulates both $A_L$ and $A_M$.

- The states of A are pair of states, the first from $A_L$ and the second from $A_M$.

- To design the transitions of A, suppose A is in state (p, q), Where p is the state of $A_L$ and

q is the state of $A_M$.

- If a is the input symbol, we see that $A_L$ ~~does on~~ goes to state s.



- If a is the input symbol in $A_M$ then it makes a transition to state t.

- Then the next state of A will be (s, t).

In that manner A has simulated the effect of both $A_L$ and $A_M$.

- We select the accepting states of A as all those pair of states (p, q) such that p is in accepting state of $A_L$ and q is in an accepting state of $A_M$.

Formally, we define:

$$A = (Q_L \times Q_M, \Sigma, \delta, (q_L, q_M), F_L \times F_M)$$

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

$$\hat{\delta}((q_L, q_M), w) = (\hat{\delta}_L(q_L, w), \hat{\delta}_M(q_m, w))$$

But A accepts w if and only if $\hat{\delta}((q_L, q_M), w)$ is a pair of accepting states.

That is $\hat{\delta}_L(q_L, w)$ must be in $F_L$.

and $\hat{\delta}_M(q_M, w)$ must be in $F_M$.

In another way, w is accepted by A if and only if both $A_L$ and $A_M$ accept w.

Thus, A accepts intersection of L and M.

Closure under Difference :

The fourth operation is difference.

In terms of languages L-M, the difference of L and M, is the set of strings that are in language L but not in language M.

<u>Theorem</u>: If L and M are regular languages then so is L-M.

<u>proof</u>: observe that $L-M = L \cap \bar{M}$. By theorem (complementation) $\bar{M}$ is regular and by theorem (Intersection) $L \cap \bar{m}$ is regular.

Therefore L-M is regular.

Closure under Reversal :

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

Eg:- $L = \{0, 01, 100\}$

$L^R = \{0, 10, 001\}$

Proof: Let E be a regular expression for L

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

Basis: If E is a symbol a, ε or φ then

$E^R = E$

Induction: If E is

1) $F+G$, then $E^R = F^R + G^R$

2) $FG$ then $E^R = G^R F^R$

3) $F^*$ then $E^R = (F^R)^*$

Eg: Let $E = 01^* + 10^*$

$E^R = (01^* + 10^*)^R$

$= (01^*)^R + (10^*)^R$

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

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

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

Homomorphism: A homomorphism on an alphabet is a function that gives a string for each symbol in that alphabet.

(or)

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

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

Eg:- $h(01010) = ababab$.

closure under homomorphism:

If L is a regular language and h is a homomorphism on its alphabet, then $h(L) = \{h(w) \mid w \text{ is in } L\}$ is also a regular language.

Proof: Let E be a reg$^l$ exp$^n$ for L

Apply h to each symbol in E

language of resulting R.E is in h(L).

Let $h(0) = ab$, $h(1) = \varepsilon$.

Let L be a language of regular expression $01^* + 10^*$.

Then h(L) is the language of regular expression.

$ab\varepsilon^* + \varepsilon(ab)^*$

It can be simplified.

$\varepsilon^* = \varepsilon$ , so $ab\varepsilon^* = ab\varepsilon$

$\varepsilon$ is the identity under concatenation.

i.e $\varepsilon R = R\varepsilon = R$.

Thus, $ab\varepsilon^* + \varepsilon(ab)^*$

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

$= ab + (ab)^*$

Finally L(ab) is contained in L((ab)*)

So RE for h(L) is $(ab)^*$

# DECISION PROPERTIES OF REGULAR LANGUAGES:

- We consider how one answers important questions about regular languages by reviewing the ways we can convert one representation into another for the same language.
- In particular, we want to observe the time complexity of the algorithms that perform the conversions.
- We then consider some of the fundamental questions about the languages.

1) Is the language described empty? $(L = \phi?)$
2) Is a particular string w in the described language? $w \in L$?
3) Does the two descriptions of a language actually describe the same language?

This question is often called "equivalence of languages".

Converting among Representations:

We shall consider the time complexity of each of the conversions we discussed.

(i) converting NFA's to DFA's:

When we start with either an NFA or an ε-NFA and convert it to a DFA, the time can be exponential in the number of states of the NFA

- To compute ε-closure(P) we follow at most $n^2$ arcs.

The DFA has $2^n$ states, for each state S and each $a \in \Sigma$ we compute $\delta(S, a)$ in $n^3$ steps.

Grand total is $O(n^3 2^n)$ steps.

- If we compute δ for reachable states only, we need to compute $\delta(S, a)$ only s times, where s is the no of -

reachable states.

Grand total is $O(n^3 s)$ steps.

From DFA to NFA :

All we need to do is to put set brackets around the states.

Total $O(n)$ steps.

From FA to regular Expr$^n$ :

We need to compute $n^3$ entries of size upto $4^n$. Total is $O(n^3$

Total is $O(n^3 4^n)$

From regular expression to FA :

We can build an expression tree for the reg$^l$ exp$^n$ in ~~n~~ steps

n steps.

We can construct the automaton in n steps.

Eliminating ε transitions takes $O(n^3)$ steps.

Testing emptiness :

$L(A) \neq 0$ for FA A if and only if a final state is reachable from the start state in A. Total $O(n^2)$ steps.

- We can inspect a regular expression E and tell if $L(E) = \phi$. We use the following method.

- $E = F + G$. Now $L(E)$ is Empty if and only if both $L(F)$ and $L(G)$ are Empty.

- $E = F \cdot G$. Now $L(E)$ is Empty if and only if either $L(F)$ or $L(G)$ is Empty.

- $E = F^*$. Now $L(E)$ is never Empty, since $\epsilon \in L(E)$.

$E = \epsilon$ Now $L(E)$ is not empty

$E = a$. Now $L(E)$ is not empty

$E = \phi$. Now $L(E)$ is empty.

## Testing membership:

To test $w \in L(A)$ for DFA A, simulate A on w, if $|w| = n$, this takes $O(n)$ steps.

If A is an NFA and has s states, simulating A on w takes $O(ns^2)$ steps.

If A is an $\epsilon$-NFA and has s states, simulating A on w takes $O(ns^3)$ steps.

If $L = L(E)$, for reg exp$^n$ E of length s, we first convert E to an $\epsilon$-NFA with 2s states. Then we simulate w on this machine, in $O(ns^3)$ steps.

# MINIMIZATION OF FSM :

Testing Equivalence states :

(i) Indistinguishable states : Two states P and q of a DFA are called indistinguishable if,

$\delta(P, w) \in F$ implies $\delta(q, w) \in F$

and $\delta(P, w) \notin F$ implies $\delta(q, w) \notin F$

for all $w \in \Sigma^*$

(ii) Distinguishable states : There exists some string $w \in \Sigma^*$ such that $\delta(P, W) \in F$ and $\delta(q, w) \in F$ or vice versa then states P and q are said to be distinguishable by a string w.

(iii) Equivalence : Two states $q_1$ and $q_2$ are equivalent (denoted by $q_1 \equiv q_2$) if both $\delta(q_1, x)$ and $\delta(q_2, x)$ are final states, or both of them are non final states for all $x \in \Sigma^*$.

(iv) K-Equivalence : Two states $q_1$ and $q_2$ are K-equivalent ($K \geq 0$) if both $\delta(q_1, x)$ and $\delta(q_2, x)$ are final states or both of non final states for all strings x of length K or less.

Minimization :

For a given language DFA can be represented in many ways differing with number of states. Always less numbered DFA is preferable because it requires less space and is easily processed.

- so it is desirable to reduce the no. of states as far as possible.

- one method for reducing the states of a DFA is based on finding and combining indistinguishable states.

Procedure for construction of Minimum Automata.

Step1: (Construction of $\Pi_0$) partition of the states of given automaton as set of final states and set of non final states.

- By def^n of 0-equivalence $\Pi_0 = \{Q_1^0, Q_2^0\}$ where,

$Q_1^0$ - set of all final states

$Q_2^0 \rightarrow (Q - Q_1^0)$, the set of non final states.

Step2: (Construction of $\Pi_1, \Pi_2$ ......)

The subsets of $\Pi_1$ are obtained by partitioning subsets of $\Pi_0$ further.

If $q_1, q_2 \in Q_1$, consider the states in each a where $a \in \Sigma$, corresponding to $q_1$ and $q_2$. If those states are in the same subset of $\Pi_0$, $q_1$ and $q_2$ are 1-equivalent.

If the states under some a are in diff^t subsets of $\Pi_0$ then $q_1$ and $q_2$ are not $\Pi_2, \Pi_2$ ---- 1-equivalent.

By proceeding in this way, subsets are constructed.

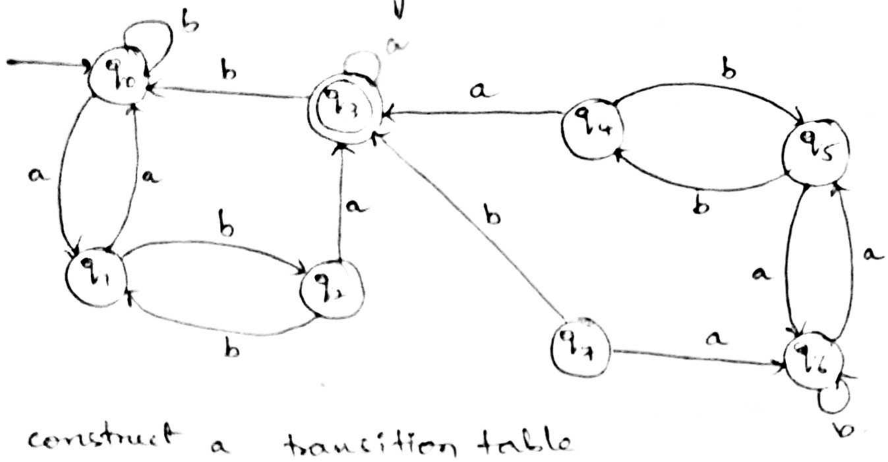
Step3: Construct $\Pi_n$ for $n=1, 2$ ---- until $\Pi_n = \Pi_{n+1}$

Step4: (Construction of minimum automaton)

For the required minimum state automaton, the states are the equivalence classes obtained in step 3.

## Minimization of FSM

Problem 1: construct minimum state automaton for the following transition diagram.



First construct a transition table

start constructing equivalence classes.

| | a | b |
|---|---|---|
| → $q_0$ | $q_1$ | $q_0$ |
| $q_1$ | $q_0$ | $q_2$ |
| $q_2$ | $q_3$ | $q_1$ |
| $q_3$ | $q_3$ | $q_0$ |
| $q_4$ | $q_3$ | $q_5$ |
| $q_5$ | $q_6$ | $q_4$ |
| $q_6$ | $q_5$ | $q_6$ |
| $q_7$ | $q_6$ | $q_3$ |

⇒ for the given automata, there is only one final state i.e $q_3$

⇒ We can divide ~~&~~ Q as $Q_1^0$ and $Q_2^0$.

$Q_1^0 = F = \{q_3\}$ ($q_3$ cannot be partitioned further).

$Q_2^0 = Q - Q_1^0$

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

$\pi_0 = \{\check{q}_0, \check{q}_1, \underline{q_2}, \underline{q_4}, q_5, q_6, \underline{q_7}\}$.

compare each state in $Q_2^0$ with other state in this set.

By comparing $q_0$ with $q_1$

|  | a | b |
|---|---|---|
| $q_0$ | $q_1$ | $q_0$ |
| $q_1$ | $q_0$ | $q_2$ |

If we consider column a
We have $(q_1, q_0) \in Q_2^0$
If we consider column b.
we have $(q_0, q_2) \in Q_2^0$.

so $q_0$ and $q_1$ are equal.

- compare state $q_0$ with $q_2$

$$\begin{array}{ccc} q_0 & q_1 & q_0 \\ q_2 & q_3 & q_1 \end{array}$$

$\Rightarrow (q_1, q_3) \notin Q_2^0$
$\Rightarrow$ They belong to diff$^t$ sets.

Hence $q_0 \neq q_2$.

- compare state $q_0$ with $q_4$

$$\begin{array}{ccc} q_0 & q_1 & q_0 \\ q_4 & q_3 & q_5 \end{array}$$

$\Rightarrow (q_1, q_3) \notin Q_2^0$.

$\therefore q_0 \neq q_4$

III$^{rly}$ when we compare $q_0$ with $q_5 \Rightarrow q_0 = q_5$
when we compare $q_0$ with $q_6 \Rightarrow q_0 = q_5$
when we compare $q_0$ with $q_7 \Rightarrow q_0 \neq q_7$.

$$\begin{array}{ccc} q_0 & q_1 & q_0 \\ q_7 & q_6 & q_3 \end{array}$$

$\Rightarrow (q_0, q_3) \notin Q_2^0$

$\therefore q_0 \neq q_7$

Now $Q_1^1 = \{q_3\}$

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

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

$Q_4^1 = \{q_7\}$

The equivalence class is

$\Pi_1 = \{\{q_3\}, \{q_0, q_1, q_5, q_6\}, \{q_2, q_4\}, \{q_7\}\}$

Now we will find 2-equivalent ie $\Pi_2$.

We start by taking

$Q'_1 = \{q_3\}$

$Q'_2 = \{q_0, q_1, q_5, q_6\}$

$Q'_3 = \{q_2, q_4\}$

$Q'_4 = \{q_7\}$

$Q'_1$ cannot be partitioned further

consider the states from $Q'_2$

compare $q_0$ with $q_1$

| | a | b |
|---|---|---|
| $q_0$ | $q_1$ | $q_0$ |
| $q_1$ | $q_0$ | $q_2$ |

→ does not belong to $Q'_2$ ∴ $q_0 \neq q_1$

compare $q_0$ with $q_5$

| | | |
|---|---|---|
| $q_0$ | $q_1$ | $q_0$ |
| $q_5$ | $q_6$ | $q_4$ |

→ does not belong to $Q'_2$ ∴ $q_0 \neq q_5$

compare $q_0$ with $q_6$

| | | |
|---|---|---|
| $q_0$ | $q_1$ | $q_0$ |
| $q_6$ | $q_5$ | $q_6$ |

→ $q_0 = q_6$.

compare $q_1$ with $q_5$

| | | |
|---|---|---|
| $q_1$ | $q_0$ | $q_2$ |
| $q_5$ | $q_6$ | $q_4$ |

→ belong to $Q'_2$ ∴ $q_1 = q_5$

→ ~~does not~~ belong to $Q'_3$

compare $q_1$ with $q_6$

| | | |
|---|---|---|
| $q_1$ | $q_0$ | $q_2$ |
| $q_1$ | $q_5$ | $q_6$ |

→ does not belong to $Q'_2$ ∴ $q_1 \neq q_6$

$Q^2_1 = \{q_3\}$

$Q^2_2 = \{q_0, q_6\}$

$Q^2_3 = \{q_1, q_5\}$

IIIrly $q_2$ is eqvt to $q_4$

$Q^2_4 = \{q_2, q_4\}$

$Q^2_5 = \{q_7\}$

Thus $\Pi_2 = \{\{q_3\}, \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}, \{q_7\}\}$

$Q^3_1 = \{q_3\}$

$Q^3_2 = \{q_0, q_6\}$

$Q^3_3 = \{q_1, q_5\}$

$Q^3_4 = \{q_2, q_4\}$

$Q^3_5 = \{q_7\}$

$\therefore \Pi_3 = \{\{q_3\}, \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}, \{q_7\}\}$

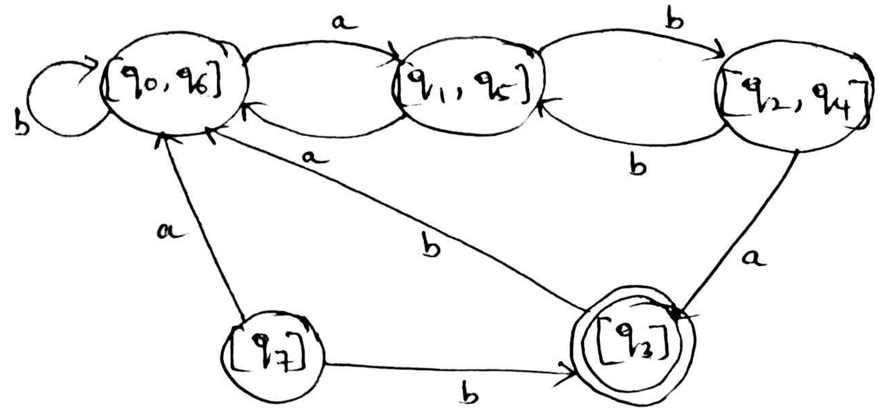
$\Pi_3 = \Pi_2$

We have got equivalence class from $\Pi_2$

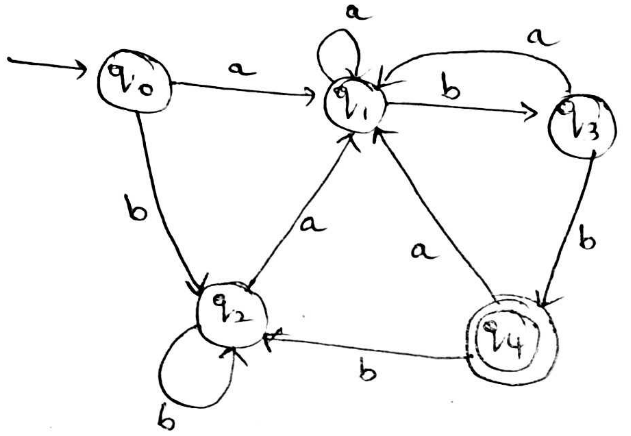
Hence we can write transition table as.

| | a | b |
|---|---|---|
| $\{q_0, q_6\}$ | $[q_1, q_5]$ | $[q_0, q_6]$ |
| $\{q_1, q_5\}$ | $[q_0, q_6]$ | $[q_2, q_4]$ |
| $\{q_2, q_4\}$ | $[q_3]$ | $[q_1, q_5]$ |
| $\{q_3\}$ | $[q_3]$ | $[q_0, q_6]$ |
| $\{q_7\}$ | $[q_0, q_6]$ | $[q_3]$ |

The minimized transition diagram will be.



Problem 2 : construct minimum state DFA for the given automata.



sol^n first construct the transition table for the FA.

|  | a | b |
|---|---|---|
| → $q_0$ | $q_1$ | $q_2$ |
| $q_1$ | $q_1$ | $q_3$ |
| $q_2$ | $q_1$ | $q_2$ |
| $q_3$ | $q_1$ | $q_4$ |
| $q_4$ | $q_1$ | $q_2$ |

Now we will first obtain

$Q_1^0 = \{q_4\}$

$Q_2^0 = \{q_0, q_1, q_2, q_3, \cancel{q_4}\} - \{\cancel{q_4}\}$

$Q_2^0 = \{q_0, q_1, q_2, q_3\}$

$\Pi_0 = \{\{q_4\}, \{q_0, q_1, q_2, q_3\}\}$

$Q_1^0$ cannot be further partitioned.

Now we will partition $Q_2^0$.

$Q_2^0 = \{q_0, \check{q}_1, \check{q}_2, q_3\}$

Now compare $q_0$ with $q_1$

| | | |
|---|---|---|
| $q_0$ | $q_1$ | $q_2$ |
| $q_1$ | $q_1$ | $q_3$ |

$\in$ same state.

$\therefore$ $\boxed{q_0 = q_1}$

compare $q_0$ with $q_2$

| | | |
|---|---|---|
| $q_0$ | $q_1$ | $q_2$ |
| $q_2$ | $q_1$ | $q_2$ |

$\boxed{q_0 = q_2}$

Compare $q_0$ with $q_3$

| | | |
|---|---|---|
| $q_0$ | $q_1$ | $q_2$ |
| $q_3$ | $q_1$ | $q_4$ |

× $\notin Q_2^0$ so $q_0 \neq q_3$

Compare $q_1$ with $q_2$

| | | |
|---|---|---|
| $q_1$ | $q_1$ | $q_3$ |
| $q_2$ | $q_1$ | $q_2$ |

$\Rightarrow$ $\boxed{q_1 = q_2}$

Compare $q_1$ with $q_3$

| | | |
|---|---|---|
| $q_1$ | $q_1$ | $q_3$ |
| $q_3$ | $q_1$ | $q_4$ |

$\notin$ same set $\therefore$ $q_1 \neq q_3$

compare $q_2$ and $q_3$

| | | |
|---|---|---|
| $q_2$ | $q_1$ | $q_2$ |
| $q_3$ | $q_1$ | $q_4$ |

∉ same set ∴ $q_2 \neq q_3$.

$Q'_1 = \{q_4\}$

$Q'_2 = \{q_0, q_1, q_2\}$

$Q'_3 = \{q_3\}$

ence $\Pi_1 = \{\{q_4\}, \{q_0, q_1, q_2\}, \{q_3\}\}$

Now compare $\{\overset{\checkmark}{q_0}, q_1, \overset{\checkmark}{q_2}\}$

$q_0$ with $q_1$

| | | |
|---|---|---|
| $q_0$ | $q_1$ | $q_2$ |
| $q_1$ | $q_1$ | $q_3$ |

∉ same set

$q_0 \neq q_1$

compare $q_0$ with $q_2$

| | | |
|---|---|---|
| $q_0$ | $q_1$ | $q_2$ |
| $q_1$ | $q_1$ | $q_2$ |

→ $\boxed{q_0 = q_2}$

∴ $Q^2_1 = \{q_4\}$

$Q^2_2 = \{q_0, q_2\}$

$Q^2_3 = \{q_3\}$

$q^2_4 = \{q_1\}$

So $\Pi_2 = \{\{q_4\}, \{q_0, q_2\}, \{q_3\}, \{q_1\}\}$

We can now state 3-equivalent.

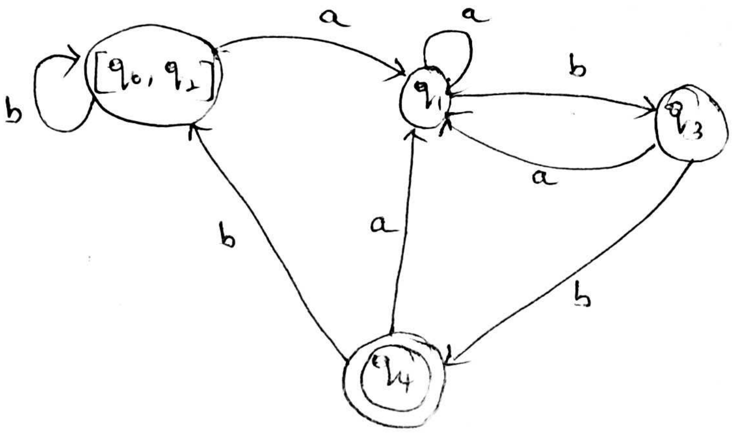
$\Pi_3 = \{\{q_4\}, \{q_0, q_2\}, \{q_3\}, \{q_1\}\}$

as $\Pi_3 = \Pi_2$. gives equivalence classes.

We can construct minimum state Automata as

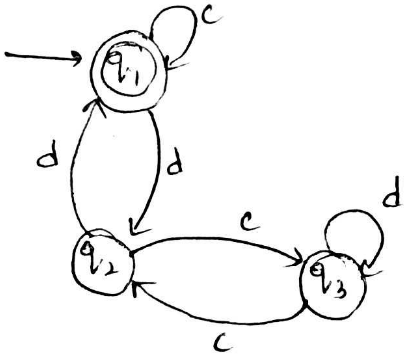
| | a | b |
|---|---|---|
| $[q_0, q_2]$ | $[q_1]$ | $[q_0, q_2]$ |
| $[q_4]$ | $[q_1]$ | $[q_0, q_2]$ |
| $[q_1]$ | $[q_1]$ | $[q_3]$ |
| $[q_3]$ | $[q_1]$ | $[q_4]$ |

The transition diagram will be

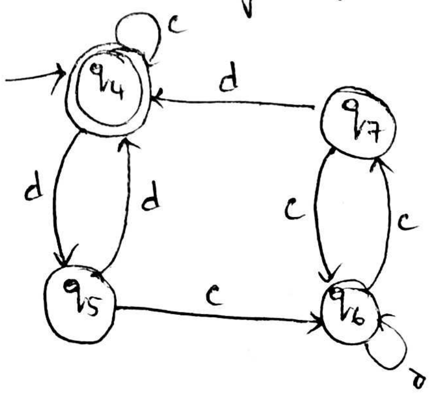


EQUIVALENCE B/W TWO FSM'S

<u>Problem 1</u> : consider DFAs given below, are they equivalent.



M



M'

<u>Soln</u> We build transition table for each i/p c and d.

$\delta(q_1, c) = q_1$ → final state in M

$\delta(q_4, c) = q_4$ → final state in M'.

IIIrly for i/p d in state $q_1, q_4$

$\delta(q_1, d) = q_2$ ~ non final state in M'

$\delta(q_4, d) = q_5$ ~ non final state in M'.

$q_1$ & $q_4$ are final states obtained in pair $(q_1, q_4)$

Both $q_2$ and $q_5$ are non final states obtained in pair $(q_2, q_5)$.

- we will obtain transition for $q_2, q_5$ for i/p c and d.

$(q_2, c) = q_3$ ← non final states.
$(q_5, c) = q_6$

$(q_2, d) = q_1$ ← both are ~~non~~ final states
$(q_5, d) = q_4$

$(q_3, c) = q_2$ ← non final.
$(q_6, c) = q_7$

$(q_3, d) = q_3$ ← non final.
$(q_6, d) = q_6$

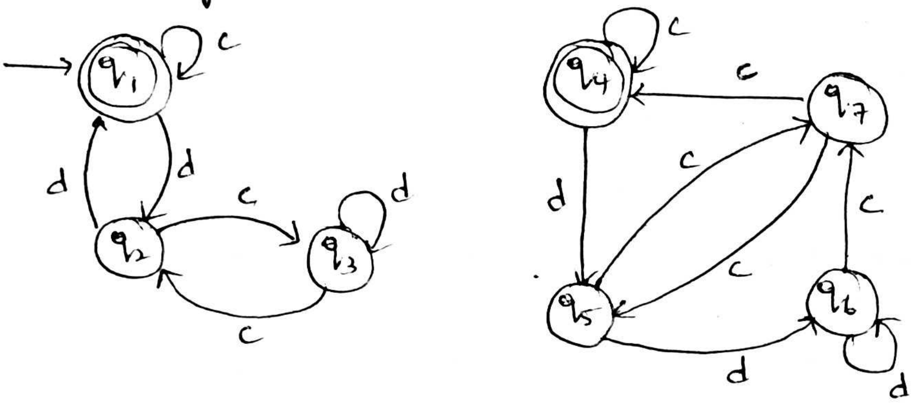
$(q_2, c) = q_3$ ← non final.
$(q_7, c) = q_6$

$(q_2, d) = q_1$ ← both are final.
$(q_7, d) = q_4$

Here we did not get one final state and one non final state in a pair.

→ Hence we declare two DFA's are equivalent.

Problem 2: Following are two FAs. check whether they are equivalent or not.



Sol^n We will design the transition table for each i/p symbol c and d

$q_1, c \rightarrow q_1$ $\quad$ $q_1, d \rightarrow q_2$

$q_4, c \rightarrow q_4$ $\quad$ $q_4, d \rightarrow q_5$

Now find $q_2, q_5$ on input c

$q_2, c \rightarrow q_3$ ← Both are non final states

$q_5, c \rightarrow q_7$

$q_2, q_5$ on input d.

$q_2, d \rightarrow q_1$ → final state

$q_5, d \rightarrow q_6$ → non final state.

Since we are getting a final state and a non final state in a pair, we terminate the construction of transition table.

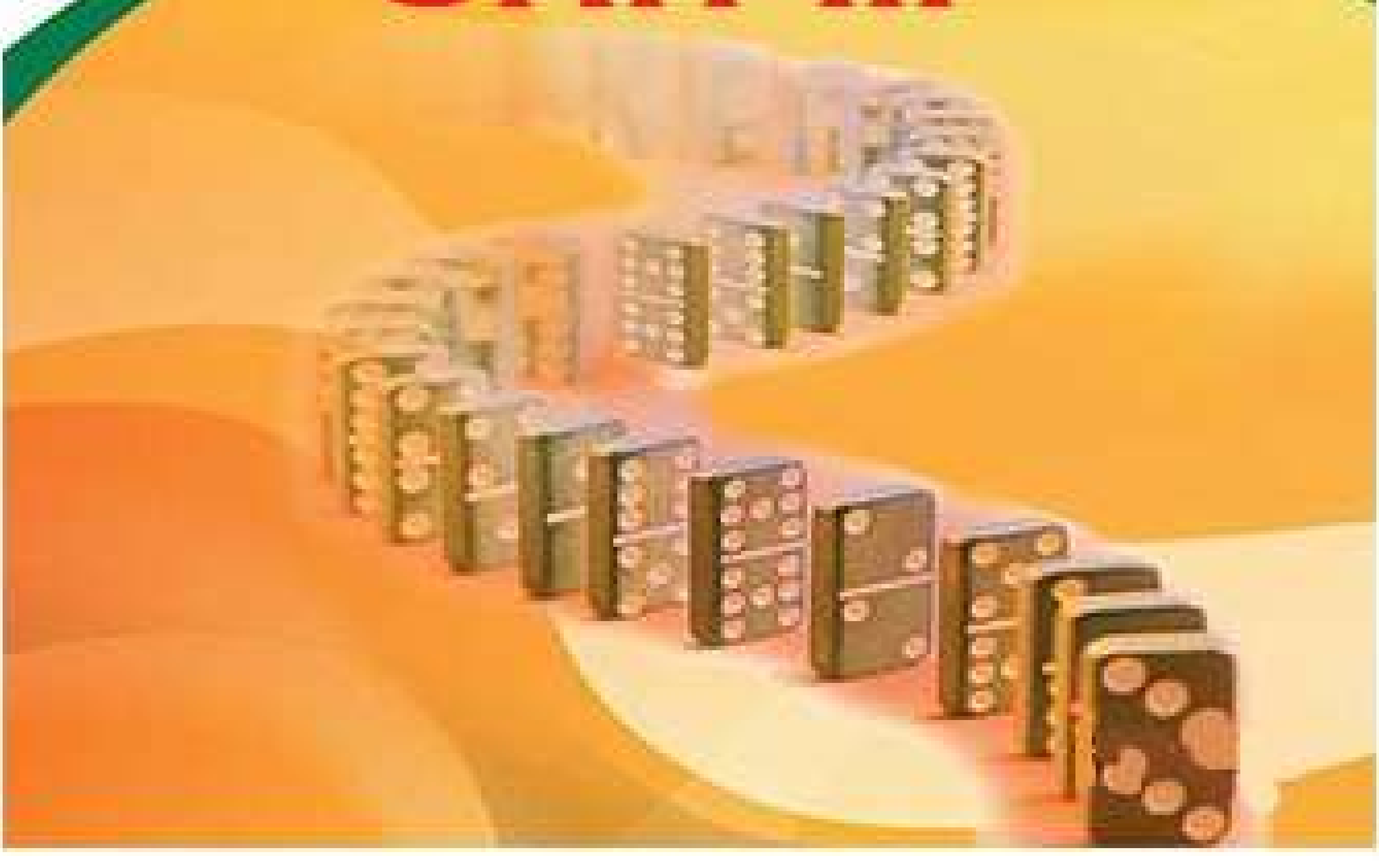
→ Hence the FA's are not equivalent.

JNTUH, R-18

MRIET

# Formal Languages and Automata Theory

## UNIT-III



# UNIT - III
# GRAMMAR FORMALISM

## Introduction:

We have three ways of characterising regular languages

(i) Regular Expressions

(ii) Finite Automata

(iii) construction of natural languages.

There is another way of expressing the regular languages, that is by using grammer.

GRAMMAR: Grammar is basically the set of rules used to define the language.

These rules can be rewritten which are used to generate the desired string

for eg:- consider the language represented by $1^+$

which can be represented by a set {1, 11, 111 - - - - }.

-To generate the strings of this language, we will use one simple method.

Let S be a symbol to start the process with.

Now we can write a rule with symbol S. as

S → 1

S → 1S

These rules means that S is rewritten as 1 or 1S.

- Now to generate a string 111, start with S, then replace S by 1S

then again apply the 2nd rule to replace S by 1S which gives 11S.

Now apply 1st rule S → 1 to replace S by 1

ultimately we will get the string 111.

Thus grammar is an effective way to represent the regular language.

## REGULAR GRAMMAR:

A regular grammar is defined as

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

V = set of symbols called non-terminals which are used to define the rules

T = set of symbols called Terminals

P = set of production rules.

S = start symbol

The production rules P are of the form

$A \longrightarrow aB$

$A \longrightarrow a$

where A and B are non terminal symbols

a is a terminal symbol.

For eg:-

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

$V = \{S, B\}$

$T = \{0, 1\}$

S is a start symbol and production rules are as given below.

$S \longrightarrow 0S$

$S \longrightarrow 1B$

$S \longrightarrow \varepsilon$.

This is called a regular language and it can be represented by some DFA.

Thus when a grammar can be rep^td by some FA, then it is a regular grammar

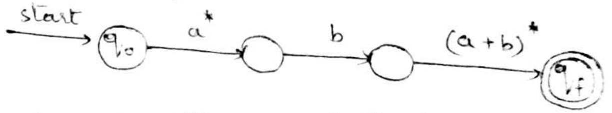
## CONSTRUCTION OF REGULAR GRAMMAR FOR R.E

We can convert the r.e into the equivalent regular grammar by using following method

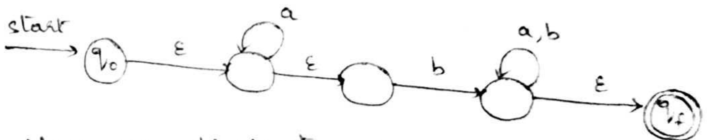
1. Construct a NFA with ε from given r.e
2. Eliminate ε transitions and convert it to Eq^t DFA.
3. From constructed DFA, the corresponding states become non terminal symbols and transitions made are eq^t to production rules.

Ex:1 construct a regular grammar for the r.e $a^*b(a+b)^*$

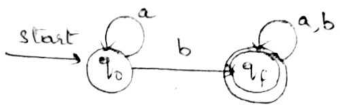
Sol^n We will first construct a DFA in a straight forward manner for given r.e



Now we will convert it to NFA with ε transitions.



Now we eliminate ε moves,



We can write the grammar as,

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

$V = \{A_0, A_1\}$

$T = \{a, b\}$

$P = \{ A_0 \longrightarrow aA_0 \quad A_1 \longrightarrow a$

$A_0 \longrightarrow bA_1 \quad A_1 \longrightarrow b$

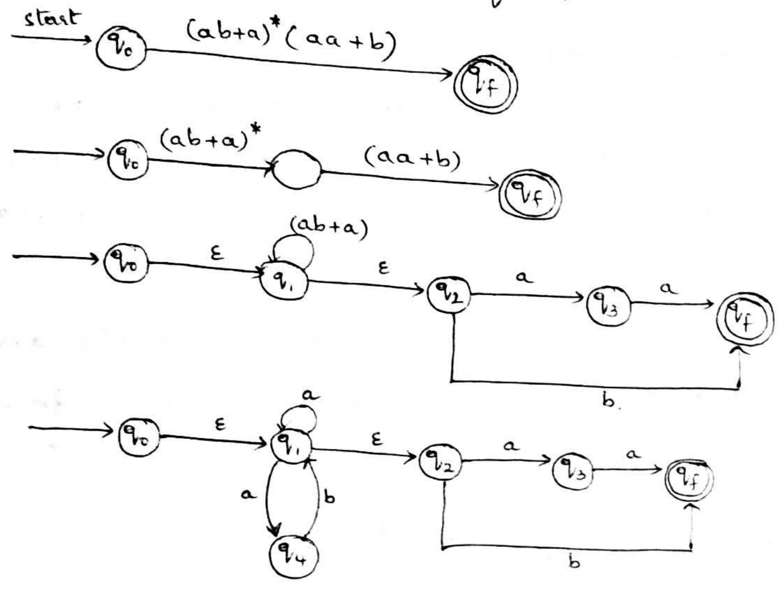
$A_1 \longrightarrow aA_1$ [redacted]

$A_1 \longrightarrow bA_1 \quad A_0 \longrightarrow b \}$

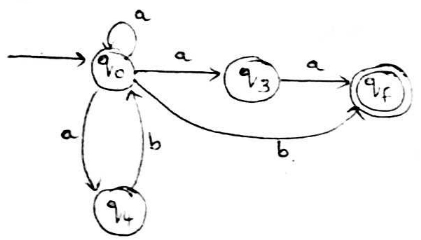
**Ex:2** construct a regular grammar for $(ab+a)^*(aa+b)$.

**Sol**

Soln We will first construct FA for given r.e



If we eliminate ε transitions then the NFA will be -



The transition table can be -

| state\Input | a | b |
|---|---|---|
| $q_0$ | $\{q_0\ q_3\ q_4\}$ | $q_f$ |
| $q_3$ | $q_f$ | $\phi$ |
| $q_4$ | $\phi$ | $q_0$ |
| $q_f$ | $\phi$ | $\phi$ |

We can convert this NFA to equivalent DFA as -

$\delta'(q_0, a) = \{q_0, q_3, q_4\}$

$= [q_0, q_3, q_4]$ ⟶ new state generated.

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

$= [q_f]$

We will find input transitions on $[q_0, q_3, q_4]$

$\delta'([q_0, q_3, q_4], a) = \delta(q_0, a) \cup \delta(q_3, a) \cup \delta(q_4, a)$

$= \{q_0, q_3, q_4\} \cup \{q_f\} \cup \phi$

$= \{q_0, q_3, q_4, q_f\}$ ⟶ new state

$\delta'([q_0, q_3, q_4], b) = \delta(q_0, b) \cup \delta(q_3, b) \cup \delta(q_4, b)$

$= \{q_f\} \cup \phi \cup \{q_0\}$

$= \{q_0, q_f\}$ ⟶ new state.

$\therefore \delta'([q_0, q_3, q_4, q_f], a) = \delta(q_0, a) \cup \delta(q_3, a) \cup \delta(q_4, a) \cup (q_f, a)$

$= \{q_0, q_3, q_4\} \cup \{q_f\} \cup \phi \cup \phi$

$= \{q_0, q_3, q_4, q_f\}$

$\delta'([q_0, q_3, q_4, q_f], b) = \delta(q_0, b) \cup \delta(q_3, b) \cup \delta(q_4, b) \cup \delta(q_f, b)$

$= \{q_f\} \cup \{\phi\} \cup \{q_0\} \cup \phi$

$= \{q_0, q_f\}$

$\delta'([q_0, q_f], a) = \delta(q_0, a) \cup \delta(q_f, a)$

$= \{q_0, q_3, q_4\} \cup \phi$

$= \{q_0, q_3, q_4\}$

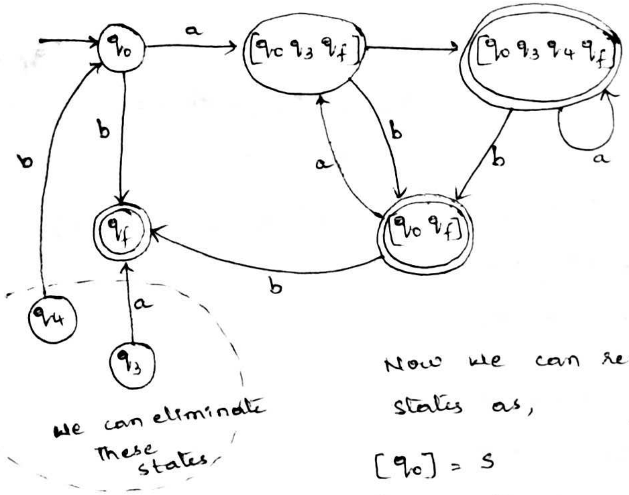
$\delta'([q_0, q_f], b) = \delta(q_0, b) \cup \delta(q_f, b)$

$= \{q_f\} \cup \phi$

$= \{q_f\}$

- transition table for this DFA will be

| state \ Input | a | b |
|---|---|---|
| → [$q_0$] | [$q_0, q_3, q_4$] | [$q_f$] |
| [$q_3$] | [$q_f$] | $\phi$ |
| [$q_4$] | $\phi$ | [$q_0$] |
| [$q_f$] | $\phi$ | $\phi$ |
| [$q_0, q_3, q_4$] | [$q_0 q_3 q_4 q_f$] | [$q_0 q_f$] |
| [$q_0, q_3, q_4, q_f$] | [$q_0 q_3 q_4 q_f$] | [$q_0 q_f$] |
| [$q_0, q_f$] | [$q_0 q_3 q_4$] | [$q_f$] |



Now we can rename, the states as,

[$q_0$] = S

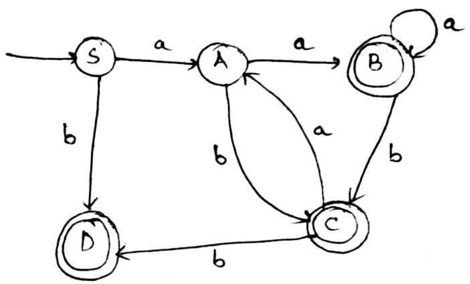
[$q_0, q_3, q_4$] = A

[$q_0, q_3, q_4, q_f$] = B

[$q_0, q_f$] = C

[$q_f$] = D.

The DFA with renamed states will be -



The regular grammar can be

S ⟶ aA
S ⟶ bD
S ⟶ b.
A ⟶ aB
A ⟶ a
A ⟶ bC
A ⟶ b

B ⟶ aB
B ⟶ a
B ⟶ bC
B ⟶ b

C ⟶ aA
~~C ⟶ a~~
C ⟶ bD
C ⟶ b
D ⟶ ε.

## RIGHT LINEAR AND LEFT LINEAR GRAMMAR

If the non terminal symbol appears as a right most symbol in each production of regular grammar, then it is called right linear grammar.

- The right linear grammar is of the form

A ⟶ aB
A ⟶ a
A ⟶ ε

where A and B are non terminal symbols and 'a' is a terminal symbol.

- If the non terminal symbol appears as a leftmost symbol in each production of regular grammer then it is called left linear grammar.

The left linear grammar is of the following form.

$A \rightarrow Ba$

$A \rightarrow a$

$A \rightarrow \varepsilon$.

where A, B are non terminal symbols.
a is a terminal symbol.

- The language is called regular if it is accepted by either left linear or right linear grammar.

CONSTRUCTION OF REGULAR GRAMMAR FROM FA :

Let $M = (\{q_0, q_1 --- q_n\}, \Sigma, \delta, q_0, F)$ be a DFA.

The equivalent grammar G can be constructed from this DFA such that productions should correspond to transitions.

Let $G = (\{A_0, A_1 ---- A_n\}, \Sigma, P, A_0)$

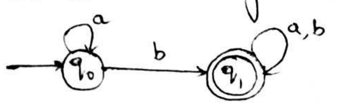
Where P the set of prodn rules can be defined by following rules.

1. $A_i \rightarrow aA_j$ is a prodn rule if $\delta(q_i, a) = q_j$, where $q_j \notin F$

2. $A_i \rightarrow aA_j$ and $A_i \rightarrow a$ are prodn rules if $\delta(q_i, a) = q_j$ where ~~$q_j \notin F$~~ $q_j \in F$.

Thus the given grammar is accepted by DFA M.

Problem 1: Construct regular grammar for given DFA.



Soln The grammar eqvt to given DFA will be

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

$V = \{A_0, A_1\}$

$T = \{a, b\}$

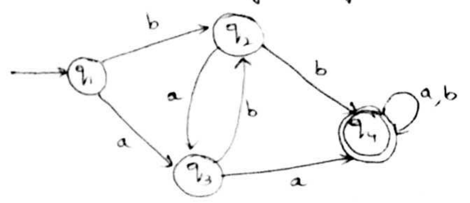
$A_0 \rightarrow aA_0 \qquad A_1 \rightarrow aA_1 | a$

$A_0 \rightarrow bA_1 \qquad A_1 \rightarrow bA_1 | b.$

$A_0 \rightarrow b$

Thus 2 states correspond to 2 non terminals.

Problem 2: construct a regular grammar for given FA.



Soln Let $G = (\{A_1, A_2, A_3, A_4\}, \{a, b\}, P, A_1)$

The regular grammar can be

$A_1 \rightarrow bA_2$

$A_1 \rightarrow aA_3$

$A_2 \rightarrow bA_4$

$A_2 \rightarrow b$

$A_2 \rightarrow aA_3$

$A_3 \rightarrow bA_2$

$A_3 \rightarrow aA_4$

$A_3 \rightarrow a$

$A_4 \rightarrow aA_4$

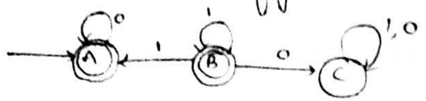
$A_4 \rightarrow a$

$A_4 \rightarrow bA_4$

$A_4 \rightarrow b$

As state $q_4$ is a final state the corresponding non terminal $A_4$ is used. Now all the edges leading to $q_4$ gives terminal productions as well.

Problem 3: a) obtain regular grammar for the following FA as shown in figure.



(b) What is the language generated by above FA.

Sol<sup>n</sup> a) We can write regular grammar as

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

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

$T = \{0, 1\}$

$P = \{ A \longrightarrow 0A$

$B \longrightarrow 1B$

$B \longrightarrow 1A$

$C \longrightarrow 1C$

$C \longrightarrow 0C$

$A \longrightarrow 0$

$B \longrightarrow 1 \}$

The non terminal A is a start symbol.

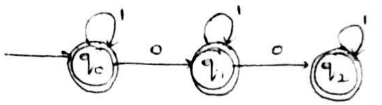
(b) For finding out the type of language it generates, we will derive some strings from the grammar.

| | | | |
|---|---|---|---|
| A | A | A | A |
| | | 0 A | 0A |
| 0 | | 0 0 A | 000A |
| | | 0 0 0 A | 0000 |
| | | 0 0 0 0 A | |
| | | 0 0 0 0 0 | |

That means the given FA generates any no. of 0's. Hence the language containing strings having any no. of 0's is accepted by the given FA.

Problem 4: obtain a regular grammar to obtain the set of all strings not containing three consecutive 0's.

Sol<sup>n</sup> We will first design FA for accepting the strings not containing three consecutive 0's.



Now from this FA we can write the regular grammar as

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

$T = \{0, 1\}$

$P = A \longrightarrow 1A \mid 0B \mid 1 \mid 0$

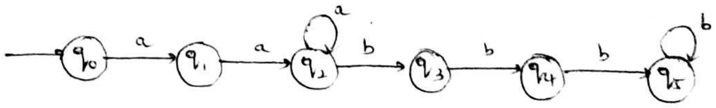
$B \longrightarrow 1B \mid 0C \mid 1$

$C \longrightarrow 1C \mid 1$

Problem 5: obtain a Right Linear Grammar for the language.

$L = \{ a^n b^m \mid n >= 2, m >= 3 \}$

Sol<sup>n</sup> We will design DFA for $L = \{ a^n b^m \mid n >= 2, m >= 3 \}$



from this FA we can write right linear grammar as

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

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

$T = \{a, b\}$

$P \Rightarrow A \rightarrow aB$

$B \rightarrow aC$

$C \rightarrow aC$

$C \rightarrow bD$

$D \rightarrow bE$

$E \rightarrow bF$

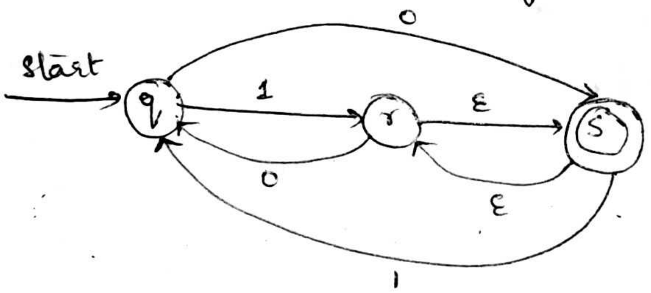
$E \rightarrow b$

$F \rightarrow bF$

$F \rightarrow b.$

The start symbol is A.

problem 6: Find regular grammar for the following F.A.



sol<sup></sup>ⁿ We will remove the ε-transitions from given grammar. Hence we will find ε-closure on the states.

ε-closure(q) = {q} — state

ε-closure(r) = {r, s} → [rs] state.

ε-closure(s) = {r, s} = [rs]

Now, we convert this to DFA

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

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

$= \{r, s\}$

$\delta'(q,0) = [rs]$

$\delta'(q,1) = \varepsilon\text{-closure}(\delta(q,1)).$

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

$\delta'(q,1) = [rs]$

$\delta'(\{r,s\},0) = \varepsilon\text{-closure}(\delta(r,s),0)$

$= \varepsilon\text{-closure}(\delta(r,0) \cup \delta(s,0))$

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

$\delta'(\{r,s\},0) = [q]$

$\delta'(\{r,s\},1) = \varepsilon\text{-closure}(\delta(r,s),1)$

$= \varepsilon\text{-closure}(\delta(r,1) \cup \delta(s,1))$

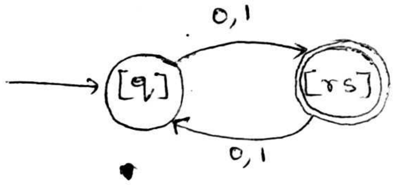
$= \varepsilon\text{-closure}(\phi \cup q)$

$\delta'(\{r,s\},1) = [q]$

Hence DFA will be (transition table)

| | States \ Input | 0 | 1 |
|---|---|---|---|
| → | [q] | [rs] | [rs] |
| * | [rs] | [q] | [q] |

The DFA Will be



Now we can write the regular grammar as -

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

$V = \{A, B\}$

$T = \{0, 1\}$

P is set of production rules, those are -

$A \longrightarrow 0B$

$A \longrightarrow 1B$

$A \longrightarrow 0$

$A \longrightarrow 1$

$B \longrightarrow 0A$

$B \longrightarrow 1A.$

# CONSTRUCTION OF F.A FROM REGULAR GRAMMAR

Let $G = (V, \Sigma, P, A_0)$ be a regular grammar.

We can construct DFA M whose

(i) states correspond to variables

(ii) Initial state Corresespond to productions in P.

If there is a production of the form $A_i \to a$

the corresponding transition terminates at a new state.

This is the unique final state.

Thus the DFA M can be

$$M = (\{q_0, q_1, \cdots q_n, q_f\}, \Sigma, \delta, q_0, \{q_f\})$$

The $\delta$ is defined as

(i) Each production $A_i \to aA_j$ induces a transition from $q_i$ to $q_j$ with label a

(ii) Each production $A_k \to a$ induces a transition from $q_k$ to $q_f$ with label a.

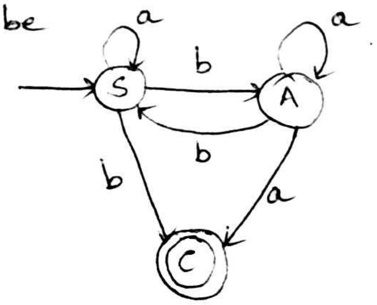
Therefore L(G) can be given by corresponding F.A M.

① Construct F.A recognizing L(G), where G is the grammar

$S \to aS \mid bA \mid b.$

$A \to aA \mid bS \mid a$

Soln The FA will be



This language generates { ab, abbb, abaabb ..... }

The FA accepts a language containing any number of a's and odd no. of b's.

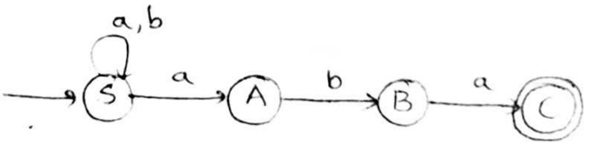
(2) construct DFA equivalent to the grammar $S \rightarrow aS \mid bS \mid aA$
$A \rightarrow bB$ , $B \rightarrow aC$ , $C \rightarrow \epsilon$.

Soln The DFA can be constructed using following rules.

1) Each production $A_i \rightarrow aA_j$ induces a transition from $q_i$ to $q_j$ with label a on the edge.

2) Each production $A_K \rightarrow a$ induces a transition from $q_K$ to $q_f$ with label a on the edge.

The DFA can be



## CONVERSION OF RIGHT LINEAR GRAMMAR TO LEFT-LINEAR GRAMMAR

Method

1. convert the given right linear grammar to equivalent F.A
2. From the FA interchange the initial state and the final state.
3 Reverse the directions of arrows on all edges.
4. Construct the regular grammar from this F.A.

(1)

① convert the given right linear grammar into equivalent left linear grammar.

Sol<sup></sup>

$S \rightarrow bB$

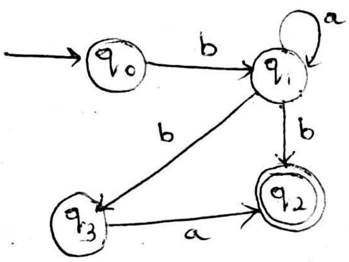
$B \rightarrow bC$

$B \rightarrow aB$

$C \rightarrow a$

$B \rightarrow b.$

The equivalent FA can be



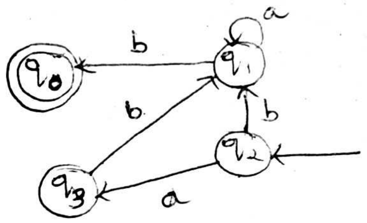
Assume $S = q_0$

$B = q_1$

$C = q_3$

$\varepsilon = q_2$

Now we will change the final states to initial states and initial state to final state.



The equivalent grammar can be

$q_2 \rightarrow q_1 b$

$q_2 \rightarrow q_0 a$

$q_0 \rightarrow q_1 b$

$q_1 \rightarrow q_1 a \mid q_3 b.$

$q_3 \rightarrow \varepsilon.$

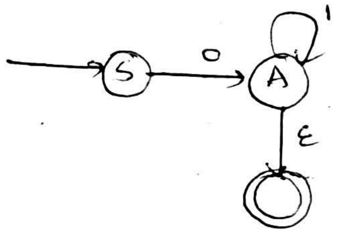
② convert the following right linear grammar to left linear,

$S \rightarrow 0A$

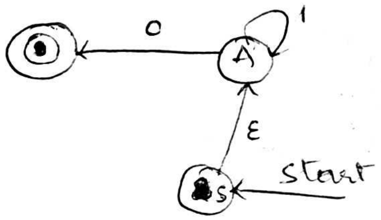
$A \rightarrow 1A$

$A \rightarrow \varepsilon$.

Sol<sup></sup>ⁿ We will first construct FA from given grammar.



Now interchange initial and final state and reverse the edges of FA.



$S \rightarrow A\varepsilon$ i,e $S \rightarrow A$

$A \rightarrow A1$

$A \rightarrow 0$

is required left linear grammar

Consider a string 01111, we will derive this string using right linear grammar as,

$S \rightarrow 0A$

$S \rightarrow 01A$ $(\because A \rightarrow 1A)$

$S \rightarrow 011A$ $(\because A \rightarrow 1A)$

$S \rightarrow 0111A$ $(\because A \rightarrow 1A)$

$S \rightarrow 01111A$ $(\because A \rightarrow 1A)$

$S \rightarrow 01111$ $(\because A \rightarrow \varepsilon)$

$|||^{rly}$ the left linear grammar can be used to derive a string 01111

$S \longrightarrow A$

$S \longrightarrow A1 \quad (\because A \longrightarrow A1)$

$S \longrightarrow A11 \quad (\because A \longrightarrow A1)$

$S \longrightarrow A111 \quad (\because A \longrightarrow A1)$

$S \longrightarrow A1111 \quad (\because A \longrightarrow A1)$

$S \longrightarrow 01111 \quad (\because A \longrightarrow 0)$

(3) Give an equivalent left linear grammar for the following right linear grammar

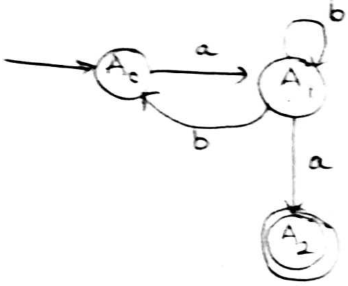
$A_0 \rightarrow aA_1$

$A_1 \rightarrow bA_1$

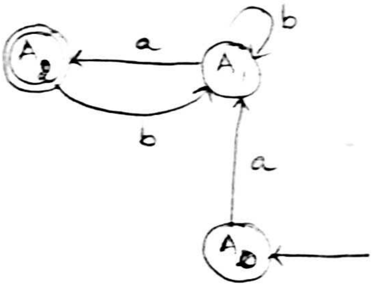
$A_1 \rightarrow a$

$A_1 \rightarrow bA_0$

Sol<sup></sup>n we will first construct FA for given right linear grammar.



Now we will reverse the edges of FA and interchange initial and final states.



And now let us write left linear grammar as -

$A_0 \rightarrow A_1 a$

$A_1 \rightarrow A_1 b$

$A_1 \rightarrow A_2 a$ *

$A_2 \rightarrow A_1 b$

$A_1 \rightarrow a$

consider a string abbaa

Right linear grammar -

$A_0$

$aA_1$ $(\because A_0 \rightarrow aA_1)$

$abA_1$ $(\because A_1 \rightarrow bA_1)$

$abbA_0$ $(\because A_1 \rightarrow bA_0)$

$abbaA_1$ $(\because A_0 \rightarrow aA_1)$

$abbaa$ $(\because A_1 \rightarrow a)$

Left linear grammar

$A_0$

$A_1 a$ $(\because A_0 \rightarrow A_1 a)$

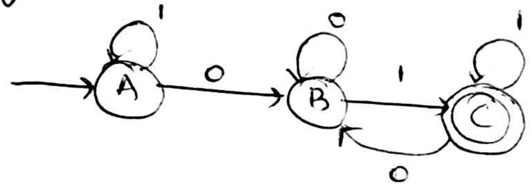
$A_2 aa$ $(\because A_1 \rightarrow A_2 a)$

$A_1 baa$ $(\because A_2 \rightarrow A_1 b)$

$A_1 bbaa$ $(\because A_1 \rightarrow A_1 b)$

$abbaa$ $(\because A_1 \rightarrow a)$

(4) obtain a left linear grammar for the DFA as shown in fig:



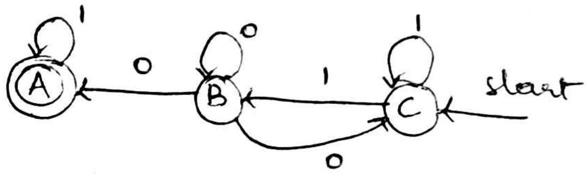
Sol

Soln We will follow following steps to obtain LLG

1) Interchange the initial and final state.

2) Reverse the directions of all the edges.

The DFA will be



The left linear grammar will be -

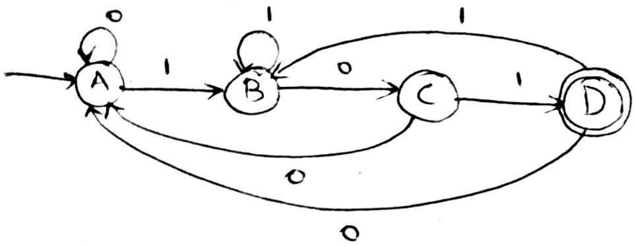
$C \rightarrow C1 \mid B1$

$B \rightarrow B0 \mid A0 \mid C0 \mid 0$

$A \rightarrow A1 \mid 1.$

The start symbol is c.

⑤ obtain a ~~right~~ left linear grammar for the following FA as shown in fig.



Solⁿ The right linear grammar will be

$A \longrightarrow 0A \mid 1B$

$B \longrightarrow 1B \mid 0C$

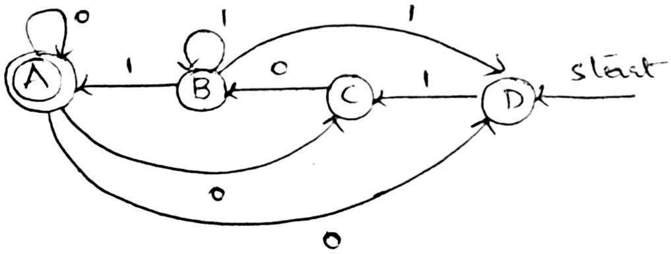
$C \longrightarrow 0A \mid 1D \mid 1$

$D \longrightarrow 0A \mid 1B.$

b) The left linear grammar can be constructed by following the steps

1) Construct DFA by changing initial and final states.
2) Reverse the directions of all the edges.

The DFA will be -



The LLG will be

$D \longrightarrow C1$

$C \longrightarrow B0$

$B \longrightarrow B1 \mid A1 \mid D1 \mid 1$

$A \longrightarrow A0 \mid C0 \mid D0 \mid 0.$

D is start symbol.

consider the string 101101

| Derivation by LLG. | Derivation by RLG. |
|---|---|
| D | A |
| C1 | 1B |
| B01 | 10C |
| D101 | 101D |
| C1101 | 1011B |
| B01101 | 10110C |
| 101101 | 101101. |

CONVERSION OF LEFT LINEAR GRAMMAR TO RIGHT LINEAR GRAMMAR

Method :

(1) convert the given left linear grammar to FA

(2) Interchange the initial and final states of FA

(3) Reverse the directions of all the arrows on all edges.

(4) Construct the regular grammar from this F.A.

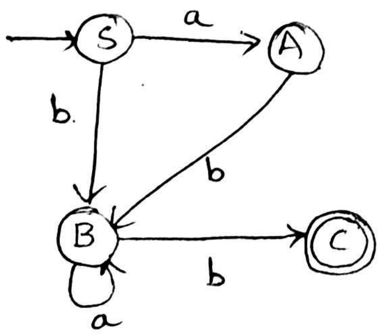
(1) Convert the following left linear grammar to right linear.

$S \rightarrow Aa \mid Bb$

$A \rightarrow Bb$

$B \rightarrow Ba \mid b$

Soln We will first design FA for given LLG



S is a start symbol.

For eg $P = \{ S \rightarrow S+S$

$S \rightarrow S * S$

$S \rightarrow (S)$

$S \rightarrow 4 \}$.

If the language is 4+4 * 4 then we can use the production rules given by P.

We use the following conventions.

1. The capital letters are used to denote nonterminals
2. The lower case letters are used to denote terminals.

## DERIVATION & LANGUAGES:

The production rules are used to derive certain strings.

We will now formally define the language generated by grammar $G = (V, T, P, S)$

The generation of language using specific rules is called derivation.

PROBLEMS ON CONTEXT FREE GRAMMAR :

① construct the CFG for the language having any number of a's over the set $\Sigma = \{a\}$.

Sol

Soln As we know the regular expression for above mentioned language is

r.e $= a^*$

Let us build the production rules for the same.

$S \rightarrow aS$ rule 1

$S \rightarrow \varepsilon$ rule 2

Now if we want "aaaa" string to be derived we can start with start symbols.

S
aS ∵ rule 1
aaS rule 1
aaaS rule 1
aaaaS rule 1
aaaaaS rule 1
aaaaa ε rule 2
= aaaaa

The r.e $= a^*$ suggest a set of $\{\varepsilon, a, aa, aaa \ldots\}$

② Try to recognize the language L for given CFG.

$G = [\{S\}, \{a,b\}, P, \{S\}]$

Where $P = \left\{ \begin{array}{l} S \rightarrow aSb \\ S \rightarrow ab \end{array} \right\}$

Soln Since $S \rightarrow aSb \mid ab$ is a rule.

'|' indicates 'or' operator.

$S \rightarrow aSb.$

If, this rule can be recursively applied then,

$S$

$aSb$

$\downarrow$

$aaSbb$

$\downarrow$

$aaaSbbb.$

finally we can put $S \rightarrow ab$

then it becomes. aaaa bbbb.

Thus we can have any no of as first and then equal number of b's following it.

Hence we can guess the language as

$\{L = a^n b^n \text{ where } n \geq 1\}$

③ Construct CFG for the r.e $(0+1)^*$

Soln The CFG can be given by

$$P \rightarrow \{ S \rightarrow 0S \mid 1S \\ S \rightarrow \varepsilon \}$$

The rules are in combination of 0's and 1's with the start symbol.

Since $(0+1)^*$ indicates $\{\varepsilon, 0, 1, 01, 10, 00, 11 \ldots\}$ in this set $\varepsilon$ is a string. So in the rules we can set the rule $S \rightarrow \varepsilon$.

④ Construct a grammar for the language containing strings of atleast two a's.

Soln Let $G = (V, T, P, S)$ where

$V = \{S, A\}$

$T = \{a, b\}$

$P = \{ S \rightarrow AaAaA$

$A \rightarrow aA \mid bA \mid \varepsilon \}$

The rule $S \rightarrow AaAaA$ is something in which two a's are maintained since atleast two a's should be there in the strings.

And $A \rightarrow aA \mid bA \mid \varepsilon$ gives any combination of a's and b's be this rules gives the strings of $(a+b)^*$.

Thus the logic for this example will be

(anything) a (anything) a (anything)

so before a or after a there could be any combination of a's and b's.

⑤ construct a grammar generating

$L = wcw^T$ where $w \in \{a, b\}^*$

sol^n The strings which can be generated for given L is

$L = \{ aacaa, bcb, abcba, bacab \ldots \}$

The grammar could be

$S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow c$

The string generated from given production rules as.

$S$

$\Rightarrow aSa$

$\Rightarrow abSba$

$\Rightarrow abcba$

⑥ construct CFG for the language L which has all the strings which are all palindrome over $\Sigma = \{a, b\}$

sol^n As $\varepsilon$ can be the palindrome

a can be palindrome,
b can be palindrome.

So we can write the production rules as,

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

P can be . $S \to aSa$
$S \to bSb$
$S \to a$
$S \to b$
$S \to \varepsilon$

The string abaaba can be derived as.

$S$
$\Rightarrow aSa$
$\Rightarrow abSba$
$\Rightarrow abaSaba$
$\Rightarrow aba\varepsilon aba$
$\Rightarrow abaaba$ which is a palindrome.

⑦ Construct CFG which consists of all strings having atleast one occurence of 000.

Soln r.e = $(0+1)^* 000 (0+1)^*$

The prodn rules are

$S \to ATA$
$A \to 0A \mid 1A \mid \varepsilon$
$T \to 000$.

⑧ Construct CFG for the language in which there are no consecutive b's, the strings may or may not have consecutive a's.

Soln $S \to aS \mid bA \mid a \mid b \mid \varepsilon$
$A \to aS \mid a \mid \varepsilon$.

(9) Recognize the context free language for the given CFG.

$S \rightarrow aB \mid bA$

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

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

Soln To find the language denoted by given CFG, we try to derive the rules and get various strings.

Then after observing those strings we come to know, which language it is denoting.

Let us rewrite all those rules and number them

| | |
|---|---|
| $S \rightarrow aB$ | rule 1 |
| $S \rightarrow bA$ | rule 2 |
| $A \rightarrow a$ | rule 3 |
| $A \rightarrow aS$ | rule 4 |
| $A \rightarrow bAA$ | rule 5 |
| $B \rightarrow b$ | rule 6 |
| $B \rightarrow bS$ | rule 7 |
| $B \rightarrow aBB$ | rule 8. |

Let us apply the rules randomly starting with start symbol.

| | |
|---|---|
| S | |
| aB | rule 1 |
| aaBB | rule 8 |
| aabSB | rule 7 |
| aabbAB | rule 2 |
| aabbaB | rule 3 |
| aabbabS | rule 7 |
| aabbabbA | rule 2 |
| aabbabba | rule 3. |

We have got some string as 'aabbabba".

Let us try out something else.

S

bA rule 2

bbAA rule 5

bbaSA rule 4

bbaaBA rule 1

bbaabA rule 6

bbaaba rule 3.

Now we got "bbaaba" string.

Conclusion:- L containing all strings having equal no of a's and b's,

(10) construct CFG for the language containing atleast one occurence of double a.

Sol) We can write CFG for double a as

$A \rightarrow aa$.

other rule for any no of a's and b's. in any combination

ie $B \rightarrow aB \mid bB \mid \varepsilon$. which is eqvt to $(a+b)^*$.

$S \rightarrow BAB$ ⇒ (anything) ( one occurence of double a ) (anything)

$A \rightarrow aa$

$B \rightarrow aB \mid bB \mid \varepsilon$.

(11) construct CFG for the language containing all the strings of diff^t first and last symbols over $\Sigma = \{0,1\}$

Soln If string starts with 1 it should end with 0
If string starts with 0 it should end with 1.

$S \rightarrow 0A1 \mid 1A0$

$A \rightarrow 0A \mid 1A$

Given CFG is eqlt to the reg Expr^n $[0(0+1)^* 1 + 1(0+1)^* 0]$.

(12) construct CFG for the language $L = a^n b^{2n}$ where $n \geq 1$

Sol^n $S \to aSbb \mid abb$.

(13) obtain a CFG to generate unequal number of a's and b's.

Sol^n for unequal number of a's and b's there are follong cases.

(1) only a's are present and number of b's are zero.
(2) " b's " " " " " a's " "
(3) Number of a's are atleast one more than number of b's.
(4) " " b's " " " " " " " a's.

$S \to A \mid B$

$A \to CaA \mid CaC$ → produces any no. of a's + equal no. of a's and b's.

$B \to CbB \mid CbC$ → more b's than a's.

$C \to aCbC \mid bCaC \mid \varepsilon$ → produces equal no of a's and b's.

Simulation: consider the string abbaaba

$S \; (S \to A)$
$A \; (A \to CaC)$
$CaC \; (C \to \varepsilon)$
$Ca\varepsilon. \; (C \to aCb\varepsilon)$
$aCbCa \; (C \to \varepsilon)$
$abCa \; (C \to bCaC)$
$abbCaCa \; (C \to \varepsilon)$
$abbaCa \; (C \to aCbC)$
$abbaaCbCa \; (C \to \varepsilon)$
$abbaabCa \; (C \to \varepsilon)$
$abbaaba.$

(14) obtain a CFG to obtain balanced set of parenthesis (i.e every left parenthesis should match with the corresponding right parenthesis)

Soln consider T = { (, [, ), ] } as a set of brackets.

P is a set of production rules which are as given below:

$S \rightarrow SS$

$S \rightarrow ()$

$S \rightarrow []$

$S \rightarrow (S)$

$S \rightarrow [S]$

Simulation: consider the string (()[])

Derivation

| | |
|---|---|
| S | ( S → (S) ) |
| (S) | ( S → SS ) |
| (SS) | ( S → () ) |
| (()S) | ( S → [] ) |
| (()[]) | |

(15) obtain CFG to generate the set of all strings over alphabet {a, {a, b} with exactly twice as many a's as b's.

Soln CFG – no of a's are twice the no of b's.

possibilities – $\boxed{a}\,b\,\boxed{a}$ or aab or baa

T = {a, b}

S = abaS | aabS | baaS | ε

Simulation: abaS
ababaaS
ababaaabaS $\underset{\varepsilon}{}$ ⇒ ababaaaba.

(16) Define the context free grammars in the triple form. (V, T, P, S) for the given languages on Σ (a,b).

(i) All strings having atleast two a's

(ii) All possible strings not containing trepple b's.

Soln

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

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

$V = \{S, T\}$

$T = \{a, b\}$

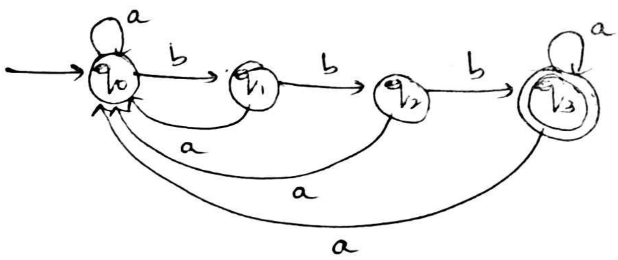
$P = \{ S \rightarrow TaTaT$

$T \rightarrow aT \mid bT \mid \varepsilon \}$

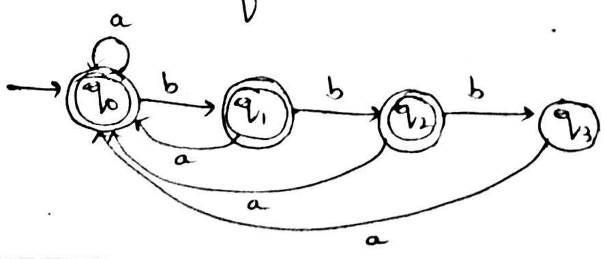
| | | |
|---|---|---|
| S | S | S |
| TaTaT | TaTaT | TaTaT |
| aa | aTaTabT | bTabTabT |
| | aaTaab. | babTabT |
| | aaaab. | bababT |
| | | babab |

..... so on.

(ii) We will first design FA accepting strings with triple b's.



Now, FA that do not accept strings with triple b's. We will change non final states to final and final state to non final.



$A \rightarrow aA \mid bB \mid a \mid b.$

$B \rightarrow aA \mid bC \mid a \mid b.$

$C \rightarrow aA \mid bD \mid a$

$D \rightarrow aA \mid a$

(17) Find of the given grammar is finite or infinite $S \rightarrow AB$

$A \rightarrow BC / a, \; B \rightarrow CC / b, \; C \rightarrow a.$

Solⁿ

| | | |
|---|---|---|
| $S$ | $S$ | $S$ |
| $AB \; (S \rightarrow AB)$ | $AB \; (S \rightarrow AB)$ | $AB$ |
| $aB \; (A \rightarrow a)$ | $BCB \; (A \rightarrow BC)$ | $BCB$ |
| $ab \; (B \rightarrow b)$ | $CCCB \; (B \rightarrow CC)$ | $bab \; (\because B \rightarrow b, \; C \rightarrow a).$ |
| | $aaab \; (\because C \rightarrow a, \; B \rightarrow b)$ | |

Thus the strings (ab, aaab, bab, aaa ---) are generated by above grammar.

Hence given grammar is finite.

(18) Find DFA and CFG for the following language.

L = {odd-length strings in $(a,b)^*$ with middle symbol a}.

Solⁿ $\boxed{a \text{ or } b} \; a \; \boxed{a \text{ or } b}$

Thus.

$$r.e = (a+b)^* \underset{\substack{\uparrow \\ \text{middle.}}}{a} (a+b)^*$$

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

$V = \{S\}$

$T = \{a, b\}$

$P = \{ S \rightarrow aSa \mid bSb \mid aSb \mid bSa \mid a \}.$

(19) For each of the following languages over $\Sigma = \{0,1\}$ write context free grammar with the minimal number of variables that generates the language.

(i) $\{w \mid w = w^R\}$ $\{w^R$ denotes the reverse of $w)$

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

$= (S, \{0,1\}, P, S)$

$P = S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon.$

(ii) $\{w \mid w \neq w^R\}$

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

$= (\{S\}, \{0,1\}, P, S)$

$P = S \rightarrow 0S1 \mid 1S0 \mid 0 \mid 1 \mid \varepsilon.$

(20) Find the language generated by the following CFG.

$S \rightarrow aSbScS \mid aScbS \mid bSaScS \mid bScSa \mid cSaSbS \mid cSbSaS \mid \wedge$

possible strings are.

$\{\wedge$, abc, bac, cba, aabcbabccabc, aacbbc, ----$\}$.

If we observe, the strings containing equal number of alphabets in any order.

(21) Construct a G so that $L(G) = \{a^n b a^m \mid m, n >= 1\}$

soln r.e = $a^* b a^*$

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

$G = ((S, A), (a, b), P, S)$.

Prodn rules will be.

$S \rightarrow AbA$

$A \rightarrow aA \mid a$

(22) If G is $S \rightarrow aS \mid a$ then show that $L(G) = \{a\}^{+}$.

Solⁿ Let,

$S \rightarrow aS$

$S \rightarrow a$

The strings generated by this prodⁿ rules are

| | | | | |
|---|---|---|---|---|
| S | S | S | S | |
| a | aS | aS | aS | |
| | aa | aaS | aaS | .... so on. |
| | | aaa | aaaS | |
| | | | aaaa | |

Thus we get $\{a, aa, aaa, aaaa ----\}$

This is the language containing any no of as except null string.

∴ r.e $= \{a\}^{+}$.

(23) If $G = (\{S\}, \{0,1\}, \{S \rightarrow 0S1,\ S \rightarrow \varepsilon\}, S)$, find $L(G)$

Let prodⁿ rules be.

$S \rightarrow 0S1$

$S \rightarrow \varepsilon$.

The strings generated are $\{\varepsilon, 01, 0011, 000111 ---\}$

This grammar denotes the language

$L(G) = \{0^{n}1^{n} \mid n \geq 0\}$.

(24) If $G = (\{S\}, \{a\}, \{S \rightarrow SS\}, S)$ find the language generated by G.

Solⁿ prodⁿ rules are given as $S \rightarrow SS$

but there is no prodⁿ rule for terminating character.

If we assume $S \rightarrow SS \mid a$

then the given CFG generates

$L(G) = \{a^{n} \mid n \geq 1\}$.

# DERIVATION TREES :

Derivation trees is a graphical representation for the derivation of the given prodⁿ rules for a given CFG. It is the simple way to show how the derivation can be done to obtain some string from given set of prodⁿ rules.

- The derivation tree is also called parse tree.

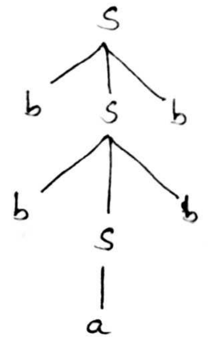
Following are properties of any derivation tree.

1. The root node is always a node indicating start symbol.
2. The derivation is read from left to right
3. The leaf nodes are always terminal nodes.
4. The interior nodes are always non terminal nodes.

For eg:-

$S \rightarrow bSb|a|b$ is a prodⁿ rule.

The S is a start symbol.

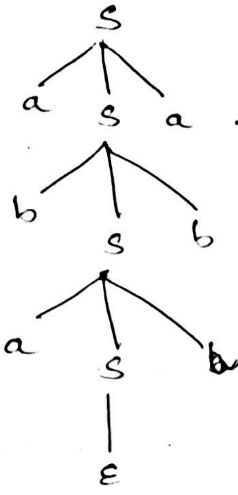


The above tree is a derivation tree drawn for deriving a string bbabb. By simply reading the leaf nodes we can obtain the desired string.

PROBLEMS ON DERIVATION TREE :

① Draw a derivation tree for the string aba$\overset{a}{\cancel{b}}$ba for the CFG given by

G where P = { S → aSa
S → bSb
S → a | b | ε }.



- we have chosen → aSa

– we have chosen S → bSb

– we have chosen S → aSa

S → ε is chosen.

② Construct the derivation tree for the string aabbabba from the CFG given by

S → aB | bA
A → a | aS | bAA
B → b | bS | aBB

Soln To draw a tree we will first try to obtain derivation for the string aabbabba.

S

$\boxed{aB}$  S → aB                . aabbabba ( A → a).

a $\boxed{aBB}$  B → aBB

aa$\underline{bS}$B  B → bS

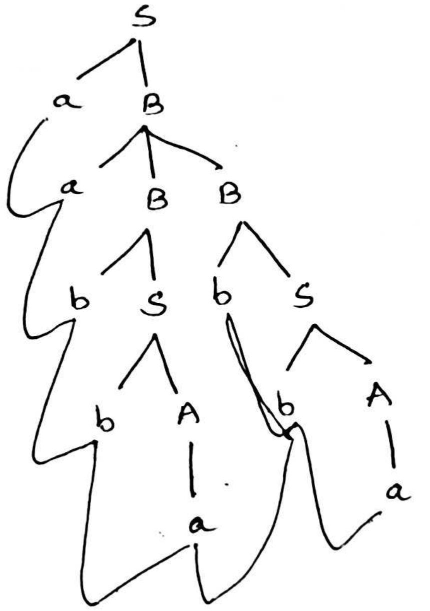
aab~~b~~b$\underline{A}$B  S → bA

aabb~~b~~aB  A → a

aabb~~b~~abS  B → bS

aabbabbA  S → bA

Let us draw a tree



aabbabba

LEFT MOST DERIVATION AND RIGHTMOST DERIVATION.

Left most Derivation: If a word w is generated by a CFG by a certain derivation, and at each step in the derivation a rule of production is applied to the leftmost non terminal in the working string, then this derivation is called a leftmost derivation.

For eg:- $S \to aSA \mid b$

$A \to Ab \mid a$

The following is a leftmost derivation.

$S \to a\underline{S}A$

$\to aa\underline{S}AA$

$\to aab\underline{A}A$

$\to aab\underline{A}bA$

$\to aabab\underline{A}$

$\to aababa$

At every stage in the derivation, the non terminal replaced is the leftmost one.

**Rightmost Derivation:** If a word w is generated by a CFG by a certain derivation and at each step in the derivation a rule of production is applied to the right most non terminal in the working string, then this derivation is called a right most derivation.

For eg:- Consider the CFG,

$S \rightarrow aSA \mid b.$

$A \rightarrow Ab \mid a.$

The following is a right most derivation.

$S \rightarrow aS\underline{A}$

$\rightarrow aS\underline{A}b.$

$\rightarrow a\underline{S}ab.$

$\rightarrow aaS\underline{A}ab.$

$\rightarrow aa\underline{S}aab$

$\rightarrow aabaab.$

At every stage in the derivation the non terminal replaced is the right most one.

**Problems:**

① Consider the grammar with productions

$S \rightarrow aAB$

$A \rightarrow bBb$

$B \rightarrow A \mid \varepsilon.$

Sol$^n$: Leftmost derivation of the string 'abbbb'.

$S \rightarrow a\underline{A}B$

$\rightarrow ab\underline{B}bB \ (\because A \rightarrow bBb).$

$\rightarrow ab\underline{A}bB \ (\because B \rightarrow A)$

$\rightarrow abb\underline{B}bbB \ (\because A \rightarrow bBb)$

→ abbεbbB (∵ B → ε)

→ abbbbB

→ abbbbε (∵ B → ε)

→ abbbb.//

Right most derivation of the string 'abbbb'

S → aAB

→ aAε (∵ B → ε)

→ abBb (∵ A → bBb)

→ abAb (∵ B → A)

→ abbBbb (∵ A → bBb)

→ abbεbb (∵ B → ε)

→ abbbb.

(2) Let G be the grammar S → aB | bA
A → a | aS | bAA
B → b | bS | aBB.

for the string aabbabab, find:

(i) Derivation tree

(ii) Rightmost derivation

(iii) Leftmost derivation

Sol<sup>n</sup> The given grammar G with productions are,

S → aB | bA ; A → a | aS | bAA ; B → b | bS | aBB.

Given string is aabbabab

Left most : S

→ aB (∵ S → aB)

→ aaBB (∵ B → aBB)

→ aabB (∵ B → b)

→ aabbS (∵ B → bS)

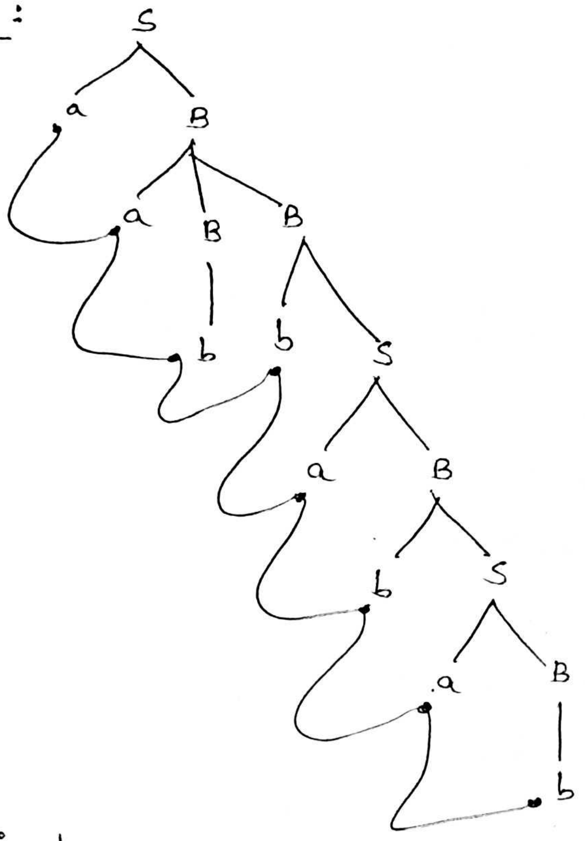
→ aabbaB (∵ S → aB)

→ aabbabS (∵ B → bS)

→ aabbabaB (∵ S → aB)

→ aabbabab (∵ B → b)

Derivation tree :



Rightmost derivation :

$S \longrightarrow a\underline{B}$

$\longrightarrow aaB\underline{B} \; (\because B \rightarrow aBB)$

$\longrightarrow aa\underline{B}b \; (\because B \rightarrow b)$

$\longrightarrow aab\underline{S}b \; (\because B \rightarrow bS)$

$\longrightarrow aabb\underline{A}b \; (\because S \rightarrow bA)$

$\longrightarrow aabba\underline{S}b \; (\because A \rightarrow aS)$

$\longrightarrow aabbab\underline{A}b \; (\because S \rightarrow bA)$

$\longrightarrow aabbabab \; (\because A \rightarrow a).$

③ Let G be a grammar $S \to 0B \mid 1A$, $A \to 0 \mid 0S \mid 1AA$, $B \to 1 \mid 1S \mid 0BB$ for the string 00110101 find.

(i) LMD
(ii) RMD
(iii) Derivation tree

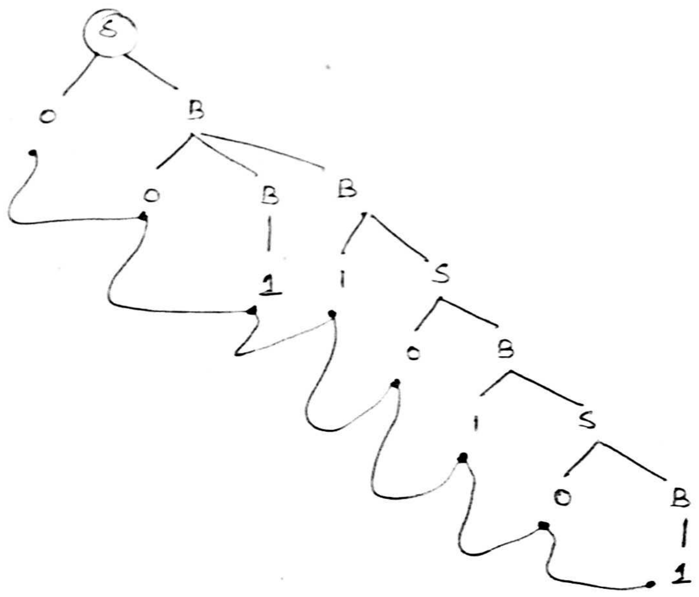
Soln: Leftmost Derivation

$S \to 0\underline{B}$
$\to 00\underline{B}B$ $(B \to 0BB)$
$\to 001\underline{B}$ $(B \to 1)$
$\to 0011\underline{S}$ $(B \to 1S)$
$\to 00110\underline{B}$ $(S \to 0B)$
$\to 001101\underline{S}$ $(B \to 1S)$
$\to 0011010B$ $(S \to 0B)$
$\to 00110101$ $(B \to 1)$

Rightmost Derivation:

$S \to 0\underline{B}$
$\to 00B\underline{B}$ $(B \to 0BB)$
$\to 00B1S$ $(B \to 1S)$
$\to 00B10\underline{B}$ $(S \to 0B)$
$\to 00B101\underline{S}$ $(B \to 1S)$
$\to 00B1010B$ $(S \to 0B)$
$\to 00\underline{B}10101$ $(B \to 1)$
$\to 00110101$ $(B \to 1)$

(iii)



(4) Let a grammar

$S \rightarrow aB \mid bA$
$A \rightarrow a \mid aS \mid bAA$
$B \rightarrow b \mid bS \mid aBB$. for the string aaabbabbba find,

(i) LMD (ii) RMD (iii) Parse Tree.

Soln Given string is aaabbabbba.

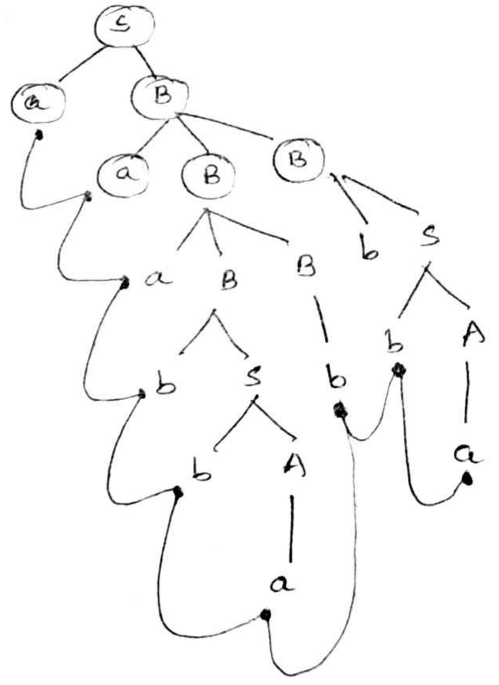
(i) Leftmost Derivation.

$S \rightarrow a\underline{B}$
$\rightarrow aa\underline{B}B \quad (B \rightarrow aBB)$
$\rightarrow aa\underline{a\dot{B}B}B \quad (B \rightarrow bS)$
$\rightarrow aaa\underline{b\dot{S}}BB \quad (S \rightarrow bA)$
$\rightarrow aaabb\underline{A}BB \quad (A \rightarrow a)$
$\rightarrow aaabb\underline{a}\underline{B}B \quad (B \rightarrow b)$
$\rightarrow aaabba\underline{b}\underline{B} \quad (B \rightarrow bS)$
$\rightarrow aaabbabb\underline{S} \quad (S \rightarrow bA)$
$\rightarrow aaabbabbbA \quad (A \rightarrow a)$
$\rightarrow aaabbabbba.$

(ii) Rightmost Derivation

$S \rightarrow a\underline{B}$
$\rightarrow aaB\underline{B} \quad (B \rightarrow aBB)$
$\rightarrow aaBbS \quad (B \rightarrow bS)$
$\rightarrow aaBbb\underline{A} \quad (S \rightarrow bA)$
$\rightarrow aaBbba \quad (A \rightarrow a)$
$\rightarrow aa\underline{B}bba \quad (B \rightarrow aBB)$
$\rightarrow aaaB\underline{B}bba \quad (B \rightarrow b)$
$\rightarrow aaa\underline{B}bbba \quad (B \rightarrow bS)$
$\rightarrow aaabSbbba \quad (S \rightarrow bA)$
$\rightarrow aaabb\underline{A}bbba \quad (A \rightarrow a)$
$\rightarrow aaabbabbba$ //

(iii) Parse Tree :



# UNIT - 3
# CONTEXT FREE GRAMMARS

## Ambiguity in context Free Grammar:

- The grammar can be derived in either leftmost derivation or right most derivation.
- One can draw a derivation tree called as parse tree or syntax tree based on these derivations.
- The parse tree has to be unique even though the derivation is leftmost or right most.
- If there exists more than one parse tree for a given grammar that means there could be more than one leftmost or right most derivation possible and then that grammar is said to be ambiguous grammar.

For eg:- The CFG given by G = {V, T, P, S}

where V = {E}

T = {id}

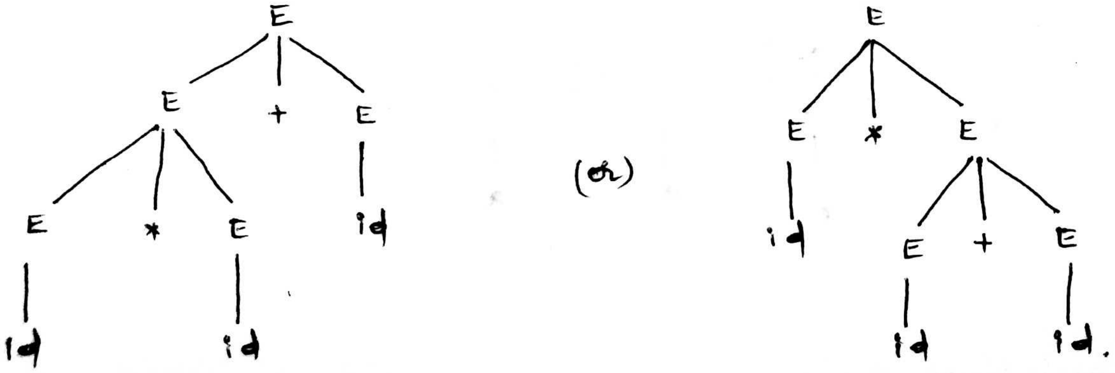
P = { E → E+E

E → E*E

E → id }

S = {E}

Now if the string is id * id + id then we can draw two different parse trees indicating our id * id + id.



Thus the above grammar is an ambiguous grammar.

PROBLEMS ON AMBIGUITY OF GRAMMAR:

① check whether the given grammar is ambiguous or not.

$S \rightarrow iCtS$

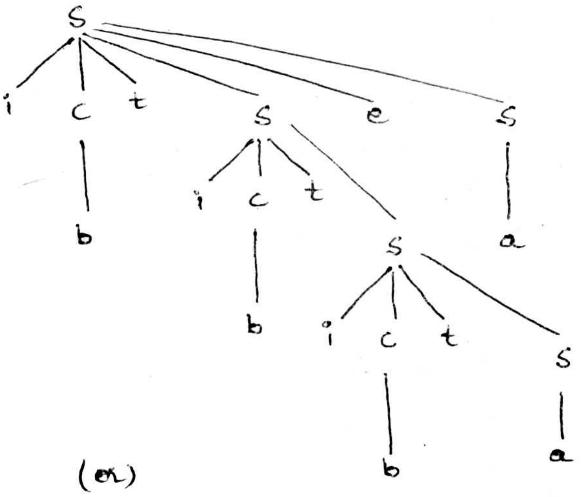
$S \rightarrow iCtSeS$

$S \rightarrow a$

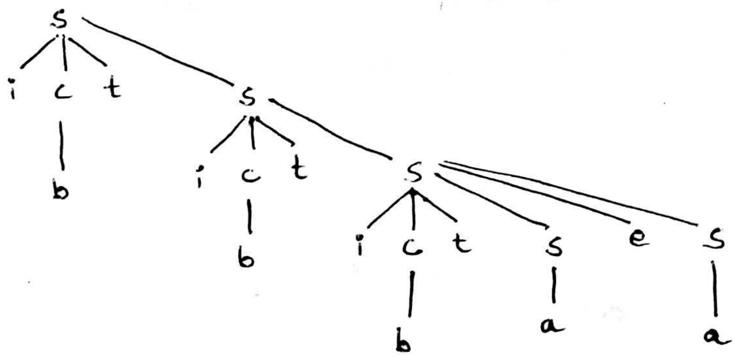
$C \rightarrow b.$

Sol<sup>n</sup> To check whether given grammar is ambiguous or not We will have some string for derivation tree such as ibtibtibtaea

Let us draw the derivation tree



(or)



Thus We have got more than two parse trees.
Hence the given grammar is ambiguous.

② consider the grammar $G = (V, \Sigma, R, S)$

where $V = \{S, A\}$

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

$R = \{ S \rightarrow AA$

$A \rightarrow AAA$.

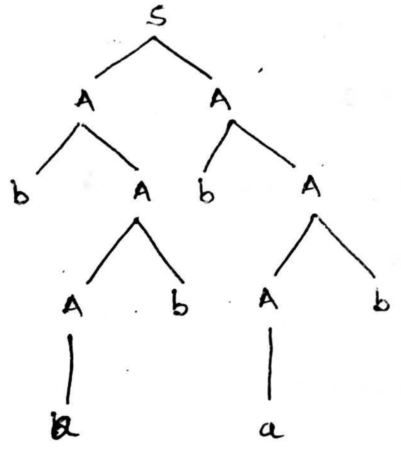
$A \rightarrow a$

$A \rightarrow bA$.

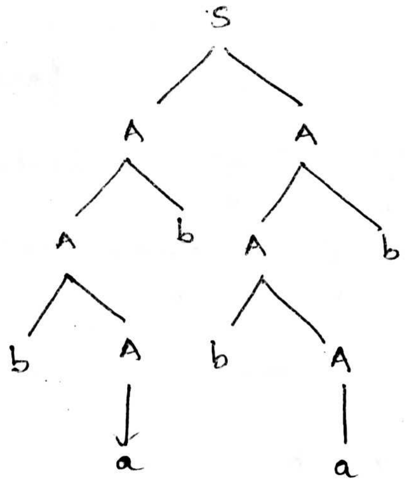
$A \rightarrow Ab \}$

Show that this is an ambiguous grammar.

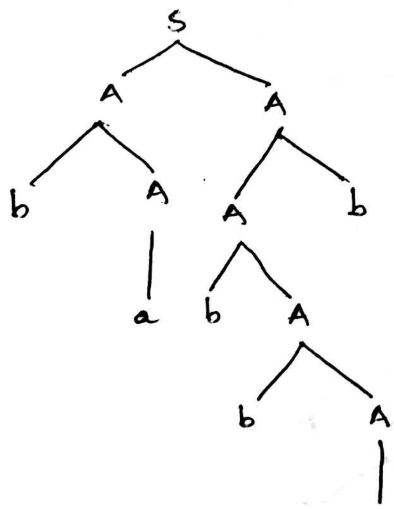
Sol^n consider a string babbab.



ⓐ



ⓑ

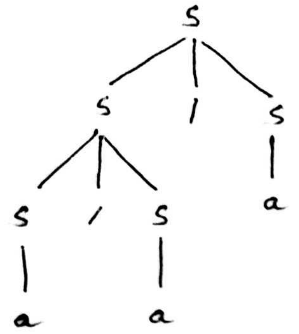


There are more than one parse tree getting generated. Hence the grammar is ambiguous.

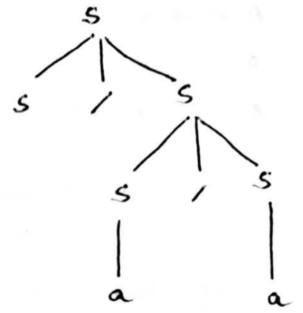
③ show that the grammar $S \rightarrow S/S$ , $S \rightarrow a$ is ambiguous.

Soln In order to prove that given grammar is ambiguous, we will consider the input string a|a|a.

The parse trees that can be drawn to derive this string are



parsetree 1



parse tree 2

- As we get two diff^t strings to derive the same input string we say that the given grammar is ambiguous.

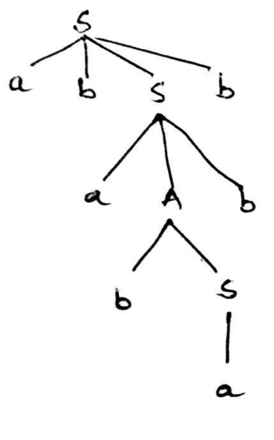
④ show that the grammar G with production

$S \rightarrow a \mid a A b \mid a b S b$

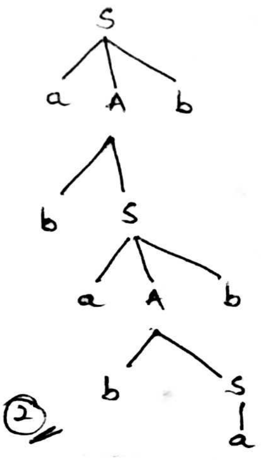
$A \rightarrow a A A b \mid b S$

is ambiguous.

Soln. Consider the valid string abababb



①



②

⑤ Find an unambiguous CFG equivalent to the grammar with productions.

$S \rightarrow aaaaS \mid aaaaaaaS \mid \wedge$

Sol^n To convert an ambiguous grammar into equivalent unambiguous grammar, there is no fixed rule or algorithm.

- We must see that every non-terminal will produce distinct strings so that unique parse tree can be built for given i/p string.

- Possible strings for given grammar are

$\{\wedge, aaaa, aaaaaaa, aaaaaaaa, aaaaaaaaaaa, \ldots\} =$

$\{\wedge, a^4, a^7, a^{11}, a^8, a^{14} \ldots\}$.

The unambiguous grammar will be

$S \longrightarrow aaaaS' \mid B \mid C$

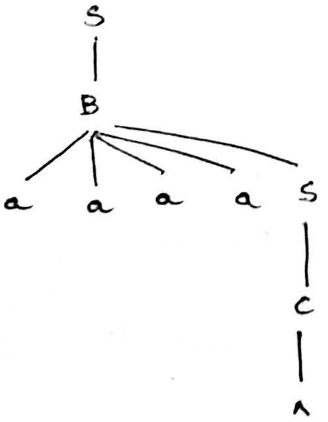
$S' \longrightarrow aaaS$

$B \longrightarrow aaaaS$

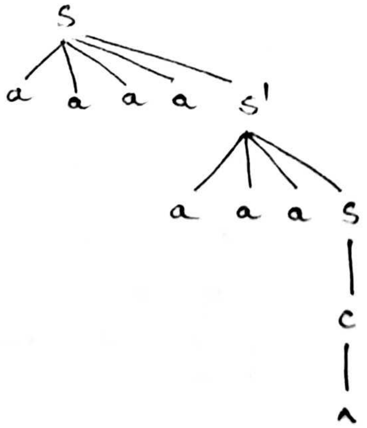
$C \longrightarrow \wedge$

Simulation:

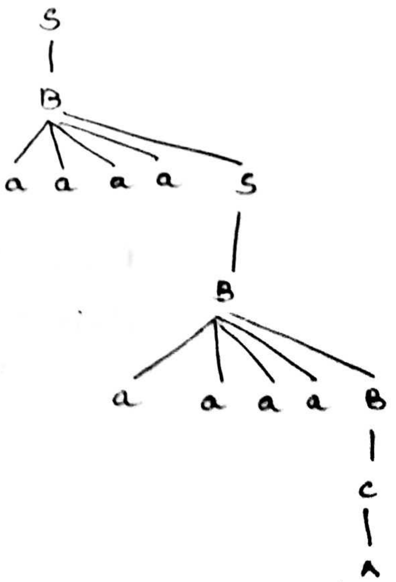
aaaa



aaaaaaa



aaaaaaaa



# MINIMIZATION OF CONTEXT FREE GRAMMAR:

- Various languages can effectively be represented by context free grammar.
- All the grammars are not always optimized.
- That means grammar may consists of some extra symbols (non-terminals).
- Extra symbols unneccasarily increase the length of grammar.
- Simplification of grammar means reduction of grammar by removing useless symbols.

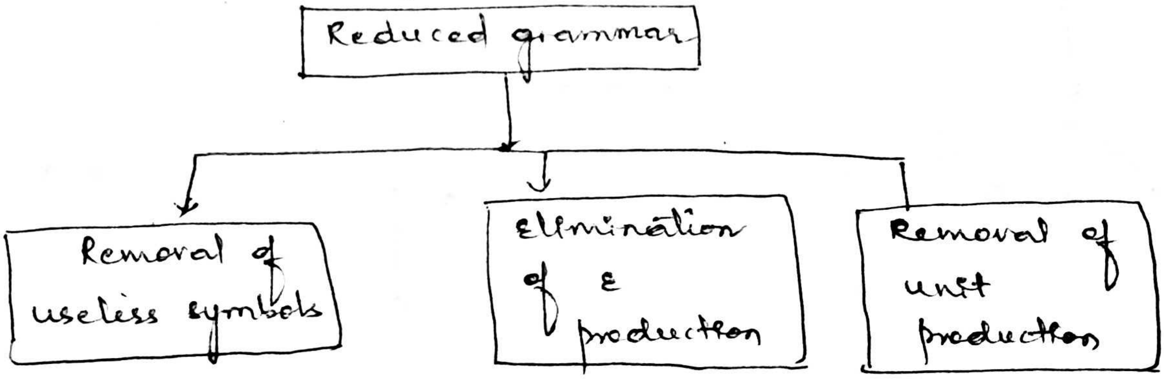
The properties of reduced grammar are

(1) Each variable (i.e non-terminal) and each terminal of G appears in the derivation of some word in L.

(2) There should not be any production as,
X → Y where X and Y are non terminals.

(3). If ε is not in the language L then there need not be the production X → ε.

We see the reduction of grammar as below:


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

## Removal of Useless symbols :

- Any symbol is useful when it appears on the right hand side in the production rule and generates some terminal string. If no such derivation exists then it is supposed to be a useless symbol.

- Any symbol p is useful if there exists some derivation in the following form.

$$S \overset{*}{\Rightarrow} \alpha p \beta$$

and $\alpha p \beta \overset{*}{\Rightarrow} w$

then p is said to be useful symbol.

$\alpha$ and $\beta$ may be some terminal or non-terminal symbols and will help us to derive certain string w in combination with P.

for eg:- $G = (V, T, P, S)$ where $V = \{S, T, X\}$, $T = \{0, 1\}$

$S \rightarrow 0T \mid 1T \mid X \mid 0 \mid 1$ rule 1

$T \rightarrow 00$ rule 2.

Now in the above CFG, the non terminals are S, T and X.

To derive some string we have to start from start symbol S.

| | |
|---|---|
| S | |
| 0T | $S \rightarrow 0T$ |
| 000 | $T \rightarrow 00$ |

Thus we can reach to certain string after following these rules.

But if $S \rightarrow X$ then there is no further rule as a definition to X.

- That means there is no point in the rule $S \rightarrow X$. Hence we can declare that X is a useless symbol.

- We can remove this, so after removal of this useless symbol CFG becomes,

$G = (V, T, P, S)$ where $V = \{S, T\}$

$T = \{0, 1\}$ and $P = \{S \rightarrow 0T \mid 1T \mid 0 \mid 1$

$T \rightarrow 00\}$

S is a start symbol.

PROBLEMS ON REMOVAL OF USELESS SYMBOL:

① Consider the CFG $G = (V, T, P, S)$

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

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

$S \rightarrow 1B \mid 11$

$A \rightarrow 0$

$B \rightarrow BB\}$ for removing useless symbols.

Sol^n Now in the given CFG, if we try to derive any string A gives some terminal symbol as 0 but B does not give any terminal string.

- By following the rules with B we simply get ample number of B's and no significant string.

- Hence we can declare B as useless symbol and can remove the rules associated with it.

Hence after removal of useless symbols we get,

$S \rightarrow 11A \mid 11$

$A \rightarrow 0$

② Find CFG with no useless symbols equivalent to

$S \rightarrow AB \mid CA$ $\quad B \rightarrow BC \mid AB$

$A \rightarrow a$ $\quad C \rightarrow aB \mid b.$

Soln consider the rule,

$S \rightarrow AB$ 1

$S \rightarrow CA$ 2

In the rule 2, C and A can be replaced by some terminating string but in rule 1, B cannot be replaced by terminating string. It tends to form a never ending loop

For Eg:-

$S \rightarrow AB$
$aAB$
$aaAB$
$aaaAB$ and so on.

Thus we come to know as B is a useless symbol.

Removing B the rules are now,

$S \rightarrow CA$

$A \rightarrow a$

$C \rightarrow b.$

③ consider the following CFG

$G = (V, \Sigma, R, S)$

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

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

$R = \{S \rightarrow XY \mid 0$

$X \rightarrow 1\}$

Remove the useless symbols from it.

Sol^n As

$S \rightarrow XY$
$S \rightarrow 0$
$X \rightarrow 1$

It is easy to recognizes that there is no derivation for Y.

Hence we can eliminate the production with Y, and the rules are

$S \rightarrow 0$
$X \rightarrow 1$

(4) Remove useless symbols from the following grammar.

$S \rightarrow aA \mid bB$
$A \rightarrow aA \mid a$
$B \rightarrow bB$
$D \rightarrow ab \mid Ea$
$E \rightarrow aC \mid d$

Sol^n We will find all the prod^ns which are giving terminal symbols.

$A \rightarrow a$
$D \rightarrow ab$
$E \rightarrow d$

Now consider start symbol.

$S \rightarrow aA \mid bB.$

Now since $B \rightarrow bB$ we will be continuously in a loop with B productions.

Thus B is an useless symbol.

So we will eliminate it.

Then the rules are

$S \rightarrow aA$
$A \rightarrow a \mid aA$
$D \rightarrow ab \mid Ea$
$E \rightarrow aC \mid d.$

Again if we look at E productions there is a production E→aC.

But there is no rule for C.

Hence we will remove the production E→aC

Now

$S \longrightarrow aA$

$A \longrightarrow aA \mid a$

$D \longrightarrow ab \mid Ea$

$E \longrightarrow d$

Now the rules S and A indicate that there is no D and E in the derivation.

- Again we get D and E as useless symbols.

So we will eliminate D and E.

Finally after removal of all useless symbols the production rules are,

$S \longrightarrow aA$

$A \longrightarrow aA \mid a.$

(5) Eliminate the useless symbols from the following grammar.

$S \longrightarrow aS \mid A \mid c$

$A \longrightarrow a$

$B \longrightarrow aa$

$C \longrightarrow aCb.$

Soln We will first write all the productions which are giving terminal symbols,

$A \longrightarrow a$

$B \longrightarrow aa.$

But if we look at start symbol S then,

$S \rightarrow aS \mid A \mid c.$

there is no production for B from start symbol.

Thus B becomes a useless symbol.

$|||^{rly}$,

$S \rightarrow C \rightarrow aCb \rightarrow aaCbb \rightarrow aaaCbbb \rightarrow .....$

There is no terminating, symbol for C. Therefore we will eliminate C. Then set of rules are

$S \rightarrow aS \mid A$

$A \rightarrow a$

Thus we obtain the grammar having these two rules after removal of useless symbols B and C.

⑥ Eliminate the useless symbols from following grammar.

$S \rightarrow aA \mid a \mid Bb \mid cC$

$A \rightarrow aB$

$B \rightarrow a \mid Aa$

$C \rightarrow cCD$

$D \rightarrow ddd$

Soln Consider all productions that are giving terminal symbols.

$S \rightarrow a$

$B \rightarrow a$

$D \rightarrow ddd$

Now consider

$S \rightarrow cC$

$C \rightarrow cCD$

$D \rightarrow ddd.$

when we try to derive the string using production rule for C we get,

$S$<br>
$c\underline{C}$<br>
$cc\underline{C}D$<br>
$cccCD$<br>
$cccCddd$

we will not get any terminal symbol for c. Thus we get a useless symbol c.

To reach to D the only rule available is by using C.

If we eliminate C, we have to eliminate D also.

$\therefore \quad S \rightarrow aA \mid a \mid Bb.$

$A \rightarrow aB$

$B \rightarrow a \mid Aa$

is a reduced grammar.

ELIMINATION OF ε PRODUCTIONS FROM GRAMMAR.

Eg:- $S \rightarrow 0S \mid 1S \mid \varepsilon.$

If we replace ε in the production $S \rightarrow 0S$ then it will become $S \rightarrow 0.$

If we replace ε in the production $S \rightarrow 1S$ then it will become $S \rightarrow 1.$

Hence after eliminating ε productions, we can write the rules as.

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

Thus ε production is removed.

PROBLEMS ON REMOVAL OF $\varepsilon$ PRODUCTION

① Remove the $\varepsilon$ production from following CFG by preserving meaning of it.

Soln

$S \rightarrow XYX$

$X \rightarrow 0X \mid \varepsilon$

$Y \rightarrow 1Y \mid \varepsilon$

Now, while removing $\varepsilon$ production we are deleting the rules $X \rightarrow \varepsilon$ and $Y \rightarrow \varepsilon$

To preserve the meaning of CFG we are actually placing $\varepsilon$ at right hand side wherever X and Y have appeared.

$S \rightarrow XYX$

if first X at right hand side is $\varepsilon$ then,

$S \rightarrow YX$

IIIrly if last X in R.H.S $= \varepsilon$

then $S \rightarrow XY$

if $Y = \varepsilon$ then

$S \rightarrow XX$

if Y and X are $\varepsilon$ then

$S \rightarrow X$ also

$S \rightarrow Y$ when both X are replaced by $\varepsilon$.

$\therefore$ $S \rightarrow XY \mid YX \mid XX \mid X \mid Y$

Now let us consider

$X \rightarrow 0X$

if we place $\varepsilon$ at right hand side for X then

$X \rightarrow 0.$

$|||^{rly}$ $Y \rightarrow 1Y|1$

collectively we can rewrite the CFG with removed ε productions as

$S \rightarrow XY|YX|XX|X|Y$

$X \rightarrow 0X|0$

$Y \rightarrow 1Y|1$

(2) For the CFG given below remove the ε production

$S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow \varepsilon.$

Soln According to the replacement procedure, we will place ε at S in the sentential form.

$S \rightarrow aSa$ if $S = \varepsilon$

$S \rightarrow aa$

$|||^{rly}$ if $S \rightarrow bSb$ if $S = \varepsilon$

$S \rightarrow bb$

Thus finally the rules are

$S \rightarrow aSa|bSb|aa|bb.$

(3) Eliminate the ε productions from the CFG given below.

$A \rightarrow 0B1 | 1B1$

$B \rightarrow 0B | 1B | \varepsilon$

Soln We have to eliminate $B \rightarrow \varepsilon$.

$A \rightarrow 0B1$

if $B \rightarrow \varepsilon.$

$A \rightarrow 01$

$|||^{rly}$, $A \rightarrow 1B1$

$A \rightarrow 11$ $(\because B \rightarrow \varepsilon)$

$|||^{ly}$ $B \rightarrow 0B$

If $B = \varepsilon$

$B \rightarrow 0.$

$B \rightarrow 1$

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

Collectively, we can write

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

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

(4) Construct CFG without ε production from the one which is given below

$S \rightarrow a \mid Ab \mid aBa$

$A \rightarrow b \mid \varepsilon$

$B \rightarrow b \mid \varepsilon$

Soln

$A \rightarrow B \rightarrow \varepsilon$

$S \rightarrow Ab$

If $A = \varepsilon$ then

$S \rightarrow b.$

If $B = \varepsilon$

$S \rightarrow aa$

$S \rightarrow a \mid Ab \mid b \mid aa \mid aBa$

$A \rightarrow b$

$B \rightarrow b$

Finally the rules are

$S \rightarrow a \mid Ab \mid b \mid aa \mid aBa$

$A \rightarrow b$

$B \rightarrow b.$

(5) consider the CFG given below.

$S \rightarrow XY$
$X \rightarrow Zb$
$Y \rightarrow bW$
$Z \rightarrow AB$ ✓
$W \rightarrow Z$
$A \rightarrow aA \mid bA \mid \varepsilon$
$B \rightarrow Ba \mid Bb \mid \varepsilon$

Soln In This example only A and B shows ε productions
If we put A = ε for A production then $A \rightarrow a|b$ ; $B \rightarrow a|b$.
Also $Z \rightarrow AB$ putting $A = \varepsilon$ $B = \varepsilon$

$Z \rightarrow \varepsilon\varepsilon \rightarrow \varepsilon$

So wherever Z appears we place ε

$X \rightarrow Zb$
$X \rightarrow \varepsilon \cdot b$
$X \rightarrow b$.

Since ~~was~~ $W \rightarrow Z \rightarrow \varepsilon$ we can straight away delete it.

$Y \rightarrow bW$
$Y \rightarrow b$.

Let us rewrite the rules

$S \rightarrow XY$
$X \rightarrow b$
$Y \rightarrow b$
$A \rightarrow aA \mid bA \mid a \mid b$
$B \rightarrow Ba \mid Bb \mid a \mid b$

Since X is a start symbol and it defines XY only where as X and Y yields b as a terminal symbol. There is no chance for producing rules by A or B Hence A and B are useless symbols.

$S \rightarrow XY$ $Y \rightarrow b$.
$X \rightarrow b$

## REMOVING UNIT PRODUCTIONS :

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

For e.g. if $X \rightarrow Y$
$Y \rightarrow Z$
$Z \rightarrow X$

then X, Y, Z are unit productions.

- To optimize the grammar we need to remove the unit productions.

If $A \overset{*}{\Rightarrow} B$ is a unit production and $B \rightarrow x_1 x_2 x_3 \dots x_n$ then while removing $A \rightarrow B$ production we should add a rule
$A \rightarrow x_1 x_2 x_3 \dots x_n$

## PROBLEMS :

① If the CFG is as below

$S \rightarrow 0A \mid 1B \mid C$
$A \rightarrow 0S \mid 00$
$B \rightarrow 1 \mid A$
$C \rightarrow 01$

then remove the unit productions.

Sol

Sol$^n$ clearly $S \rightarrow C$ is a unit production.

so we can add a rule to S,

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

III$^{rly}$ $B \rightarrow A$ is also a unit production. so we can modify it as $B \rightarrow 1 \mid 0S \mid 00$

Thus finally we can write CFG without unit production as

$S \rightarrow 0A|1B|01$

$A \rightarrow 0S|00$

$B \rightarrow 1|0S|00$

$C \rightarrow 01$.

② optimize the CFG given below by reducing the grammar S is a start symbol.

$S \rightarrow A|0C1$

$A \rightarrow B|01|10$

$C \rightarrow \varepsilon|CD$

sol^n $S \rightarrow A$ ; $A \rightarrow B$ is a unit production.

$C \rightarrow \varepsilon$ is a null production.

$C \rightarrow CD$

C and D are useless symbols

By reducing the grammar we have

$S \rightarrow A$

$A \rightarrow B$ - since B is a useless symbol.

$S \rightarrow 01|10$

i.e $S \rightarrow 01|10|0C1$

But $C \rightarrow \varepsilon$

Hence ultimately $S \rightarrow 01|10$

$A \rightarrow 01|10$

In start symbol there is no A - so it is a useless symbol.

Final CFG $\Rightarrow \boxed{S = 01|10}$

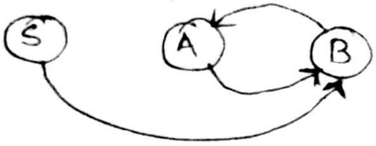
H.W Remove all unit productions from,

$S \rightarrow Aa \mid B$

$B \rightarrow A \mid bb$

$A \rightarrow a \mid bc \mid B.$

Sol^n The dependency graph for the unit production is,



$S \rightarrow A$, $S \rightarrow B$

$B \rightarrow A$, $A \rightarrow B$.

Hence we add to the original non unit productions

$S \rightarrow Aa$ ———— ①

$A \rightarrow a \mid bc$

$B \rightarrow bb$

Now we have unit production from ①

(i) $A \rightarrow B$. so add the B productions ie bb to A.

$A \rightarrow a \mid bb$

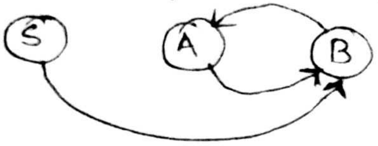
H.W Remove all unit productions from,

$S \to Aa \mid B$

$B \to A \mid bb$

$A \to a \mid bc \mid B$.

Solⁿ The dependency graph for the unit production is,



$S \to A$, $S \to B$

$B \to A$, $A \to B$.

Hence we add to the original non unit productions

$S \to Aa$ ——— ①

$A \to a \mid bc$

$B \to bb$

Now we have unit production from ①

(i) $A \to B$. so add the B productions ie bb to A.

$A \to a \mid bb$

(3) Eliminate the unit productions from following grammar.

S → AB
A → a
B → c|b
C → D
D → E|bc
E → d/Ab.

Soln As S → AB and
A → a

There is only one rule with A and that too giving terminal symbol. Hence there is no question of getting unit production with A. Now consider,

B → C
C → D
D → E

It is clear that B, C, D are unit productions.

As E → d|Ab, we will replace value of D. Then,

D → d|Ab|bc

Similarly as C → D we can write

C → d|Ab|bc

Hence B becomes

B → d|Ab|bc|b

Thus the grammar after removing unit productions

S → AB
A → a
B → d/Ab|bc|b.
C → d|Ab|bc
D → d/Ab|bc
E → d|Ab.

Now there is no path for D and E. From start state we Can remove them by considering useless symbols.

The optimized grammar will be

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow d \mid Ab \mid bC \mid d$

$C \rightarrow d \mid Ab \mid bC$

(4) simplify the grammar

$\{\{S, A, B, C, E\}, \{a, b, c\}, P, S\}$

where P is $S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b$

$B \rightarrow C$

$E \rightarrow c / \wedge$

Soln For simplifying the grammar

1) we must eliminate null productions

2) we must remove unit productions

3) we must remove useless symbols.

The production

$E \rightarrow \wedge / c$ is a useless production

because E cannot be derived from start symbol.

The production

$B \rightarrow C$ is a unit production

There is no rule for deriving C.

Hence we will eliminate this production as well.

The simplified grammar then becomes,

$S \rightarrow AB$
$A \rightarrow a$
$B \rightarrow b.$

(5) Reduce the grammar G given by

$S \rightarrow aAa$
$A \rightarrow Sb \mid bcc \mid DaA$
$C \rightarrow abb \mid DD$ ✓
$E \rightarrow ac$ ✓
$D \rightarrow aDA$ into an equivalent grammar by removing useless symbols and useless productions from it.

Soln The productions for C and E are non reachable from start state.
Hence those are considered to be useless productions
Hence

$C \rightarrow abb \mid DD$
$E \rightarrow ac$

The production rule for D is reachable from start state but it generates infinite string.

$D \rightarrow aDA \rightarrow aaDAA \rightarrow aaaDAAA \rightarrow aaaaDAAAA \rightarrow \text{-----}$

because D is not deriving any terminal symbol.
Hence D is a useless symbol.
Hence we remove the productions for D.
The simplified grammar then becomes.

$S \rightarrow aAa$
$A \rightarrow Sb \mid bcc$

(6) simplify the following grammar

$S \rightarrow AB$.

$A \rightarrow a$

$B \rightarrow C$

$B \rightarrow b$.

$C \rightarrow D$

$D \rightarrow E$

Soln from the given grammar

$B \rightarrow C$

$C \rightarrow D$

$D \rightarrow E$ are unit productions.

moreover E does not derive any terminating symbol.

Hence, $B \rightarrow C$, $C \rightarrow D$, $D \rightarrow E$ are considered to be useless productions.

Hence these rules can be eliminated.

The simplified grammar now becomes,

$S \rightarrow AB$.

$A \rightarrow a$

$B \rightarrow b$.

# UNIT-3

# PDA

## PUSHDOWN AUTOMATA :

- The push down automata is an extension of finite automata, with control of both an i/p tape and a stack. Therefore, a PDA has an i/p tape, a finite control and a stack.
- The i/p tape is divided in many cells. At each cell only one input symbol is placed thus certain input string is placed on tape.
- The finite control has some pointer which point the current symbol which is to be read. At the end of input $ or A(blank) symbol is placed which denotes end of input.
- The stack is such a structure in which you can push and remove the items from one end only.
- In the PDA, we are using the stack for storing the items temporarily.

## DEFINITION OF PDA :-

PDA can be defined as a collection of seven components.

1) $Q$ = The finite set of states
2) $\Sigma$ = i/p alphabet.
3) $\Gamma$ = stack alphabet
4) $q_0$ = initial state
5) $Z_0$ = start symbol.
6) $F$ = set of final states
7) $\delta$ = mapping function

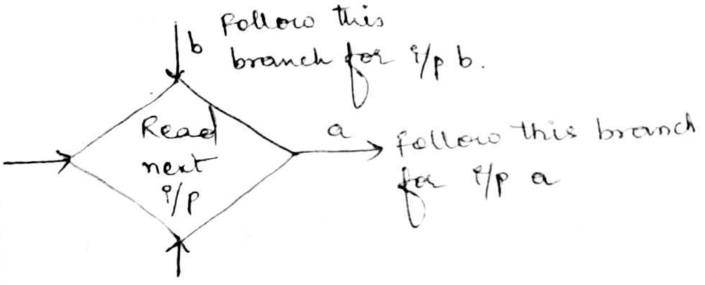
Following symbols are used while drawing the PDA.

(Start) = start symbol

(Accept) = Accept state (It is always a final state)

(Reject) = Reject state

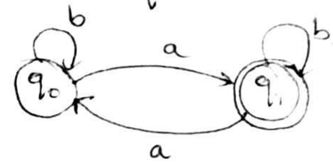
↓ = connecting edge.

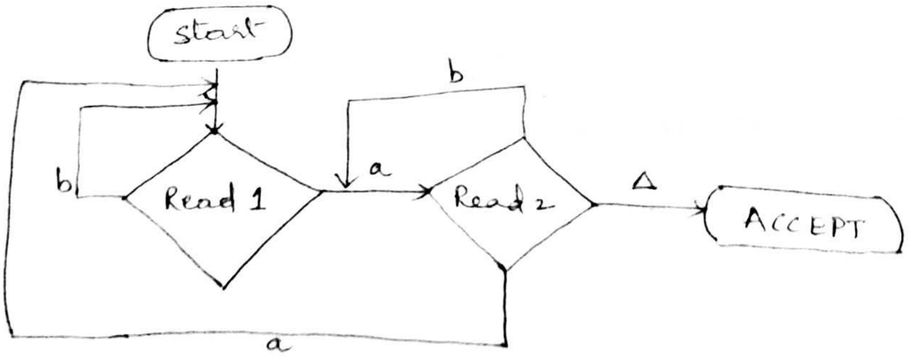


Read symbol

(Ex 1) Design push down Automata which accepts only odd number of a's over $\Sigma = \{a, b\}$.

Soln





Note that this PDA is very much resembling to the FA drawn.

We will simulate this PDA for some string.

consider i/p string "baaab" is placed on i/p tape.

| | | | | | |
|---|---|---|---|---|---|
| b | a | a | a | b | |

We will read only one character at a time from left to right. As soon as we read Δ we should reach to ACCEPT state of PDA.

b a a a b Δ Read 1
↑

b a a a b Δ Read 2
  ↑

b a a a b Δ Read 1
    ↑

b a a a b Δ Read 2
      ↑

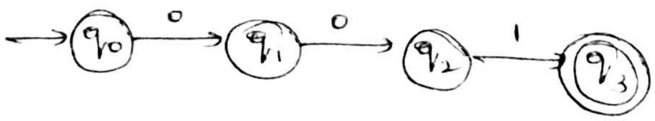
b a a a b Δ Read 2
        ↑

b a a a b Δ ACCEPT.
          ↑

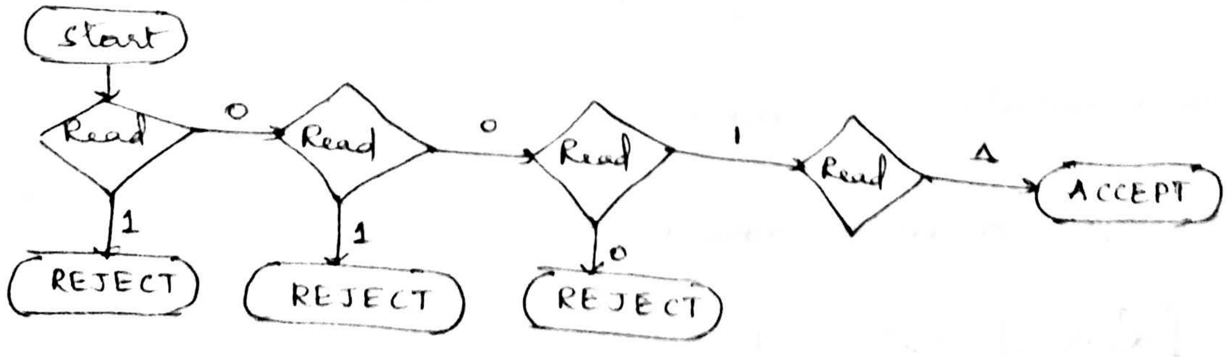
This shows that only odd no. of a's are allowed and there is no restriction on no. of b's.

(Ex 2) Design PDA for the language L = {001}

sol^n The simple FA for this language is



We can draw an eqvt PDA as

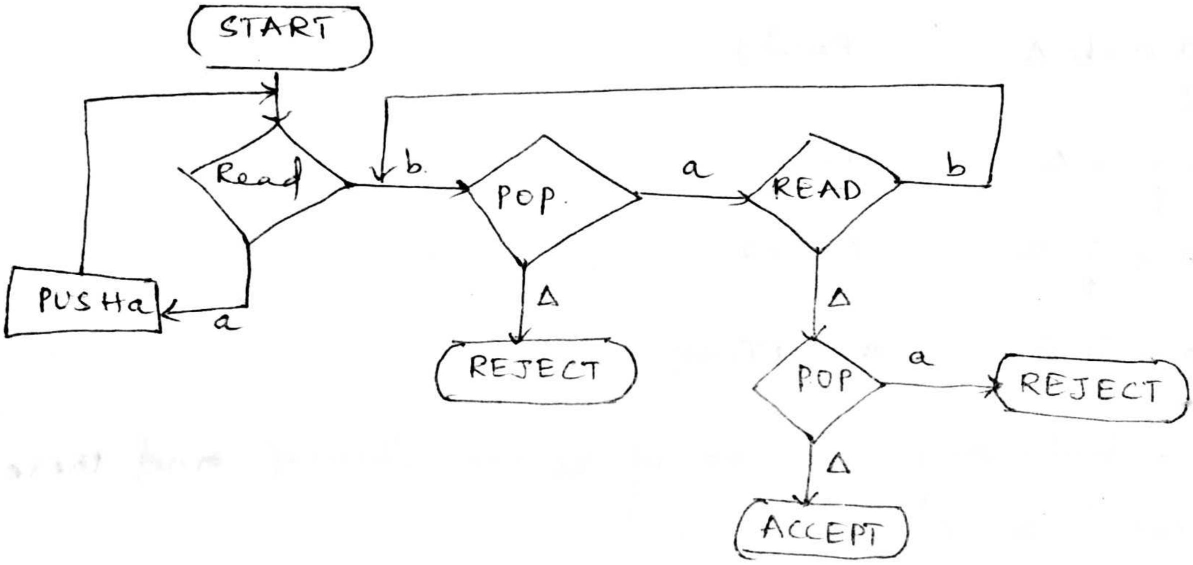


Ex-3 Design a PDA for the language $L = \{a^n b^n\}$ where $n \geq 1$.

Sol^n We cannot draw FA for this problem bcz, here the condition is that what many no. of a's are occuring, those many b's should be after a's.

- If n=5 then the string will be aaaaabbbbb.

Let us draw the PDA for the same using memory i.e stack.



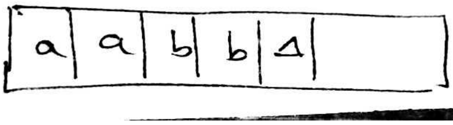
We start with start symbol.

- At the first read statement if we read a, we push that a onto the stack. By this all a's will be pushed onto the stack,

- Now when we read one b we will pop one a,
- This process will be repeated till the end of i/p.

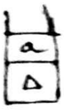
Let us simulate this machine for some valid string.

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



step 1 : we start with start symbol.

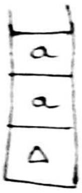
step 2 : we will reach to read state, we will read first a and push it onto stack



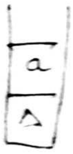
The stack initially contains Δ that means it is empty.

step 3 : Read next symbol from i/p tape

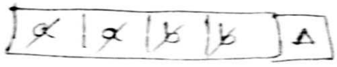
It is a. push it onto stack.



step 4 : Read b from the tape and pop one a from the stack.



step 5 : After reading 2nd b we pop one more a

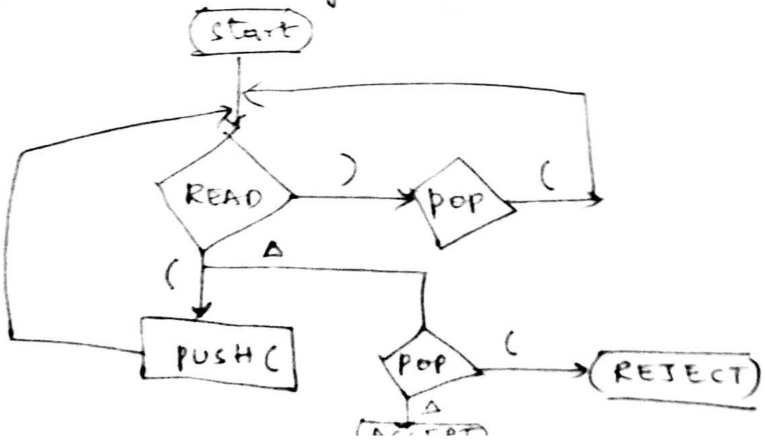


Now both tape and stack both are empty.

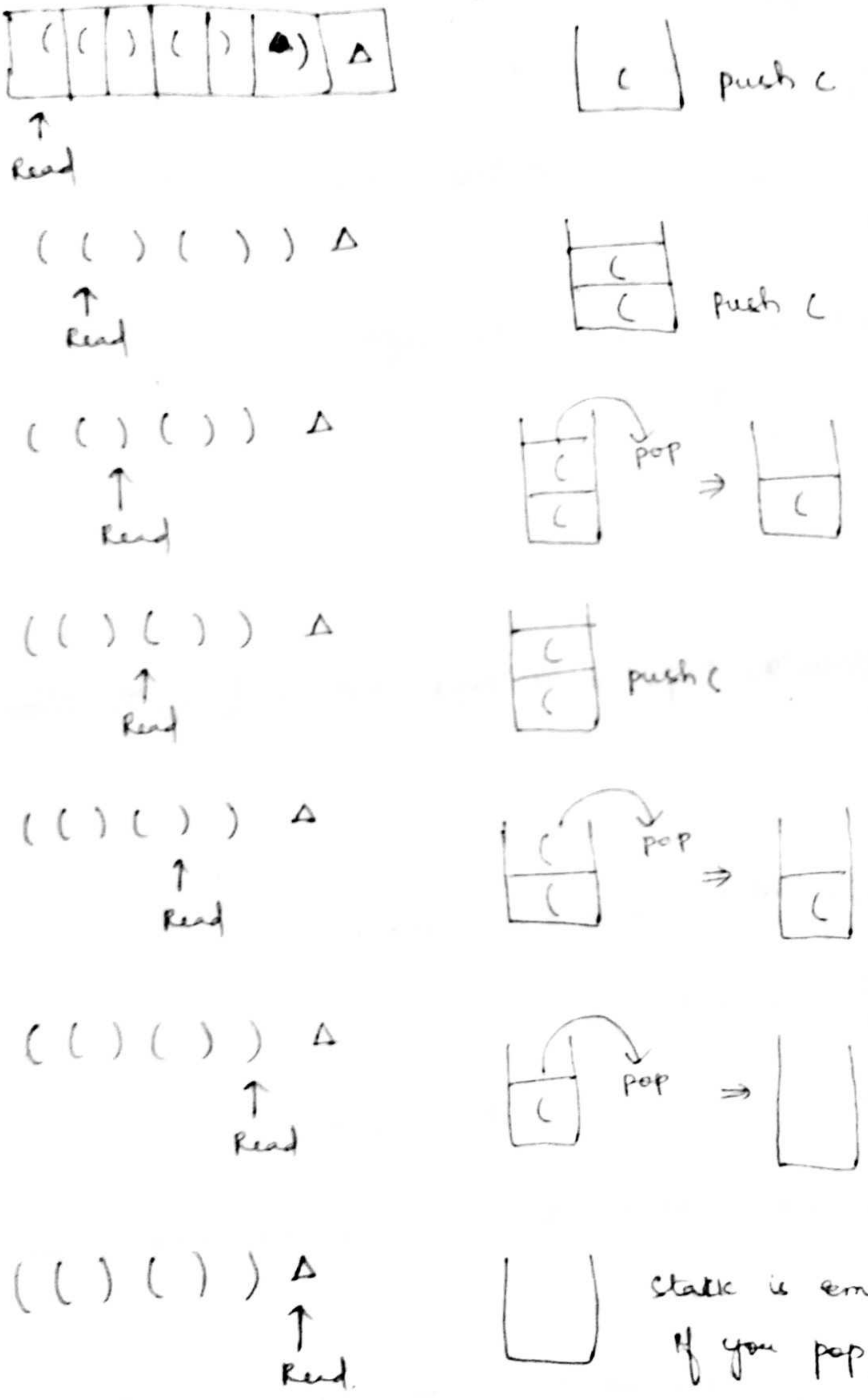
Ex-4 Design PDA that checks the well formedness of parenthesis

soln Ex:- (()) or ()()() or (()())

We will design PDA and we will check a valid string.



Let us simulate PDA for the I/P



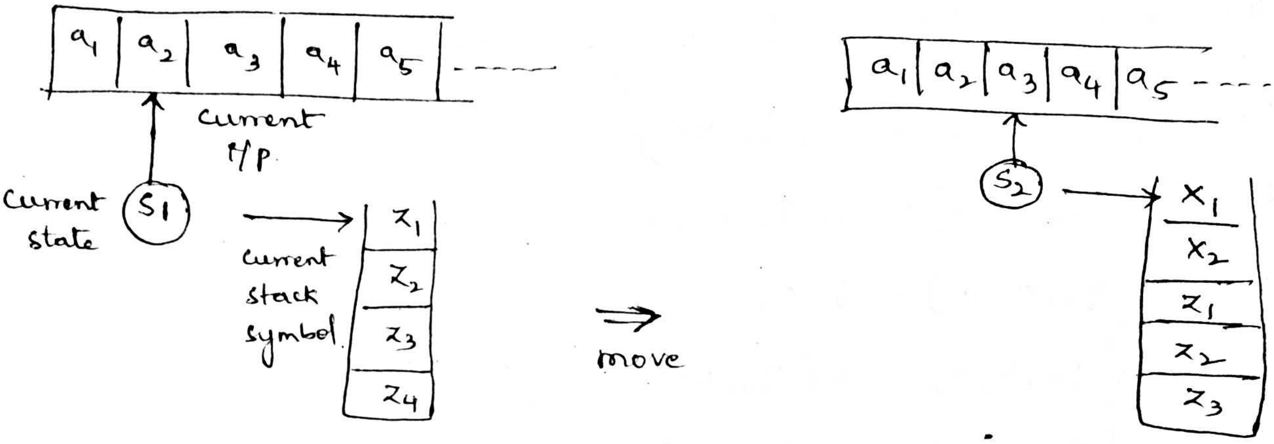
Stack is empty

If you pop, you will get Δ.

# INSTANTANEOUS DESCRIPTION

- It is given by ID.

The moves of ID's are as shown below.



- If we are reading the current symbol $a_2$ at current state $S_1$ and current stack symbol $z_1$ then after a move we will reach to state $S_2$ and there will be some new symbol on top of the stack.

- This description can be represented as.

current stack top

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

Read i/p on the tape

change the state from $q_0$ to $q_1$

push x onto the stack.

(a) <u>Push operation</u>.

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

Read i/p on tape

current stack top

change the state from $q_0$ to $q_1$

Pop the stack.

(b) <u>PoP operation</u>.

EXAMPLE PROBLEMS.

(1) For a given PDA

$M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{a, b, z_0\}, \delta, q_0, z_0, \{q_2\})$

the mapping function δ is given as

$R_1 = \delta(q_0, a, z_0) = \delta(q_0, az_0)$

$R_2 = \delta(q_0, b, z_0) = \delta(q_0, bz_0)$

$R_3 = \delta(q_0, a, a) = \delta(q_0, aa)$

$R_4 = \delta(q_0, b, a) = \delta(q_0, ba)$

$R_5 : \delta(q_0, a, b) = \delta(q_0, ab)$

$R_6 : \delta(q_0, b, b) = \delta(q_0, bb)$

$R_7 : \delta(q_0, c, z_0) = \delta(q_1, z_0)$

$R_8 : \delta(q_0, c, a) = \delta(q_1, a)$

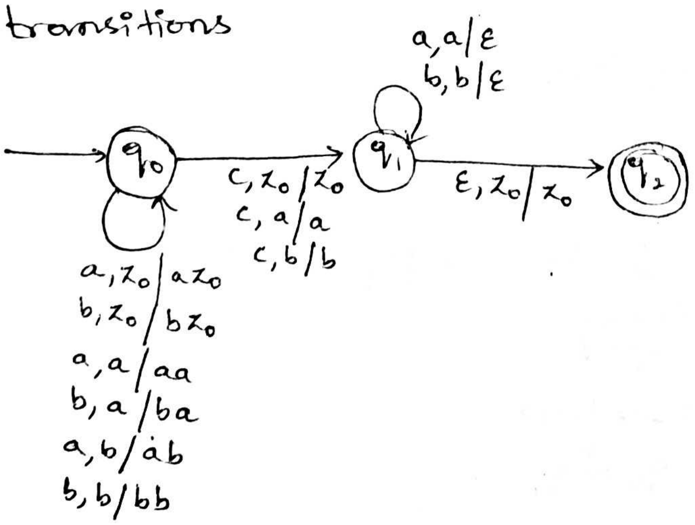
$R_9 : \delta(q_0, c, b) = \delta(q_1, b)$

$R_{10} : \delta(q_1, a, a) = \delta(q_1, \varepsilon)$

$R_{11} : \delta(q_1, b, b) = \delta(q_1, \varepsilon)$

$R_{12} : \delta(q_1, \varepsilon, z_0) = \delta(q_2, z_0)$

sol$^n$ we will first draw the transition graph for given δ transitions



Now we will summarize this PDA for the input bbacabb.

From initial ID, we will try to match rules $R_1$ and $R_{12}$.

$(q_0, bbacabb, Z_0) \vdash (q_0, bacabb, bZ_0)$

$\vdash (q_0, acabb, bbZ_0)$

$\vdash (q_0, cabb, abbZ_0)$

$\vdash (q_1, abb, abbZ_0)$

$\vdash (q_1, bb, bbZ_0)$

$\vdash (q_1, b, bZ_0)$

$\vdash (q_1, \varepsilon, Z_0)$

$\vdash (q_2, Z_0)$

Which is Accept state.

(2) Design a PDA for accepting a language $\{L = a^n b^n \mid n \geq 1\}$.

Sol$^n$ This is a language in which equal number of a's are followed by equal number of b's.

The instantaneous description can be given as

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

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

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

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

$\delta(q_1, \varepsilon, Z_0) = (q_2, \varepsilon)$

Where $q_0$ is start state and $q_2$ is accept state.

We will stimulate this PDA for following string.

$(q_0, aaabbb, Z_0) \vdash (q_0, aabbb, aZ_0)$

$\vdash (q_0, abbb, aaZ_0)$

$\vdash (q_0, bbb, aaaZ_0)$

$\vdash (q_1, bb, aaZ_0)$.

$\vdash (q_1, b, aZ_0)$

$\vdash (q_1, \varepsilon, Z_0)$

$\vdash (q_2, \varepsilon)$

Accept state.

③ Design a PDA that accepts a string of well formed parenthesis. Consider the parenthesis is as (, ), [, ], {, }.

Soln We can write ID as

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

$\delta(q_0, \{, Z_0) = (q_1, \{Z_0)$

$\delta(q_0, [, Z_0) = (q_1, [Z_0)$

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

$\delta(q_1, [, [) = (q_1, [[)$

$\delta(q_1, \{\{) = (q_1, \{\{)$

$\delta(q_1, (, [) = (q_1, ([)$

$\delta(q_1, (, \{) = (q_1, (\{)$

$\delta(q_1, [, () = (q_1, [()$

$\delta(q_1, \{, () = (q_1, \{()$

$\delta(q_1, [, \{) = (q_1, [\{)$

$\delta(q_1, \{, [) = (q_1, \{[)$

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

$\delta(q_1 ], [) = (q_1, \varepsilon)$

$\delta(q_1, \}, \{) = (q_1, \varepsilon)$

$\delta(q_1, \varepsilon, Z_0) = (q_0, Z_0)$

The PDA $P = (\{q_0\ q_1\}, \{, (, [, \{, ), ], \}\}, \{[\ (\ \{\ Z_0\},$
$\delta, q_0, Z_0, \{q_0\})$

Let us simulate it for some input string.

$(\{\}[])$

$\delta(q_0, (\{\}[]), Z_0) \vdash (q_1, \{\}[]), Z_0)$

$\vdash (q_1, \}[]), \{(Z_0)$

$\vdash (q_1, []), (Z_0)$

$\vdash (q_1, ]), [(Z_0)$

$\vdash (q_1, )), (Z_0)$

$\vdash (q_1, \varepsilon, Z_0)$

$(q_0, Z_0)$

Accept state.

(1) Design a PDA for the language $L = \{w \mid w \in (a+b)^*$ and $n_a(w) > n_b(w)\}$.

Sol$^n$ $n_a(w)$ means total no of a's in i/p string.

$n_b(w)$ means total no of b's in i/p string.

The total no of a's are more than total no of b's.

The ID can be

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

$\delta(q_0, b, Z_0) = (q_0, bZ_0)$

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

$\delta(q_0, b, b) = (q_0, bb)$

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

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

$\delta(q_0, \varepsilon, a) = (q_f, a)$

Where $P = (\{q_0\ q_f\}, \{a, b\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_f\}\}$

(5) Design PDA for the language that accepts strings with $n_a(w) < n_b(w)$ where $w \in (a+b)^*$.

Sol$^n$ Here in this problem the language requires more number of b's than total no of a's.

Hence the ID will be,

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

$\delta(q_0, b, z_0) = (q_0, bz_0)$

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

$\delta(q_0, b, b) = (q_0, bb)$

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

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

$\delta(q_0, \varepsilon, b) = (q_f, b)$ ← b's are more in number.

Simulate this PDA for the input "abbab"

$\delta(q_0, abbab, z_0) \vdash (q_0, bbab, az_0)$

$\vdash (q_0, bab, z_0)$

$\vdash (q_0, ab, bz_0)$

$\vdash (q_0, b, z_0)$

$\vdash (q_0, \varepsilon, bz_0)$

$\vdash (q_f, b)$

Accept state.

## Acceptance of Language by PDA:

The language can be accepted by a PDA using two approaches -

1. Acceptance by final state: The PDA accepts its input by consuming it and then it enters in the final state.
2. Acceptance by empty stack: On reading the input string from initial configuration for some PDA, the stack of PDA gets empty.

Acceptance by final state:

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

Then the language $L(P)$ is called the language accepted by final state.

The language $L(P)$ can be defined as

$$L(P) = \{ w \mid (q_0, w, Z_0) \overset{*}{\vdash} (P, \varepsilon, \alpha) \}$$

i.e The PDA P reads the entire input w and enters in a final state P.

Theorem:

For the language $L_{wwr} = \{ ww^R \mid w \text{ is in } (0+1)^* \}$.

Consider the PDA P that accepts string x by final state if and only if x is of the form $ww^R$.

Proof: We will prove theorem in two parts.

(i) If

(ii) only if.

Consider if-part first. We have to prove that if $x = ww^R$ then PDA leads to accept state.

$(q_0, ww^R, z_0) \vdash^* (q_0, w^R, w^R z_0) \vdash (q_1, w^R, w^R z_0)$

$\vdash^* (q_1, \varepsilon, z_0) \vdash (p, \varepsilon, z_0)$

That means PDA P reads w from input tape and stores it onto the stack in reverse. Then it goes to state $q_1$ and in $q_1$ state if $w^R$ matches with $w^R$ on stack top then finally it goes to final state P.

Now we will prove only if part.

That is final state is achieved only if the input is of the form $x = ww^R$.

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

i.e to reach to state p we should have PDA to be in state $q_1$ with $z_0$ on the top of stack with i/p $\varepsilon$.

- And to reach to $q_1$ with input $\varepsilon$ and $z_0$ at the top we should have

$(q_1, w^R, w^R z_0)$

Hence we can make a more general statement as

$\delta(q_0, x, \alpha) \vdash^* (q, \varepsilon, \alpha)$ then x is of the form $ww^R$.

Basis: Consider $x = \varepsilon$ then

$x = ww^R = \varepsilon$.

then $(q_0, x, \alpha) \vdash^* (q_1, x, \beta)$ becomes

$(q_0, \varepsilon, \alpha) \vdash^* (q_1, \varepsilon, \alpha)$ is true.

Induction: Now consider $x = a_1, a_2, \ldots, a_n$ for $n > 0$.

There are two moves that PDA P can make.

1. for ID $(q_0, x, \alpha) \vdash (q_1, x, \alpha)$.

Now being in $q_1$ state then PDA P can pop the stack.

As PDA P reads every input symbol the stack is popped.

Thus

$(q_1, x, \alpha) \vdash^* (q_1, \varepsilon, \beta)$ where $\beta$ will be shorter than $\alpha$.

2. $(q_0, a_1, a_2 ---- a_n, \alpha) \vdash (q_0, a_2 ---- a_n, a_1\alpha)$

We can continue this move by pushing each i/p symbol onto the stack. But the last move for a final state would be

$(q_1, a_n\ a_1\alpha) \vdash (q_1, \varepsilon, \alpha)$

In that case $a_1 = a_n$ from this we can conclude that $a_2 ---- a_{n-1}$ is of the form $yy^R$ for some $y$.

We can conclude that $x$ is of the form $ww^R$.

## ACCEPTANCE BY EMPTY STACK :

Let $P = (Q, \Sigma, T, \delta, q_0, z_0, F)$ be the PDA then the language accepted by empty stack $N(P)$ is defined as.

$N(P) = \{w \mid (q_0, w, z_0) \vdash^* (P, \varepsilon, \varepsilon)\}$.

## EQUIVALENCE OF EMPTY STACK AND FINAL STATE :

1) If $L = N(P_1)$ for some PDA $P_1$ then there is a PDA $P_2$ such that $L = L(P_2)$. That means the language accepted by empty stack PDA will also be accepted by final state PDA.

2) If there is a language $L = L(P_1)$ for some PDA $P_1$ then there is a PDA $P_2$ such that $L = N(P_2)$.

That means language accepted by final state PDA is also acceptable by empty stack PDA.

## PUSHDOWN AUTOMATA AND CFG:

- FSM acts as the acceptor model for regular expressions and regular languages.
- Similarly PDA is a model for recognizing context free languages.
- For the languages that can be modelled by PDA, it is possible to write CFG for them.
- But it is not possible to write CFG for deterministic PDA.
- We require a non deterministic model of PDA to accept certain CFGs.
- Fundamentally any CFL can be modelled by NPDA.

## NON DETERMINISTIC PUSH DOWN AUTOMATA:

- NDPDA is very much similar to NFA.
- The CFG which accepts deterministic PDA accepts non deterministic PDAs as well.
- IIIly there are some CFGs which can be accepted only by NPDA and not by DPDA.
- Thus NPDA is more powerful than DPDA.

## EXAMPLE PROBLEMS:

(1) Construct a NPDA for the language L containing all the strings which are palindrome over $\Sigma = \{a, b\}$.

Sol^n L contains all the strings which are palindrome.

$L = \{ \varepsilon, aba, a, b, aa, bb, bab, bbabb, aabaa \ldots \}$

The string can be of odd palindrome or even palindrome.

(1) We will push a symbol onto the stack till half of the string, then we will read each symbol and then perform pop operation.

(2) Compare to see whether the symbol which is popped is similar to the symbol which is read.

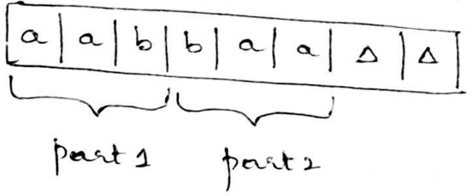
(3) When we reach to end of the i/p, we expect the stack to be empty.

For eg:- if the string is,

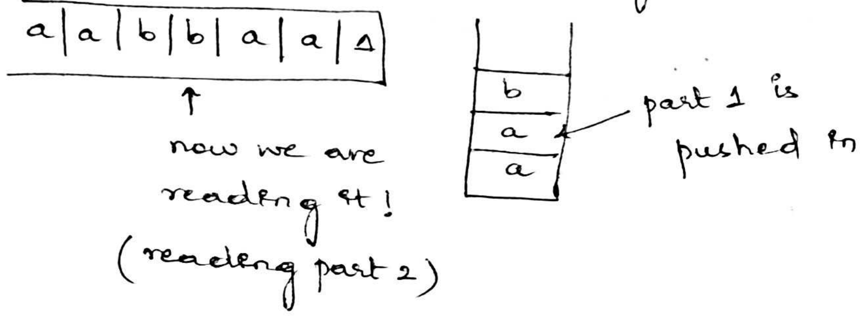
| | | | | | | |
|---|---|---|---|---|---|---|
| a | a | b | b | a | a | Δ |

input tape

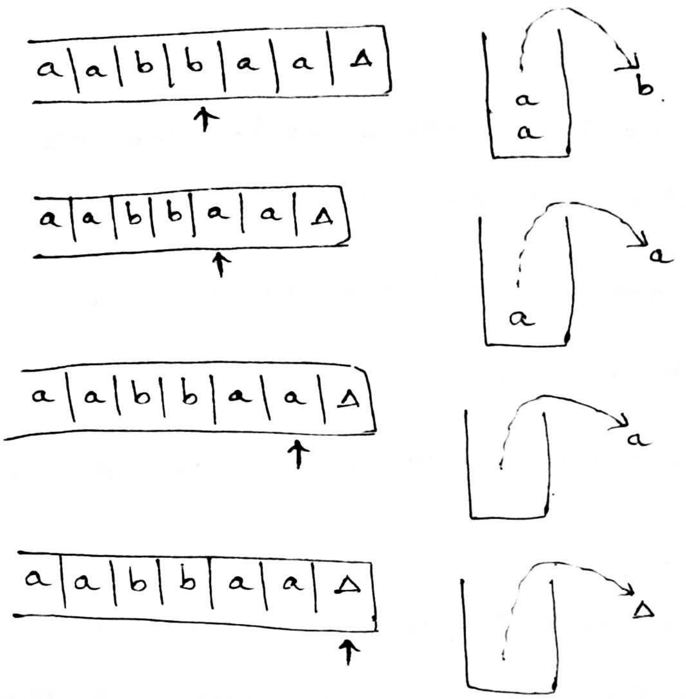
Now the list can be divided into two halves.



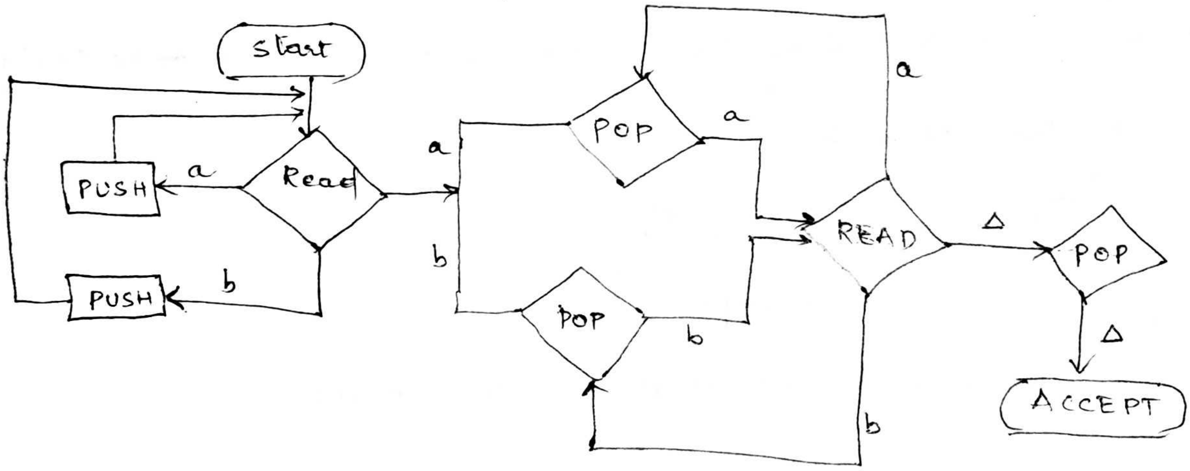
We will push all the elements of part 1 onto the stack.

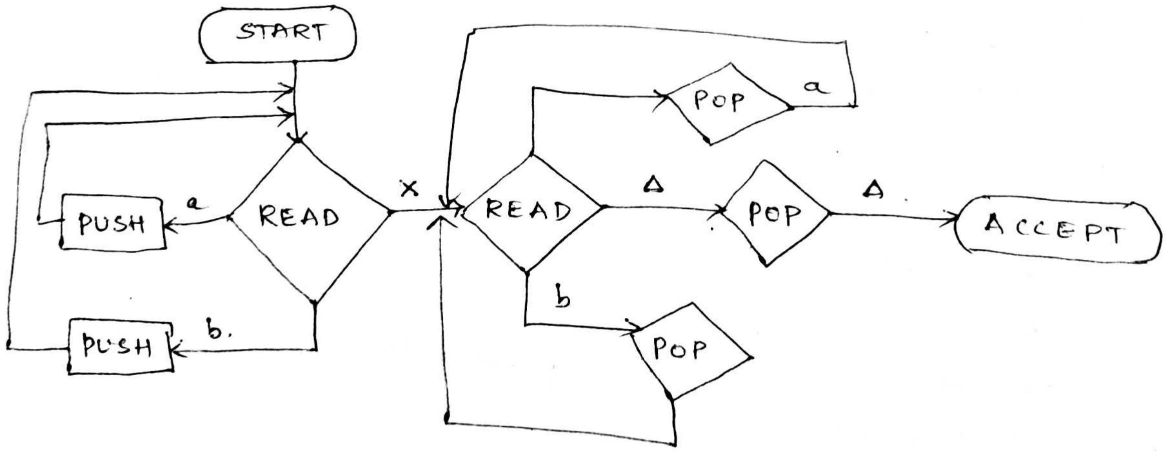


When the tape reads b, we will see whether at the top of the stack same symbol is there or not.


a | a | b | b | a | a | Δ
a
a
b.
a | a | b | b | a | a | Δ
a
a
a | a | b | b | a | a | Δ
a
a | a | b | b | a | a | Δ
Δ

The NPDA is as below,





Thus there exists a NPDA for a given DPDA.

But reverse is not always true.

## EXAMPLE PROBLEMS :

① Construct the PDA M for the language $L = \{ww^R \mid w \in \{a,b\}^*\}$ such that $L = L(M)$

Soln The PDA for accepting this language will be given as below.

$M = (Q, \Sigma, \delta, \Gamma, S, z_0, F)$

$= (\{q_0, q_1, q_f\}, \{a,b\}, \delta, \{a,b\}, \{q_0\}, \{z_0\}, \{q_f\}\}$

The mapping function $\delta$ will be -

1. $\delta(q_0, a, z_0) = (q_0, az_0)$
2. $\delta(q_0, b, z_0) = (q_0, bz_0)$
3. $\delta(q_0, a, a) = (q_1, aa)$
4. $\delta(q_0, b, b) = (q_0, bb)$
5. $\delta(q_0, a, b) = (q_0, ab)$
6. $\delta(q_0, b, a) = (q_0, ba)$
7. $\delta(q_0, \varepsilon, z_0) = (q_1, z_0)$
8. $\delta(q_0, \varepsilon, a) = (q_1, a)$
9. $\delta(q_0, \varepsilon, b) = (q_1, b)$
10. $\delta(q_1, a, a) = (q_1, \varepsilon)$
11. $\delta(q_1, b, b) = (q_1, \varepsilon)$
12. $\delta(q_1, \varepsilon, z_0) = (q_f, \varepsilon)$ Accept

Simulation :- We will push the a's and b's onto the stack until the mid of the given string. At the middle position $\varepsilon$ is read. At this stage a state is changed from $q_0$ to $q_1$. Being in state $q_1$, if we read a, pop a from top of stack. If we read b, pop b from the stack. Finally we read ε as end of string and at this point stack should be empty.

· so while reading abbbba we read it as abbεbba.

$\delta(q_0, \underline{a}bb\varepsilon bba, \underline{Z_0})$

$\vdash \delta(q_0, bb\varepsilon bba, aZ_0)$

$\vdash \delta(q_0, b\varepsilon bba, baZ_0)$

$\vdash \delta(q_0, \varepsilon bba, \underline{b}baZ_0)$

$\vdash \delta(q_1, \underline{b}ba, \boxed{b}baZ_0)$

$\vdash \delta(\underline{q_1}, \underline{b}a, \underline{b}aZ_0)$

$\vdash \delta(\underline{q_1}, \underline{a}, \underline{a}Z_0)$

$\vdash \delta(q_1, \varepsilon, Z_0)$

$\vdash (q_f, \varepsilon)$ Accept.

## CONVERSION OF CFG TO PDA

For converting given CFG to PDA, by this method. the necessary condition is that the first symbol on R.H.S production must be a terminal symbol.

The rule that can be used to obtain PDA from CFG is

Rule 1: For non terminal symbols, add the rule,

$$\delta(q, \varepsilon, A) = (q, \alpha)$$

where the production rule is $A \rightarrow \alpha$

Rule 2: for each terminal symbols, add the rule,

$\delta(q, a, a) = (q, \varepsilon)$ for every terminal symbol.

## EXAMPLE PROBLEMS:

① construct PDA for the given CFG

$S \rightarrow 0BB$

$B \rightarrow 0S \mid 1S \mid 0$.

Test whether $010^4$ is acceptable by this PDA.

Soln Let PDA $A = \{\{q\}, \{0, 1\}, \{S, B, 0, 1\}, \delta, q, S, \phi\}$

The production rules δ can be given as.

$R1: \delta(q, \varepsilon, S) = \{(q, 0BB)\}$

$R2: \delta(q, \varepsilon, B) = \{(q, 0S), (q, 1S), (q, 0)\}$.

$R3: \delta(q, 0, 0) = \{(q, \varepsilon)\}$

$R4: \delta(q, 1, 1) = \{(q, \varepsilon)\}$.

Testing $010^4$ ie 0100 00 against PDA

$\delta(q, 010000, S) \vdash \delta(q, 010000, \cancel{0}BB)$ ∵ R1

$\vdash \delta(q, 10000, BB)$ ∵ R3

$\vdash \delta(q, 10000, \cancel{1}SB)$ ∵ R2

$\vdash \delta(q, 0000, SB)$ ∵ R4

$\vdash \delta(q, 0000, \cancel{0}BBB)$ ∵ R1

$\vdash \delta(q, 000, BBB)$ ∵ R3

$\vdash \delta(q, 000, \cancel{0}BB)$ ∵ R2

$\vdash \delta(q, 00, BB)$ ∵ R3

$\vdash \delta(q, 00, \cancel{0}B)$ ∵ R2

$\vdash \delta(q, 0, B)$ ∵ R3

$\vdash \delta(q, 0, 0)$ ∵ R2

$\vdash \delta(q, \varepsilon)$ ∵ R3.

ACCEPT.

Thus $010^4$ is accepted by the PDA.

② construct the PDA for the following grammar.

$S \rightarrow AA \mid a$

$A \rightarrow SA \mid b$.

Soln The PDA $P = (Q, \Sigma, \Gamma, \delta, S, F)$

$= (\{q_0 q, q_f\}, \{a,b\}, \{S, A, a, b\}, \{q_f\})$.

The $\delta$, mapping fuction can be given as.

1. $\delta(q_0, \varepsilon, \varepsilon) = (q_0, S\$)$
2. $\delta(q_0, \varepsilon, S) = (q_0, AA)$
3. $\delta(q_0, \varepsilon, S) = (q_0, a)$
4. $\delta(q_0, \varepsilon, A) = (q_0, SA)$
5. $\delta(q_0, \varepsilon, A) = (q_0, b)$
6. $\delta(q_0, a, a) = (q_0, \varepsilon)$
7. $\delta(q_0, b, b) = (q_0, \varepsilon)$
8. $\delta(q_0, \varepsilon, \$) = (q_f, \varepsilon)$ Accept state.

Simulation: consider the string abab for simulation.

$\delta(q_0, abab, \varepsilon) = \vdash \delta(q_0, abab, \underline{S}\$)$

$\vdash \delta(q_0, abab, AA)$

$\vdash \delta(q_0, abab, \textcircled{a}A)$

$\vdash \delta(q_0, bab, A)$

$\vdash \delta(q_0, bab, \underline{S}A)$

$\vdash \delta(q_0, bab, AAA)$

$\vdash \delta(q_0, \underline{b}ab, \underline{b}AA)$

$\vdash \delta(q_0, ab, AA)$

$\vdash \delta(q_0, ab,$

$\vdash \delta(q_0, abab, \underline{S})$

$\vdash \delta(q_0, abab, AA)$

$\vdash \delta(q_0, abab, \underline{S}AA)$

$\vdash \delta(q_0, \underline{a}bab, \textcircled{a}AA)$

$\vdash \delta(q_0, \bullet bab, AA)$

$\vdash \delta(q_0, bab, bA)$

$\vdash \delta(q_0, ab, \underline{A})$

$\vdash \delta(q_0, ab, \underline{S}A)$

$\vdash \delta(q_0, \underline{a}b, \textcircled{a}A)$

$\vdash \delta(q_0, b, A)$

$\vdash \delta(q_0, b, b)$

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

$\vdash (q_f, \varepsilon)$ ⇒ Accept state.

③ convert the grammar

$S \rightarrow 0S1 \mid A$

$A \rightarrow 1A0 \mid S \mid \varepsilon$

to the PDA that accepts the same language by empty stack.

<u>sol</u>

sol$^n$ The PDA can be

$$A = \{ \{q\}, \{0,1\}, \{S, A, 0, 1\}, \delta, q, S, \phi \}$$

The $\delta$ can be,

R1 : $\delta(q, \varepsilon, S) = \{ (q, 0S1), (q, A) \}$

R2 : $\delta(q, \varepsilon, A) = \{ (q, 1A0, (q, S), (q, \varepsilon) \}$

R3 : $\delta(q, 0, 0) = \{ (q, \varepsilon) \}$

R4 : $\delta(q, 1, 1) = \{ (q, \varepsilon) \}$.

(4) Design a PDA for the following grammar

$S \rightarrow 0A$

$A \rightarrow 0AB \mid 1$

$B \rightarrow 1$

Sol^n PDA $P = (Q, \Sigma, T, \delta, S, F)$

$= ((q_0, q_1, q_f), (0,1), (S, A, 0, 1), q_0, q_f\})$

The mapping function $\delta$ will be

$\delta(q_0, \varepsilon, \varepsilon) = \delta(q_1, S\$)$

$\delta(q_1, \varepsilon, S) = \delta(q_1, 0A)$

$\delta(q_1, \varepsilon, A) = (q_1, 0AB)$

$\delta(q_1, \varepsilon, A) = (q_1, 1)$

$\delta(q_1, \varepsilon, B) = (q_1, 1)$

$\delta(q_1, 0, 0) = (q_1, \varepsilon)$

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

$\delta(q_1, \varepsilon, \$) = (q_f, \varepsilon)$ ACCEPT.

Consider the string 000111 for derivation.

$\delta(q_0, \varepsilon, \varepsilon) = \vdash \delta(q_1, S\$)$

$\vdash \delta(q_1, 000111, S\$)$

$\vdash \delta(q_1, \underline{0}00111, \underline{0}A)$

$\vdash \delta(q_1, 00111, A)$ $\rightarrow \delta(q_1, 1, B)$

$\vdash \delta(q_1, \underline{0}0111, \underline{0}AB)$ $\rightarrow \delta(q_1, 1, 1)$

$\vdash \delta(q_1, 0111, \underline{A}B)$ $\rightarrow \delta(q_1, \varepsilon, \$)$

$\vdash \delta(q_1, \underline{0}111, \underline{0}ABB)$ $\rightarrow (q_f, \varepsilon)$

$\vdash \delta(q_1, 111, \underline{A}BB)$ ACCEPT.

$\vdash \delta(q_1, \underline{1}11, \underline{1}BB)$

$\vdash \delta(q_1, 11, BB)$

$\vdash \delta(q_1, \underline{1}1, \underline{1}B)$

(5) Let G be the grammar given by

$S \rightarrow aABB \mid aAA$

$A \rightarrow aBB \mid a$

$B \rightarrow bBB \mid A$

construct the PDA that accepts the language generated by this grammar G.

<u>Soln</u> 1st eliminate unit prodns.

$S \rightarrow aABB \mid aAA$

$A \rightarrow aBB \mid a.$

$B \rightarrow bBB \mid aBB \mid a$

PDA $P = (\{q_0, q_f\}, \{a, b\}, \{Z_0, S, A, B, a, b\}, \delta, q_0, Z_0, q_f)$

The mapping function $\delta$ can be.

$\delta(q_0, \varepsilon, S) = (q_0, aABB), (q_0, aAA)$

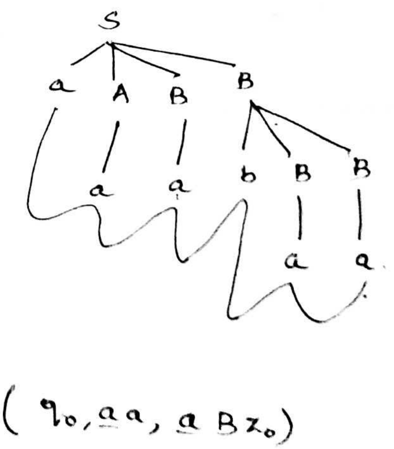
$\delta(q_0, \varepsilon, A) = (q_0, aBB), (q_0, a)$

$\delta(q_0, \varepsilon, B) = (q_0, bBB), (q_0, aBB), (q_0, a)$

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

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

$\delta(q_0, \varepsilon, Z_0) = (q_0, \varepsilon)$ ACCEPT



<u>Simulation</u>: $\delta(q_0, \underline{\varepsilon}aaabaa, \underline{S}Z_0)$

$\vdash \delta(q_0, \underline{a}aabaa, \underline{a}ABBZ_0)$

$\vdash \delta(q_0, aabaa, ABBZ_0)$

$\vdash \delta(q_0, \underline{a}abaa, \underline{a}BBZ_0)$

$\vdash \delta(q_0, abaa, BBZ_0)$

$\vdash \delta(q_0, \underline{a}baa, \underline{a}BZ_0)$

$\vdash \delta(q_0, baa, BZ_0)$

$\vdash \delta(q_0, \underline{b}aa, \underline{b}BBZ_0)$

$\vdash \delta(q_0, aa, BBZ_0)$

$\rightarrow \delta(q_0, \underline{a}a, \underline{a}BZ_0)$

$\vdash \delta(q_0, a, BZ_0)$

$\vdash \delta(q_0, \underline{a}, \underline{a}Z_0)$

$\vdash \delta(q_0, \varepsilon, Z_0)$

$\vdash (q_0, \varepsilon)$ ACCEPTED//

⑥ Let G be a CFG that generates the set of palindromes given by $S \rightarrow aSa \mid bSb \mid a \mid b$

find The PDA that accepts $L(G)$.

soln To find PDA from given CFG, we need to convert the CFG to some normal form.

We will convert this CFG to GNF.

$S \rightarrow aSA \mid bSB \mid a \mid b.$

$A \rightarrow a$

$B \rightarrow b.$

The PDA $P = (\{q_0, q_f\}, \{a, b\}, \{S, A, a, b, B\}, \delta, q_0, Z_0, q_f)$

$\delta(q_0, \varepsilon, S) = (q_0, aSA), (q_0, bSB), (q_0, a), (q_0, b)$

$\delta(q_0, \varepsilon, A) = (q_0, a)$

$\delta(q_0, \varepsilon, B) = (q_0, b)$

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

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

$\delta(q_0, \varepsilon, Z_0) = (q_f, \varepsilon)$ ACCEPT

<u>Simulation:</u> consider string abbbbba.

$\delta(q_0, \varepsilon abbbbba, S) \vdash (q_0, \underline{a}, bbbbba, \underline{a}SA)$

$\vdash (q_0, bbbbba, SA)$

$\vdash (q_0, \underline{b}bbbba, \underline{b}SBA)$

$\vdash (q_0, bbbba, SBA)$

$\vdash (q_0, \underline{b}bbba, \underline{b}SBBA)$

$\vdash (q_0, bbba, \underline{S}BBA)$

$\vdash (q_0, bbba, bBBA)$

$\vdash \delta(q_0, bba, BBA)$

$\vdash \delta(q_0, \underline{b}ba, \underline{b}BA)$

$\vdash \delta(q_0, ba, BA)$

$\vdash \delta(q_0, \underline{b}a, \underline{b}A)$

$\vdash \delta(q_0, a, A)$

$\vdash \delta(q_0, \underline{a}, \underline{a})$

$\vdash \delta(q_0, \varepsilon, Z_0)$

$\vdash (q_f, \varepsilon)$ Accept

(7) Let G be a CFG with the following productions

$S \rightarrow aBc$
$A \rightarrow abc$
$B \rightarrow aAb.$
$C \rightarrow AB$
$C \rightarrow c.$

Construct PDA M such that the language generated by M and G are equivalent.

Sol^n If we observe the prod^n rules, We notice that $C \rightarrow AB$, $C \rightarrow c$ are non reacheable from start state S.

Hence these are supposed to be useless productions.

Therefore we will eliminate these productions.

The productions are then

$S \rightarrow aBc$
$A \rightarrow abc$
$B \rightarrow aAb.$

For constructing a PDA from given CFG it is necessary to convert this CFG to some normal form.

Hence we will convert this CFG to GNF.

$S \rightarrow aBP$

$A \rightarrow aQP$

$B \rightarrow aAQ$

$P \rightarrow c$

$Q \rightarrow b.$

Now the PDA $P = (Q, \Sigma, T, \delta, S, F)$

$= ((q_0, q_f), (a, b) ( S, A, B, P, Q, a, b, c\} \{q_0\} \{q_f\}$
$\delta, Z_0)$

The mapping function $\delta$ can be given as follows.

$\delta(q_0, \varepsilon, S) = (q_0, aBP)$

$\delta(q_0, \varepsilon, A) = (q_0, aQP)$

$\delta(q_0, \varepsilon, B) = (q_0, aAQ)$

$\delta(q_0, \varepsilon, P) = (q_0, c)$

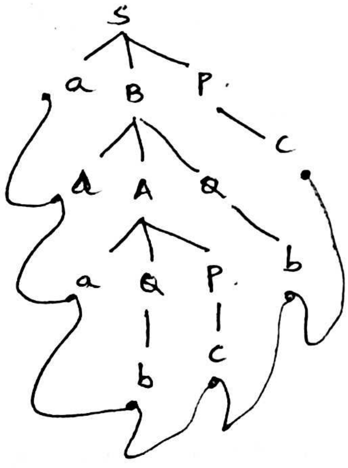
$\delta(q_0, \varepsilon, Q) = (q_0, b)$

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

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

$\delta(q_0, c, c) = (q_0, \varepsilon)$

$\delta(q_0, \varepsilon, Z_0) = (q_f, \varepsilon)$ Accept.



simulate the string

aaa bcbc.

(8) convert the CFG to PDA

$S \rightarrow aSbb \mid aab.$

Sol^n The given CFG will be converted to GNF. It is as follows

Rule : $S \rightarrow aSbb$

$S \rightarrow aSBB$

$B \rightarrow b.$

Rule :· $S \rightarrow aab.$

$S \rightarrow aAB$

$A \rightarrow a$

To summarize the CFG will now be

$S \rightarrow aSBB \mid aAB$

$A \rightarrow a$

$B \rightarrow b.$

The PDA $P = (\{q\}, \{a,b\}, \{a,b,S,A,B\}, \delta, q, z_0, q\}$

$\delta(q, \varepsilon, S) = \{(q\ aSBB), (q, aAB)\}$

$\delta(q, \varepsilon, A) = \{(q, a)\}$

$\delta(q, \varepsilon, B) = \{(q, b)\}$

$\delta(q, a, a) = \{(q, \varepsilon)\}$

$\delta(q, b, b) = \{(q, \varepsilon)\}$

$\delta(q, \varepsilon, Z_0) = \{(q_f, \varepsilon)\}$ Accept

(9) Convert the following CFG to PDA

$S \rightarrow aA \mid bB$

$A \rightarrow aB \mid a$

$B \rightarrow b.$

Sol^n The PDA $P = (\{q\}, \{a,b\}, \{S, A, B, a, b\}, \delta, q, z_0, q).$

$\delta(q, \varepsilon, S) = \{(q, aA), (q, bB)\}$

$\delta(q, \varepsilon, A) = \{(q, aB), (q, a)\}$

$\delta(q, \varepsilon, B) = \{(q, b)\}.$

$\delta(q, a, a) = \{(q, \varepsilon)\}$

$\delta(q, b, b) = \{(q, \varepsilon)\}$

$\delta(q, \varepsilon, Z_0) = \{(q_f, \varepsilon)\}$ Accept state.

String:- aab.

$\delta(q, aab, S)$

$\vdash \delta(q, \underline{a}ab, \underline{a}A)$

$\vdash \delta(q, ab, A)$

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

$\vdash \delta(q, b, B)$

$\vdash \delta(q, b, b)$

$\vdash \delta(q, \varepsilon, Z_0)$

$(q_f, \varepsilon) \Rightarrow$ Accept

CONSTRUCTION OF CFG FROM GIVEN PDA.

(or)

## PDA TO CFG

Theorem:- If $A = (Q, \Sigma, T, \delta, q_0, Z_0, F)$ is a PDA Then there exists CFG G which is accepted by PDA A.

Let G be a CFG which could be generated by PDA A. The G can be defined as $G = \{V, T, P, S\}$ where S is a start symbol.

For getting The production rules P we will apply the following algorithm.

Algorithm for getting production rules of CFG.

Rule 1: The start symbol prod$^n$ can be

$$S \rightarrow [q_0, Z_0, q]$$

Where $q$ indicates the next state and $q_0$ is a start state.

Rule 2: If there exists a move of PDA

$$\delta(q, a, Z) = \{(q', \varepsilon)\}$$

Then the prod$^n$ rule can be written as

$$\delta(q, Z, q') \rightarrow a$$

Rule 3: If there exist a move of PDA as

$$\delta(q, a, Z) = \{(q', Z_1, Z_2 \text{----} Z_n)\}$$

Then the production rule of CFG can be written as.

$$\delta(q, a, Z) \longrightarrow a[q_1, Z_1, q_2][q_2, Z_2, q_3][q_3, Z_3, q_4] \text{----} [q_m, Z_m, q']$$

EXAMPLE PROBLEMS:

(1) The PDA is as given below.

$$A = (\{q_0, q_1\}, \{0, 1\}, \{S, A\}, \delta, q_0, S, \phi)$$

Where $\delta$ is as given below.

$$\delta(q_0, 1, S) = \{(q_0, AS)\}$$

$$\delta(q_0, \varepsilon, S) = \{(q_0, \varepsilon)\}$$

$$\delta(q_0, 1, A) = \{(q_0, AA)\}$$

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

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

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

Construct the CFG equivalent to this PDA.

Sol<sup></sup>

Solⁿ Let us build the production rules as

using rule 1 from the algorithm

$P_1 : S \longrightarrow [q_0\ S\ q_0]$

$P_2 : S \longrightarrow [q_0, S, q_1]$

using rule 3 of algorithm for the

$\delta(q_0, 1, \underline{S}) = \{q_0, AS)\}$ we get.

$P3 : [q_0\ S\ q_0] \longrightarrow 1[q_0, A, q_0][q_0\ S\ q_0]$

$P4 : [q_0\ S\ q_0] \longrightarrow 1[q_0\ A\ q_1][q_1\ S\ q_0]$

$P5 : [q_0\ S\ q_1] \longrightarrow 1[q_0\ A\ q_0][q_0\ S\ q_1]$

$P6 : [q_0\ S\ q_1] \longrightarrow 1[q_0\ A\ q_1[q_1\ S\ q_1]$

Now for $\delta(q_0, \varepsilon, S) = (q_0, \varepsilon)$

using rule 2 of algorithm we get,

$P7 : [q_0, S, q_0] \longrightarrow \varepsilon$

for $\delta(\underline{q_0}, 1, \underline{A}) = (\underline{q_0}, AA)$ using rule 3 gives

$P8 : [q_0, A, q_0] = 1[q_0, A, q_0][q_0\ A, q_0]$

$P9 : [q_0\ A, q_1] = 1[q_0, A, q_1][q_1, A, q_0]$

$P10 : [q_0, A, q_1] = 1[q_0, A, q_0][q_0, A, q_1]$

$P11 : [q_0, A, q_1] = 1[q_0, A, q_1][q_1, A, q_1]$

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

$P12 : [q_0, A, q_0] = 0[q_1, A, q_0]$

$P13 : [q_0, A, q_1] = 0[q_1, A, q_1]$

$\delta(\underline{q_1}, 1, \underline{A}) = (q_1, \varepsilon)$ gives

$P14 : [q_1, A, q_1] \rightarrow 1$

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

$P_{15} : [q_1, S, q_0] \longrightarrow 0[q_0, S, q_0]$

$P_{16} : [q_1, S, q_1] \longrightarrow 0[q_0, S, q_1]$

Ex: 2 Find the CFG corresponding to PDA whose transition mapping is as follows.

$\delta(S, a, X) = (S\ A, x)$

$\delta(S, b, A) = (S, AA)$

$\delta(S, a, A) = (S, \overset{\varepsilon}{AA})$

Solⁿ Now we will apply conversion algorithm for each $\delta$ transition and obtain production rules as follows-

$\delta(\underline{S}, a, \underline{X}) = (S, A, x)$

$P1 : \{S, X, S\} \rightarrow a[S\ A\ S][S\ x\ S]$

$P2 : [S\ X\ S] \longrightarrow a[S\ A\ \underline{S}][\underline{S}\ x\ S]$

$P3 : [S\ x\ \underline{S}] \longrightarrow a[S\ A\ S][S\ x\ \underline{S}]$

$P4 : [S\ x\ \underline{S}] \longrightarrow a[S\ A\ \underline{S}][\underline{S}\ x\ \underline{S}]$

$\delta(\underline{S}, b, \underline{A}) = (\underline{S}, AA)$

$P5 : [S\ A\ S] \longrightarrow b[S\ A\ S][S\ A\ S]$

$P6 : [S\ A\ S] \longrightarrow b[S\ A\ \underline{S}][\underline{S}\ A\ S]$

$P7 : [S\ A\ \underline{S}] \longrightarrow b[S\ A\ S][S\ A\ \underline{S}]$

$P8 : [S\ A\ \underline{S}] \longrightarrow b[S\ A\ \underline{S}][\underline{S}\ A\ \underline{S}]$

$\delta(\underline{S}, a, \underline{A}) = (S, \overset{\varepsilon}{AA})$

$P9 : [S, A, S] \longrightarrow a$

3 construct CFG which accepts $N(M)$ where
$M = (\{q_0, q_1\}, \{a, b\}, \{Z_0, Z\}, \delta, q_0, Z_0, \phi)$ $\delta$ is given by

$\delta(q_0, b, Z_0) = \{(q_0, ZZ_0)\}$

$\delta(q_0, \varepsilon, Z_0) = \{(q_0, \varepsilon)\}$

$\delta(q_0, b, Z) = \{(q_0, ZZ)\}$

$\delta(q_0, a, Z) = \{(q_1, Z)\}$

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

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

Sol$^n$ The prod$^n$s are

$P1 : S \longrightarrow [q_0\ Z_0\ q_0]$

$P2 : S \longrightarrow [q_0\ Z_0\ q_1]$

$\delta(q_0, b, Z_0) = \{(q_0, ZZ_0)\}$ gives

$P3 : [q_0\ Z_0\ q_0] \longrightarrow b[q_0\ Z\ q_0][q_0\ Z_0\ q_0]$

$P4 : \{q_0\ Z_0\ q_0] \longrightarrow b[q_0\ Z\ q_1][q_1\ Z_0\ q_0]$

$P5 : [q_0\ Z_0\ q_1] \longrightarrow b[q_0\ Z\ q_0][q_0\ Z_0\ q_1]$

$P6 : [q_0\ Z_0\ q_1] \longrightarrow b[q_0\ Z\ q_1][q_1\ Z_0\ q_1]$

$\delta(q_0, \varepsilon, Z_0) = (q_0, \varepsilon)$ gives:

$P7 : [q_0\ Z_0\ q_0] \longrightarrow \varepsilon$

$\delta(q_0, b, Z) = (q_0, ZZ)$ gives

$P8 : [q_0\ Z\ q_0] \longrightarrow b[q_0\ Z\ q_0][q_0\ Z\ q_0]$

$P9 : [q_0\ Z\ q_0] \longrightarrow b[q_0\ Z\ q_1][q_1\ Z\ q_0]$

$P10 : [q_0\ Z\ q_1] \longrightarrow b[q_0\ Z\ q_0][q_0\ Z\ q_1]$

$P11 : [q_0\ Z\ q_1] \longrightarrow b[q_0\ Z\ q_1][q_1\ Z\ q_1]$

$\delta(q_1, a, z) = (q_1, z)$ yields.

P12 : $[q_0 \; z \; q_0] \rightarrow a[q_1 \; z \; q_0]$

P13 : $[q_0 \; z \; q_1] \rightarrow a[q_1 \; z \; q_1]$

$\delta(q_1, b, z) = (q_1, \varepsilon)$ gives.

P14 : $[q_1, z, q_1] \rightarrow b$.

$\delta(q_1, a, z_0) = (q_0, z_0)$ gives.

P15 : $[q_1 \; z_0 \; q_0] \rightarrow a[q_0 \; z_0 \; q_0]$

P16 : $[q_1 \; z_0 \; q_1] \rightarrow a[q_0 \; z_0 \; q_1]$

## DETERMINISTIC PUSH DOWN AUTOMATA

The DPDA can be defined as a collection of

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

where Q is a finite set of states
$\Sigma$ is finite set of input
$\Gamma$ is a finite set of stack symbols.
$\delta$ is a mapping function.
$q_0$ is initial state.
$z_0$ is initial symbol in stack
F is a finite set of final states.

Ex 1: consider $L = \{w \mid w \in (a+b)^*$ and $n_a(w) > n_b(w)\}$. The ID is given below.

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

$\delta(q_0, b, z_0) = (q_0, bz_0)$

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

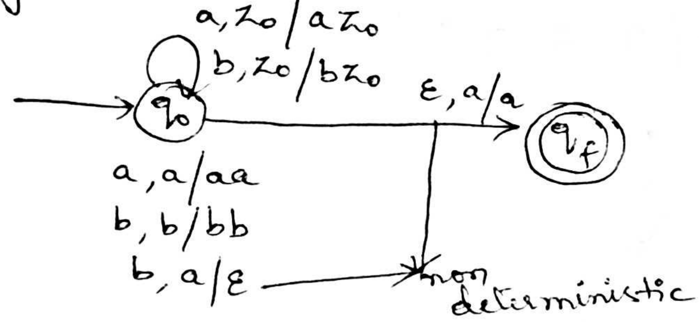
$\delta(q_0, b, b) = (q_0, bb)$

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

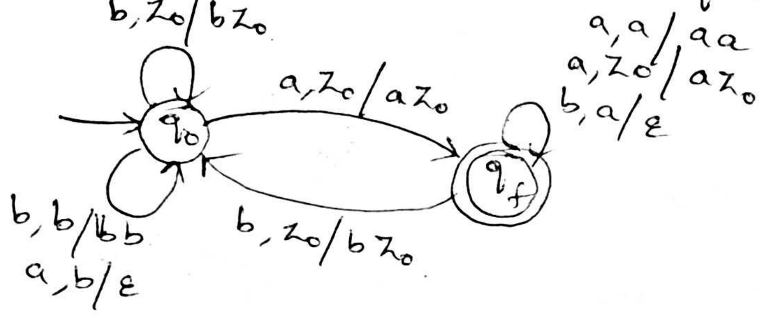
$\delta(q_0, b, a) = (q_0, \varepsilon)$

$\delta(q_0, \varepsilon, a) = (q_f, a)$ Is the PDA deterministic?

Solⁿ We will draw the transition graph for the given PDA.



We can convert it into an equivalent DPDA as -



i.e $\delta(q_0, a, Z_0) = (q_f, aZ_0)$

$\delta(q_0, b, Z_0) = (q_0, bZ_0)$

$\delta(q_0, b, b) = (q_0, bb)$

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

$\delta(q_f, a, Z_0) = (q_f, aZ_0)$

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

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

$\delta(q_f, b, Z_0) = (q_0, Z_0)$

$\delta(q_f, \varepsilon, a) = (q_f, \varepsilon)$ → ACCEPT

Ex:2: Design DPDA for $L = a^n b^n$ where $n \geq 1$.

ID can be - $\delta(q_0, a, Z_0) = (q_1, aZ_0)$

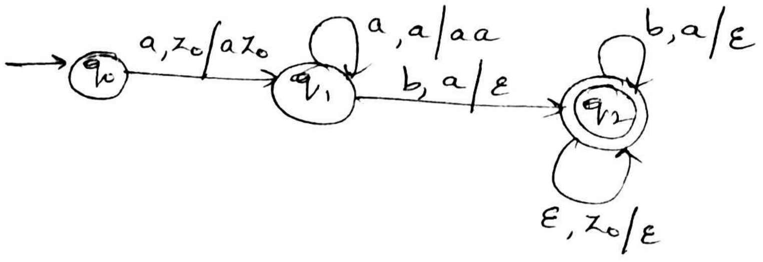
$\delta(q_1, a, a) = (q_1, aa)$

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

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

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

The transition graph will be



simulation : $\delta(q_0, aabb, Z_0)$

$\vdash (q_1, abb, aZ_0)$

$\vdash (q_1, bb, aaZ_0)$

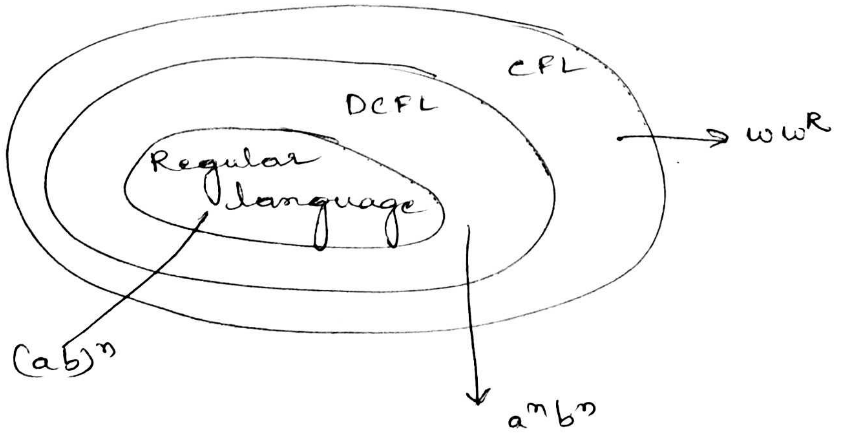
$\vdash (q_2, b, aZ_0)$

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

$\vdash (q_2, \varepsilon)$

Deterministic CFL :

A language L is a deterministic CFL (DCFL) if it is accepted by DPDA.



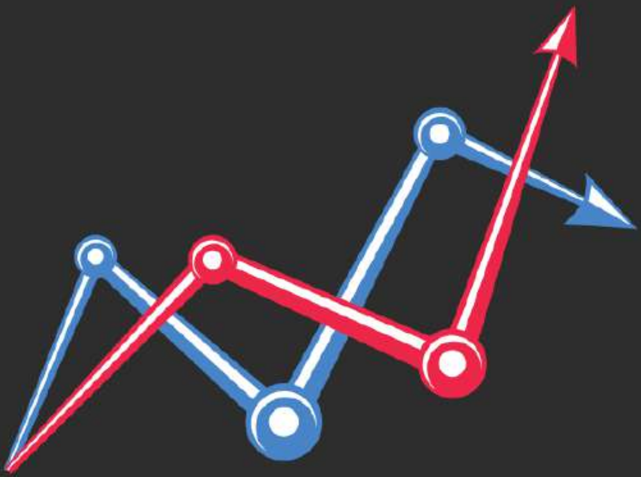
for eg:- $\{L = a^n b^n \mid n \geq 1\}$ is a DCFL.

Generally all regular languages can be accepted by DPDA.
bcz every DFA is DPDA without having stack.

MRIET

# FORMAL LANGUAGES AND AUTOMATA THEORY

# UNIT-4



JNTUH (R-18)

# UNIT - IV

## NORMAL FORMS FOR CFGs

In a CFG at the right hand of the production there are many no. of terminal or non terminal symbols in any combination.

We need to normalize such a grammar.

- That means there should be fixed no. of terminals and non terminals in the CFG.

There are two important normal Forms.

(i) chomsky's Normal Form

(ii) Greibach Normal Form.

CNF NOTATION:

Nonterminal → Nonterminal

Nonterminal → terminal.

GNF NOTATION:

Nonterminal → one terminal. any no. of non terminals.

Non terminal → terminal.

## CHOMSKY'S NORMAL FORM (CNF):

chomsky's Normal form can be defined as.

Nonterminal → Nonterminal . Nonterminal

Nonterminal → terminal.

- The given CFG should be converted in the above format, then we can say that the grammar is in CNF.
- Before converting the grammar into CNF it should be in reduced form.
- That means remove all the useless symbols, ε productions and unit productions from it.
- Thus this reduced grammar can be then converted to CNF.

## PROBLEMS ON CNF:

(1) Convert the following CFG into CNF.

S → aaaaS

S → aaaa.

Soln As we know the rule for chomsky's Normal Form is

Non-terminal → Non-terminal . Non terminal

Non-terminal → terminal.

But the CFG given is

S → aaaaS rule 1

S → aaaa. rule 2

If we add a rule

A → a which is in CNF.

Then rule 1 and rule 2 becomes.

$S \rightarrow AAAAS.$

$S \rightarrow AAAA$

Let us take

$S \rightarrow A\boxed{AAAS}$ can be replaced by $P_1$

If we define

$P_1 \rightarrow AAAS$ then the rule becomes.

$S \rightarrow AP_1$ which is in CNF.

But the new rule $P_1$ is not in CNF, so let us convert it

$P_1 \rightarrow A\boxed{AAS}$ can be replaced by $P_2$

∴ $P_1 \rightarrow AP_2$ which is in CNF

$P_2 \rightarrow AAS$ which is not in CNF

So let us convert it into CNF

$P_2 \rightarrow A\boxed{AS}$ can be replaced by $P_3$

$P_2 \rightarrow AP_3$

$P_3 \rightarrow AS.$

Now both $P_2$ and $P_3$ are in CNF.

collectively rewrite these rules.

$$\boxed{\begin{array}{l} S \rightarrow AP_1 \\ P_1 \rightarrow AP_2 \\ P_2 \rightarrow AP_3 \\ P_3 \rightarrow AS. \end{array}}$$

Now consider rule 2

$S \rightarrow AAAA$

If we break rule 2 as

$S \rightarrow \boxed{AA}\ \boxed{AA}$

$$S \rightarrow \underbrace{\boxed{AA}}_{P_4}\ \underbrace{\boxed{AA}}_{P_5}$$

The rule becomes.

$S \rightarrow P_4 P_5$ which is in CNF.

But $P_4$ and $P_5$ indicate the same rule.

So we can eliminate either of them.

Hence rule 2 becomes.

$S \rightarrow P_4 P_4$.

Finally we can collectively show the CFG converted to CNF as.

$S \rightarrow AP_1$
$P_1 \rightarrow AP_2$
$P_2 \rightarrow AP_3$
$P_3 \rightarrow AS$
$S \rightarrow P_4 P_4$
$P_4 \rightarrow AA$
$A \rightarrow a$.

(2) Convert the given CFG to CNF

$S \rightarrow aSa \mid bSb \mid a \mid b$

Soln Let us start by adding new symbols for the terminals.

$S \rightarrow ASA$
$S \rightarrow BSB$
$A \rightarrow a$
$B \rightarrow b$

Note: that although $S \rightarrow a$ and $A \rightarrow a$ are similar still we are not replacing any a by S

This is because, S is not simply giving a.

It has other productions also

We want such a terminal having only one production and that is a.

Same is with b.

Hence we have added the rules $A \to a$ and $B \to b$.

Let us take $S \to ASA$ for converting to CNF

$S \to A\boxed{SA}$

replace it by $A_1$

$S \to AA_1$
$A_1 \to SA$ } both are in CNF.

Take, $S \to B\boxed{SB}$

replace it by $A_2$

$S \to BA_2$ both are in CNF
$A_2 \to SB$.

Hence we can write the rule in CNF as

$S \to AA_1$
$A_1 \to SA$
$A \to a$
$S \to BA_2$
$A_2 \to SB$
$B \to b$.
$S \to a$
$S \to b$.

(3) Consider $a = (\{S, A\}, \{a, b\}, P, S)$

where P consists of

$S \to aAS \mid a$
$A \to SbA \mid SS \mid ba$

Soln Let us consider

$S \to aAS$.

If we put $R_1 \to a$.

then $S \rightarrow R_1 AS$

and if $R_2 \rightarrow AS$.

then $S$ becomes.

$S \rightarrow R_1 R_2$ is now in CNF

$S \rightarrow a$ is already in CNF

$A \rightarrow SbA$

Let $R_3 \rightarrow b$.

Hence $A \rightarrow SR_3A$

Add $R_4 \rightarrow R_3A$

then $A \rightarrow SR_4$

$A \rightarrow SS$ is already in CNF.

Now as we have defined $R_1 \rightarrow a$ and $R_3 \rightarrow b$.

$A \rightarrow R_1 R_3$

Finally, we can write

$S \rightarrow R_1 R_2$ $\qquad R_1 \rightarrow a$

$S \rightarrow a$ $\qquad R_2 \rightarrow AS$

$A \rightarrow S.R_4.$ $\qquad R_3 \rightarrow b.$

$A \rightarrow SS.$

$A \rightarrow R_1 R_3.$

(4) Convert the given CFG to CNF.

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

where $V = \{S, A, B\}$

$T = \{a, b\}$

P consists of

$S \rightarrow aB$

$S \rightarrow bA$

$A \rightarrow a$

$A \rightarrow aS$

$A \rightarrow bAA$

$B \rightarrow b$

$B \rightarrow aS$

$B \rightarrow aBB$

Sol<sup></sup>

Soln Let us start the first rule.

$R_1 \rightarrow a$

$\boxed{S \rightarrow R_1 B}$ is in CNF for $S \rightarrow aB$

$R_2 \rightarrow b$.

$\boxed{S \rightarrow R_2 A}$ is in CNF for $S \rightarrow bA$

$A \rightarrow a$ – is in CNF

$A \rightarrow aS$

$A \rightarrow R_1 S$ – is in CNF

Now.

$A \rightarrow bAA$

$A \rightarrow R_2 \boxed{AA}$ replace by $R_3$

$\boxed{A \rightarrow R_2 R_3}$ is in CNF.

$R_3 \rightarrow AA$ is in CNF

$B \rightarrow b$

$B \rightarrow aS$ can be written as

$\boxed{B \rightarrow R_1 S}$ which is in CNF

Now,

$B \rightarrow aBB$ can be written as

$B \rightarrow R_1 \boxed{BB}$ can be written as.

$R_4 \rightarrow BB$.

$B \rightarrow R_1 R_4$

Finally, we can write,

$S \rightarrow R_1 B$

$S \rightarrow R_2 A$

$A \rightarrow R_1 S$

$A \rightarrow R_2 R_3$

$A \rightarrow a$

$B \rightarrow b$

$B \rightarrow R_1 S$

$B \rightarrow R_1 R_4$

$R_1 \rightarrow a$

$R_2 \rightarrow b$.

$R_3 \rightarrow AA$

$R_4 \rightarrow BB$.

is a chomsky Normal Form.

(5) Convert the following CFG to CNF

$S \rightarrow ABA$

$A \rightarrow aA \mid \epsilon$

$B \rightarrow bB \mid \epsilon$

Sol$^n$ In chomsky's Normal form the $\epsilon$ production is not allowed.

So first we will eliminate $\epsilon$ productions.

$A \rightarrow \epsilon$ and $B \rightarrow \epsilon$.

If we put $\epsilon$ instead of A and B.

We can get

$S \rightarrow ABA \mid AB \mid BA \mid AA \mid A \mid B$

III$^{rly}$ $A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$.

Now $S \rightarrow A$ and $S \rightarrow B$ is unit production

getting introduced into the grammar after removal of $\epsilon$ production.

- So we will remove unit production also

$S \rightarrow ABA \mid AB \mid BA \mid AA \mid aA \mid a \mid bB \mid b$.

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$.

Now let us convert this grammar to chomsky's normal form.

Let $S \rightarrow A\boxed{BA}$

$R_1 \rightarrow BA$

then $\boxed{S \rightarrow AR_1}$

Now $\boxed{S \rightarrow AB \text{ , } S \rightarrow BA \text{, } S \rightarrow AA}$ already in CNF

Let $S \rightarrow aA$

we will define $R_2 \rightarrow a$

then $\boxed{S \rightarrow R_2 A}$

$\boxed{S \rightarrow a}$ already in CNF.

$S \rightarrow bB$

$R_3 \rightarrow b$.

then $\boxed{S \rightarrow R_3 B}$

$\boxed{S \rightarrow b}$ already in CNF.

Now consider the rules,

$A \rightarrow aA \mid a$

$A \rightarrow aA$

$R_2 \rightarrow a$

$\boxed{A \rightarrow R_2 A}$

$\boxed{A \rightarrow a}$ already in CNF.

Now consider the rules,

$B \rightarrow bB \mid b.$

$B \rightarrow bB$

$R_3 \rightarrow b$

$\therefore \boxed{B \rightarrow R_3 B}$

$\boxed{B \rightarrow b}$ already in CNF.

Collectively we can write,

$S \rightarrow AR_1 \mid AB \mid BA \mid AA \mid R_2 A \mid R_3 B \mid a \mid b.$

$A \rightarrow R_2 A \mid a$

$B \rightarrow R_3 B \mid b.$

$R_1 \rightarrow BA$ $\quad$ $R_3 \rightarrow b$

$R_2 \rightarrow a$

6 convert the following grammar to chomsky Normal form.

$S \to a \mid b \mid aSS.$

solⁿ The chomsky's Normal form is

$NT \to NT.NT$

$NT \to$ terminal.

$\boxed{S \to a}$ already in CNF

$\boxed{S \to b}$ already in CNF

$S \to a\boxed{SS}$

$\boxed{R_1 \to SS}$

then $S \to aR_1$

$\boxed{R_2 \to a}$

then $\boxed{S \to R_2 R_1}$

The CNF of given grammar is

$S \to a \mid b \mid R_2R_1$

$R_1 \to SS$

$R_2 \to a$

7 convert the following grammar to chomsky's Normal form.

$S \to \sim S \mid [S \supset S] \mid p \mid q$

solⁿ chomsky's Normal form is

nonterminal → nonterminal. nonterminal.

nonterminal → terminal.

Consider given grammar rule by rule

$S \to \sim S.$

Let $A \to \sim$

then $\boxed{S \to AS}$

IIIrly $S \to [S \supset S]$

Can be written as

$S \to GF$

$B \to [$

$C \to \supset$

$D \to ]$

$E \to SC$

$F \to SD$

$G \to BE$

And $S \to p$

$S \to q$ already in CNF.

Hence the complete grammar is

| | | | |
|---|---|---|---|
| $S \to AS$ | $B \to [$ | $E \to SC$ | $S \to P$ |
| $A \to \sim$ | $C \to \supset$ | $F \to SD$ | $S \to q$ |
| $S \to GF$ | $D \to ]$ | $G \to BE$ | |

$S \to \underset{B}{[}\overset{E}{\boxed{S \underset{C}{\supset}}}\ \overset{F}{\boxed{S \underset{D}{]}}}$

$B \to [$

$C \to \supset$

$D \to ]$

$E \to SC$

$F \to SD$

$G \to BE$

BEF

$S$

$GF \ (G \to BE)$

$BEF \ (B \to [)$

$[EF \ (E \to SC)$

$[SCF \ (C \to \supset)$

$[S \supset F \ (F \to SD)$

$[S \supset SD \ (D \to ])$

$[S \supset S]$

⑧ Convert the following grammar into CNF.

$S \to aAD$

$A \to aB \mid bAB$

$B \to b$

$D \to d.$

Sol^n Consider $S \to a\boxed{AD}$

$R_1 \to AD$ ✓

then $S \to aR_1$

$R_2 \to a$ ✓

then $S \to R_2R_1$ ✓

**Rule 2 :**

$A \rightarrow aB \mid bAB$.

$A \rightarrow \underline{a}B$

$R_2 \rightarrow a$ ✓

$\boxed{A \rightarrow R_2B}$ is in CNF.

$A \rightarrow b\boxed{AB}$

$R_3 \rightarrow AB$ ✓

then $A \rightarrow b.R_3$

$R_4 \rightarrow b$ ✓

$\boxed{A \rightarrow R_3 R_4}$ is in CNF

**Rule 3:**

$B \rightarrow b$ is already in CNF

**Rule 4:** $D \rightarrow d$ is already in CNF

Collectively, the CNF is

$S \rightarrow R_2R_1$

$R_1 \rightarrow AD$

$R_2 \rightarrow a$

$R_3 \rightarrow AB$

$A \rightarrow R_2B$

$R_4 \rightarrow b$

$A \rightarrow R_3R_4$

$B \rightarrow b$.

$D \rightarrow d$.

# GREIBACH NORMAL FORM (GNF)

The rule for GNF is

Non terminal → one terminal. Any number of non terminals.

Eg:-

$S \to aA$ is in GNF

$S \to a$ is in GNF

But $S \to AA$ is not in GNF

$S \to Aa$ is not in GNF.

Procedure for converting CFG into GNF:

Lemma 1: Let $G = (V, T, P, S)$ be a given CFG.

And $A \to a_1 B$ is a production

$B \to \beta_1 | \beta_2 | \beta_3 | \beta_4 \cdots \beta_n$.

Then we can obtain $G_1 = (V, T, P_1, S)$ where

$P_1$ Consists of

$A \to a_1\beta_1 | a_1\beta_2 | a_1\beta_3 | a_1\beta_4 \cdots a_1\beta_n$

That means replacing B by its productions.

Then $L(G)$ is equivalent to $L(G_1)$

For eg:-

$S \to Aa$

$A \to aA | bA | aAS$

Then we can write S as

$S \to aAa | bAa | aASa$

$S \to aAR1 | bAR1 | aASR1$ and $R1 \to a$

Lemma 2 :

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

If P consists of

$A \rightarrow Aa_1 | Aa_2 | Aa_3 ---- | Aa_n | \beta_1 | \beta_2 | \beta_3 | --- \beta_K$

Such that $\beta_i$ do not start with A then eqvt grammar in GNF can be

$A \rightarrow \beta_1 | \beta_2 | \beta_3 --- \beta_K$ where

$A \rightarrow \beta_1 X | \beta_2 X | \beta_3 X --- \beta_K X.$

$X \rightarrow a_1 | a_2 | a_3 ---- | a_n$

$X \rightarrow a_1 X | a_2 X | a_3 X | ---- | a_n X.$

For eg:- $\underline{A} \rightarrow \underline{A}a | Ab | a | b.$

Then we can convert this grammar to eqvt GNF as.

⇒ Here $\beta_1 = a$ ; $\beta_2 = b$

$a_1 = a$   $a_2 = b$

Then $A \rightarrow a | b$

$A \rightarrow aZ | bZ$

$Z \rightarrow A$

$Z \rightarrow aZ | bZ.$

PROBLEMS ON GNF:

(1) Convert the given CFG into GNF.

$S \rightarrow ABA$
$A \rightarrow aA \mid \varepsilon$
$B \rightarrow bB \mid \varepsilon$.

Soln Before converting the CFG to its normal form we usually reduce the grammar i.e we first eliminate ε prodn, unit prodn and useless symbols.

Let us simplify the given CFG,

$S \rightarrow ABA \mid AB \mid BA \mid AA \mid A \mid B$.
$A \rightarrow aA \mid a$
$B \rightarrow bB \mid b$.

Now remove unit prodns.

$S \rightarrow aABA \mid aAB \mid aB \mid bA \mid aBA \mid bBA \mid aAA \mid aA \mid a \mid bB \mid b$
$A \rightarrow aA \mid b$.
$B \rightarrow bB \mid b$.

Which is in GNF.

(2) Convert given CFG to GNF

Where $V = \{S, A\}$, $T = \{0, 1\}$ and P is $S \rightarrow AA \mid 0$
$A \rightarrow SS \mid 1$.

Soln 1 step:- Rename the given Non Terminals.

S as $A_1$
A as $A_2$

Now the prodns are

$A_1 \rightarrow A_2 A_2 \mid 0$
$A_2 \rightarrow A_1 A_1 \mid 1$.

Step 2:- check for $i \leq j$.

$A_1 \to A_2A_2 | 0$ (satisfies the cond$^n$)

$A_2 \to A_1A_1 | 1$ (does not satisfy the cond$^n$)

$A_2 \to \boxed{A_1}A_1 | 1.$ (By using lemma 2)

$A_2 \to A_2\underbrace{A_2A_1}_{\alpha_1} | \underbrace{0A_1}_{\beta_1} | \underbrace{1}_{\beta_2}$

$A_2 \to 0A_1 | 1$

$A_2 \to 0A_1Z | 1Z$

$Z \to A_2A_1$

$Z \to A_2A_1Z$

Now $A_2$ is in GNF.

Now modify $A_1$ prod$^n$s.

$A_1 \to A_2A_2 | 0.$

$$\boxed{A_1 \to 0A_1A_2 | 1A_2 | 0A_1ZA_2 | 1ZA_2 | 0}$$

Now $A_1$ is in GNF.

Let us modify Z.

$Z \to \boxed{A_2}A_1 | A_2A_1Z$

$Z \to 0A_1A_1 | 1A_1 | 0A_1ZA_1 | 1ZA_1 |$

~~$Z \to 0A_1A_1Z_1 | 1ZA_1Z | 0A_2$~~

$Z \to 0A_1A_1Z | 1A_1Z | 0A_1ZA_1Z | 1ZA_1Z$

collectively we can write

$A_1 \to 0A_1A_2 | 1A_2 | 0A_1ZA_2 | 1ZA_2 | 0.$

$A_2 \to 0A_1 | 1 | 0A_1Z | 1Z.$

$Z \to 0A_1A_1 | 1A_1 | 0A_1ZA_1 | 1ZA_1 | 0A_1A_1Z | 1A_1Z | 0A_1ZA_1Z |$
$1ZA_1Z.$

(3) Convert the given CFG to GNF.

$S \longrightarrow CA$
$A \longrightarrow a$
$C \longrightarrow aB \mid b.$

Solⁿ A, C are in GNF.

But S is not in GNF.

Let us apply lemma 1 for S.

$S \longrightarrow \underline{C}A$
$S \longrightarrow aBA \mid bA.$

Thus prodⁿˢ in GNF are

$S \longrightarrow aBA \mid bA$
$A \longrightarrow a.$
$C \longrightarrow aB \mid b.$

(4) Convert the following CFG to GNF.

$A_1 \longrightarrow A_2A_3$
$A_2 \longrightarrow A_3A_1 \mid b.$
$A_3 \longrightarrow A_1A_2 \mid a$

Solⁿ check for $i<j$.
So rule $A_3$ does not satisfy $i \leq j$ condⁿ.

$A_3 \longrightarrow A_1A_2 \mid a$

If we apply lemma 1 in rule $A_3$

$A_3 \longrightarrow A_2A_3A_2 \mid$ ~~$aA_2$~~ $\mid a.$

$A_3 \longrightarrow \underline{A_2}A_3A_2 \mid a.$

Substitute $A_2 \longrightarrow A_3A_1 \mid b$ in rule $A_3$.

$A_3 \longrightarrow A_3A_1A_3A_2 \mid bA_3A_2 \mid a.$

Now let us apply lemma 2 on this production.

$$A_3 \rightarrow A_3 \underbrace{A_1 A_3 A_2}_{\alpha_1} \mid \underbrace{b A_3 A_2}_{\beta_1} \mid \underbrace{a}_{\beta_2}$$

The prod$^{n}$s are

$$\boxed{\begin{array}{l} A_3 \rightarrow b A_3 A_2 \mid a \\ A_3 \rightarrow b A_3 A_2 Z \mid a Z \end{array}} \text{ — } ③$$

$Z \rightarrow A_1 A_3 A_2$

$Z \rightarrow A_1 A_3 A_2 Z$

Now if we observe $A_3$ productions are in GNF.

Now consider $A_2$ productions.

$$A_2 \rightarrow \underbrace{A_3}_{} A_1 \mid b$$

↳ Replace it by Eq$^{n}$ ③

Then we get

$$\boxed{A_2 \rightarrow b A_3 A_2 A_1 \mid a A_1 \mid b A_3 A_2 Z A_1 \mid a Z A_1 \mid b} \text{ — } ④$$

Now $A_2$ is in GNF.

Now consider $A_1$ productions.

$$A_1 \rightarrow A_2 A_3$$

↳ replace it by Eq$^{n}$ ④

$$\boxed{A_1 \rightarrow b A_3 A_2 A_1 A_3 \mid a A_1 A_3 \mid b A_3 A_2 Z A_1 A_3 \mid a Z A_1 A_3 \mid b A_3} \text{ — } ⑤$$

It is in GNF.

Now remaining productions are $Z$.

$$Z \rightarrow A_1 A_3 A_2$$

↳ replace it by Eq$^{n}$ ⑤

$$Z \rightarrow b A_3 A_2 A_1 A_3 A_3 A_2 \mid a A_1 A_3 A_3 A_2 \mid b A_3 A_2 Z A_1 A_3 A_3 A_2 \mid a Z A_1 A_3 A_3 A_2 \mid b A_3 A_3 A_2$$

Also

$z \to \underline{A_1}\underbrace{A_3 A_2 Z}_{same}$

$z \to b A_3 A_2 A_1 A_3 A_3 A_2 z \mid a A_1 A_3 A_3 A_2 z \mid b A_3 A_2 z A_1 A_3 A_3 A_2 Z \mid$

$a z A_1 A_3 A_3 A_2 Z \mid b A_3 A_3 A_2 z$

Thus z is also in GNF.

(5) convert the following grammar to GNF.

$S \to SS$

$S \to 0S1 \mid 01$

$\underline{\underline{Sol^n}}$ The GNF is

N.T → terminal. set of Nonterminals

N.T → terminal.

$S \to SS$ $\quad S \to 01$

$S \to 0SA$ $\quad S \to \underline{0A}$

$A \to 1$

$S \to \underline{SS}$

$S \to 0SAS \mid 0AS$

Thus the GNF rules are

$S \to 0SAS \mid 0AS \mid 0SA \mid 0A$

PUMPING LEMMA FOR CONTEXT FREE LANGUAGES

- The pumping lemma for regular sets states that every sufficiently long string in a regular set contains a short substring that can be pumped.
- In other words if a long string is given and if we push or pump any number of substrings in any number of times then we always get a regular set.

# PUMPING LEMMA FOR CONTEXT FREE LANGUAGES

According to ~~The~~ pumping lemma for ~~regular~~ CFL's there are always two short substrings which are close to each other and these both the substrings can be repeated as many times as required.

Lemma :- Let L be any context free language, then there is a constant n, which depends only upon L, such that there exist a string $w \in L$ and $|w| \geq n$ where $w = pqrst$ such that

1. $|qs| \geq 1$
2. $|qrs| \leq n$ and
3. For all $i \geq 0$ $pq^{i}rs^{i}t$ is in L.

Proof :- This pumping lemma states that if there is a language L which is without unit productions and without a null production and there exist w where $w \in L$. The string w can be derived by a CFG. G.

- G be a grammar which is in CNF.

- The grammar G generates language L.
- for the string w, we can obtain a parse tree which derives the string w.

  Then if the length of the path to w is less than or equal to i then the length of the word w is less than or equal to $2^{i-1}$.

- We can prove this by induction step.

Basis: If i = 1.

Let G contain the rule $S \rightarrow a$ where length of the derived string is 1 i.e $i = 1$.

· Now according to the rule the word length should be $\leq 2^{i-1}$.

i.e $2^0 = 1$.

· observe that we have a word 'a' is of length 1.

Also observe that the grammar G is in chomsky's Normal form.

This language is regular since $|w| = |pqrst| = 1$.

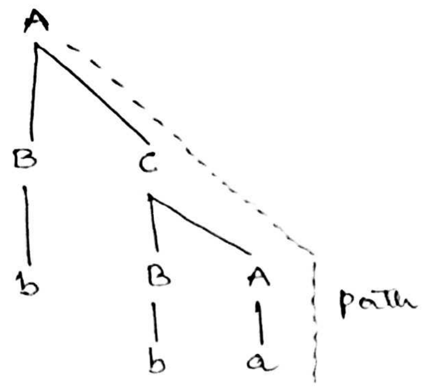
Induction step: Let w be a string which is derived by grammar G. Let k be a variable such that $n = 2^k$, $|w| \geq n$ then $|w| > 2^{k-1}$ while deriving w string we may get some non-terminals of CFG. G can be repeated for any number of times and will give the string w.

· If we pump the substrings to w such that the path length of this newly formed string w' (w+ pumped string = w') is i and the word length of w' is $\leq 2^{i-1}$ then the grammar G deriving w' is called a regular grammar.

The necessary cond^n is that grammar G is in chomsky's Normal form.

Let us consider a grammar

$$G = (\{A, B, C\}, \{a\}, \{ A \rightarrow BC$$
$$B \rightarrow BA$$
$$C \rightarrow BA$$
$$A \rightarrow a$$
$$B \rightarrow b\}, A).$$



Thus.

$A \overset{*}{\Rightarrow} bba = W$.

ie path length $i = 3$.

$|w| \leq 2^{i-1}$ ie $3 \leq 2^2$

If we pump a substring into w which satisfies the condition as $i \leq |w| \leq 2^{i-1} \leq n$ the grammar producing string w is a regular grammar.

(1) prove whether the given language $L = \{ s s^T | s \in (a,b)^* \}$ is context free or not.

Soln If we consider $W = pqrst$ be the string Which $\in L$ and L is a language of palindrome

Let us say L is a CFL

According to lemma

1. $|qs| \geq 1$
2. $|~~p~~rs| \leq n$.

If we pump some substring of length i

We should get

$W = pq^i rs^i t \in L$ where $i \geq 0$.

For eg:- if we take abaaba

Let us group the i/p.

$w = (ab)(a)(a)(b)(a)$

If we modify w by $pq^{i+1}rs^{i+1}t$ it becomes.

$w' = (ab)(a)(a)(a)(b)(b)(a)$ and it is not palindrome

then $w' \notin L$.

Hence $L = \{ss^T \mid s \in (a,b)^*\}$ is not CF language.

② show that $L = \{a^n b^n c^n \mid n \geq 0\}$ is not a CFL.

Sol$^n$ Let us assume that

$L = a^n b^n c^n$ is a CFL

Let w be any string such that $w \in L$

Let

$w = pqrst$

let $|qs| \geq 1$ and $|qrs| \leq n$.

Now let us consider

$w = pq^i rs^i t$

ie we have additional occurences of q and s.

Consider various cases.

Case 1: consider $i=0$ then,

$W = pq^i rs^i t$

$W = \underbrace{aa}_{p}\underbrace{b}_{q}\underbrace{b}_{r}\underbrace{c}_{s}\underbrace{c}_{t}$

If $i=0$ then $q^0$ and $s^0$. Then p and q are absent

then $W = prt$.

$= aabc \notin L$.

Hence our assumption of L being CFG is wrong.

Case 2: Consider $i = 2$ then,

$$W = pq^i r s^i t$$
$$= pqqrsst$$

Consider $$W = \underbrace{aa}_{p}\underbrace{b}_{q}\underbrace{b}_{r}\underbrace{c}_{s}\underbrace{c}_{t}$$

Now with $i=2$, we get

$$W = \underbrace{aa}_{p}\underbrace{b}_{q}\underbrace{b}_{q}\underbrace{b}_{r}\underbrace{c}_{s}\underbrace{c}_{s}\underbrace{c}_{t}$$

$$= a^2 b^3 c^3 \notin L.$$

Thus our assumption of L being CFG is wrong.

This proves that the given language L is not context free.

# CLOSURE PROPERTIES OF CONTEXT FREE LANGUAGES:

Regular languages are closed under Union, concatination and kleen closure. The context free languages are closed under some operation means after performing that particular operati on these CFLs the resultant language is context free languag

The properties are given below.

(1) The context free languages are closed under Union.

Theorem: If $L_1$ and $L_2$ are context free languages then $L = L_1 \cup L_2$ is also context free. That is CFLs are closed under Union.

Proof: We will consider two languages $L_1$ and $L_2$ which are context free languages.

We can give these languages using context free gramma $G_1$ and $G_2$ such that $G_1 \in L_1$ and $G_2 \in L_2$. The $G_1$ can be given as $G_1 = \{V_1, \Sigma, P_1, S_1\}$ where $P_1$ can be given as

$P_1 = \{$

$S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S_1 B_1 \mid \varepsilon$

$A_1 \rightarrow a$

$B_1 \rightarrow b$

$\}$

Here $V_1 = \{S_1, A_1, B_1\}$ and $S_1$ is a start symbol

Illrly, we can write $G_2 = \{V_2, \Sigma, P_2, S_2\}$

$S_2$ is a start symbol.

$P_2$ can be given as.

$P_2 = \{$

$S_2 \rightarrow aA_2A_2 \mid bB_2B_2$

$A_2 \rightarrow b.$

$B_2 \rightarrow a$

$\}.$

Now $L = L_1 \cup L_2$ gives $G \in L$. This can be written as.

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

$V = \{S_1, A_1, B_1, S_2, A_2, B_2\}$

$P = \{P_1 \cup P_2\}$

S is a start symbol.

$P = \{ S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S_1 B_1 \mid \varepsilon$

$A_1 \rightarrow a$

$B_1 \rightarrow b$

$S_2 \rightarrow aA_2A_2 \mid bB_2B_2$

$A_2 \rightarrow b.$

$B_2 \rightarrow a$

$\}$

Thus grammar G is a context free grammar which produces languages L which is context free Language.

(2) Context free languages are closed under concatenation.

Theorem: If $L_1$ and $L_2$ are two context free languages then $L_1L_2$ is CFG

proof: Let $L_1$ is a context free language which can be represented by a context free grammar $G_1$ such that $G_1 \in L_1$ and

$G_1 = \{V_1, \Sigma, P_1, S_1\}$

$V_1 = \{S_1, A_1, B_1\}$

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

$S_1$ is a start symbol, and $P_1$ is a set of production rules,

$P_1 = \{S_1 \longrightarrow A_1 S_1 A_1 \mid B_1 S_1 B_1 \mid \varepsilon$

$A_1 \longrightarrow a$

$B_1 \longrightarrow b.$

$\}$

IIIrly $L_2$ is a context free language which can be represented by a context free grammar $G_2$ such that $G_2 \in L_2$ and

$G_2 = \{V_2, \Sigma, P_2, S_2\}$

$V_2 = \{S_2, A_2, B_2\}$

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

$S_2$ is a start symbol and $P_2$ is a set of production rules,

$P_2 = \{S_2 \longrightarrow aA_2A_2 \mid bB_2B_2$

$A_2 \longrightarrow b$

$B_2 \longrightarrow a$

$\}$

Now $L = L_1L_2$ can be obtained by $G$ such that $G = G_1G_2$.

Therefore $G = (V, \Sigma, P, S)$

$V = \{S, S_1, A_1, B_1, S_2, A_2, B_2\}$

where $S$ is a start symbol.

The production rules P can be given as

$P = \{ S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S_1 B_1 \mid \varepsilon$

$A_1 \rightarrow a$

$B_1 \rightarrow b$

$S_2 \rightarrow a A_2 A_2 \mid b B_2 B_2$

$A_2 \rightarrow b$

$B_2 \rightarrow a$

$\}$

As grammar G is context free grammar the language L produced by G is also context free language. Hence context free languages are closed under concatenation.

(3). Context free languages are closed under kleen closure.

<u>Theorem</u>: If $L_1$ is a context free language then $L_1^*$ is also context free.

<u>proof</u>: Let $L_1$ can be context free language represented by $G_1$ such that $G_1 \rightarrow \varepsilon L_1$

The CFG $G_1$ can be given as

$G_1 = \{ V_1, \Sigma, P_1, S_1 \}$ where $S_1$ is a start symbol.

$P_1 = \{ S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S_1 B_1 \mid \varepsilon$

$A_1 \rightarrow a$

$B_1 \rightarrow b.$

$\}$

Now $L = L_1^*$ can be represented by a grammar G such that $G = (V, \Sigma, P, S)$

$V = \{S, S_1, A_1, B_1\}$

$P = S \rightarrow S_1 S \mid \varepsilon$

$S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S_1 B_1$

$A_1 \rightarrow a$

$B_1 \rightarrow b$

Thus grammar G is a context free grammar and language L produced by G is also context free language. Hence context free languages are closed under kleen closure.

(4) Context free languages are not closed under intersection.

<u>Theorem</u>: If $L_1$ and $L_2$ are two CFLs then $L = L_1 \cap L_2$ may be CFL or may not be CFL. That means L is not closed under intersection.

<u>proof</u>: Let $L_1 = \{0^n 1^n 2^i / n \geq 1, i \geq 1\}$

$L_2 = \{0^n 1^n 2^n / n \geq i \geq 1\}$

The grammar for $L_1$ is

$S \rightarrow AB$

$A \rightarrow 0A1 \mid 01$

$B \rightarrow 2B \mid 2$

111ly $L_2$ can be represented by grammar.

$S \rightarrow AB$

$A \rightarrow 0A \mid 0$

$B \rightarrow 1B2 \mid 12$

Now if we try to obtain

$L = L_1 \cap L_2$ then we get sometimes context free languages and some times non context free languages.

Thus we can say that CFLs are not closed under intersection.

(5). context free languages are not closed under complement.

Theorem: If $L_1$ is a CFL then $L_1'$ may or may not be CFL.

proof: Let $L_1$ and $L_2$ are two CFLs. We will assume that complement of a context free language is a CFL itself. Hence $L_1'$ and $L_2'$ both are CFLs.

- We can also state that $L_1' \cup L_2'$ is context free (since CFLs are closed under union).

But $(L_1' \cup L_2') = L_1 \cap L_2$ i.e $L = L_1 \cap L_2$ may or may not be CFL.

The $L_1$ and $L_2$ are arbitrary CFLs, there may exist. $L_1'$ and $L_2'$ which are not CFL. Hence complement of certain language may be context free or may not be.

Therefore we can say that CFL is not closed under complement operation.

## Decision properties of CFL's :

The major tests we are able to make are

(1) whether the language is empty and

(2) whether a given string is in the language.

Complexity of Converting Among CFG's and PDA's.

Let us consider the complexity of converting from one representation to another.

- We shall let n be the length of the entire representation of a PDA or CFG. Using this parameter as the repntn of the size of the grammar some algorithms have a running time that could be described more precisely in terms of more specific parameters such as the number of variables of a grammar or the sum of the lengths of the stack strings that appear in the transition function of PDA.

- However the total length measure is sufficient to distinguish the most important issues: is an algorithm linear in the length (i.e does it take little more time than it takes to read its input) is it exponential in length (i.e you can perform the conversion only for rather small examples) or is it some nonlinear polynomial (i.e you can run the algorithm even for large examples).

There are several conversions we have seen so far that are linear in the size of the i/p. Since They take linear time, the representation that they produce as output is not only produced quickly, but it is of size Comparable to the input size

These conversions are :

1) Converting a CFG to PDA

2) Converting a PDA that accepts by final state to a PDA that accepts by empty stack.

(3) Converting a PDA that accepts by empty stack to a PDA that accepts by final state.

Running Time of Conversion to chomsky Normal Form:

- Decision algorithms may ~~first~~ depend on first putting a CFG into chomsky Normal Form, We should also look at the running time of various algorithms that We used to convert an arbitrary grammar to a CNF grammar.

- Most of the steps preserve, up to a constant factor, the length of the grammar's description ; that is starting with a grammar of length $n$ they produce another grammar of length $O(n)$.

List of observations :

(1) Using the proper algorithm, detecting the reachable and generating symbols of a grammar can be done in $O(n)$ time. Eliminating the resulting useless symbols takes $O(n)$ time and does not increase the size of the grammar.

(2) Constructing the unit pairs and eliminating unit productions, ~~as in~~ it takes $O(n^2)$ time and the resulting grammar has length $O(n^2)$.

(3) The replacement of terminals by variables in production bodies, takes $O(n)$ time and results in a grammar whose length is $O(n)$.

(4) The breaking of production bodies of length 3 or more into bodies of length 2, takes $O(n)$ time and results in a grammar of length $O(n)$.

(5). For eliminating $\varepsilon$ productions - if we have a production body of length $k$, we could construct from that one production $2^k - 1$ productions for the new grammar. Since $k$ could be proportional to $n$, this part of the construction could take $O(2^n)$ time and result in a grammar whose length is $O(2^n)$.

# Introduction to Turing Machines:

## Problems that computers Cannot Solve:

the problem: whether the first thing a C program prints is hello, world.

- We will give the intuition behind the formal ~~&~~ proof.

C program that prints "Hello, world"

```
main()
{
  print (" hello, world \n");
}
```

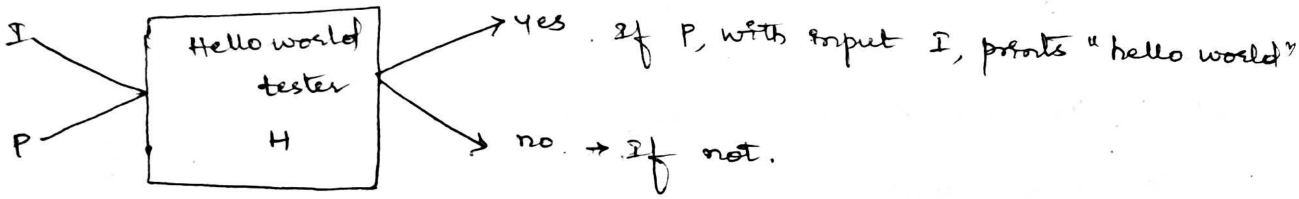
Define hello world problem to be: determine whether a given C program, with a given input, prints hello, world as the first 12 characters that it prints.

The ~~program~~ problem described alternatively using symbols: Is there a program H that could examine any program P and input I for P and tell whether P run with its input I, would print hello, world!

- The answer is undecidable! That is, there exists no such program H. We will prove this by contradiction

Hypothetical "Hello world" tester

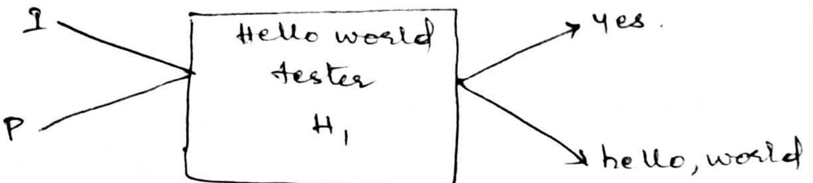
(i) Assume H exists in the following form



(ii) transform H to another form $H_2$ in simple ways which can be done by c programs.

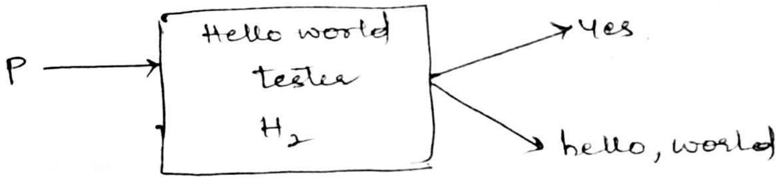
(iii) prove $H_2$ not existing. so H also not existing.

→ Transform H to $H_1$ in the following way



(print 'hello, world' instead of 'no')

→ transform $H_1$ to $H_2$ in the following way

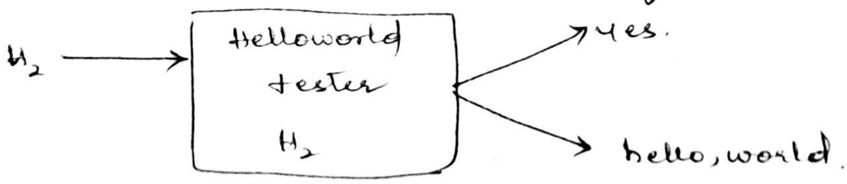


Use P both as input and program.

→ The function of $H_2$ is : given any program P as input, if P prints hello world as first output, then $H_2$ makes output yes; If P does not prints Hello world as first output, then $H_2$ prints hello world.

→ prove $H_2$ does not exist

- Let P for $H_2$ be $H_2$ itself, as follows:



($H_2$ takes $H_2$ as input to itself)

(1) If the box $H_2$, given itself as input, makes output yes, then it means that

the input $H_2$, given itself as input, prints hello, world as first o/p.

But this is contradictory because we just suppose that $H_2$ given itself as input, makes output yes.

This contradiction means the other alternative must be true.

∴ We conclude that the assumption that $H_2$ exists is wrong. by the principle of contradiction.

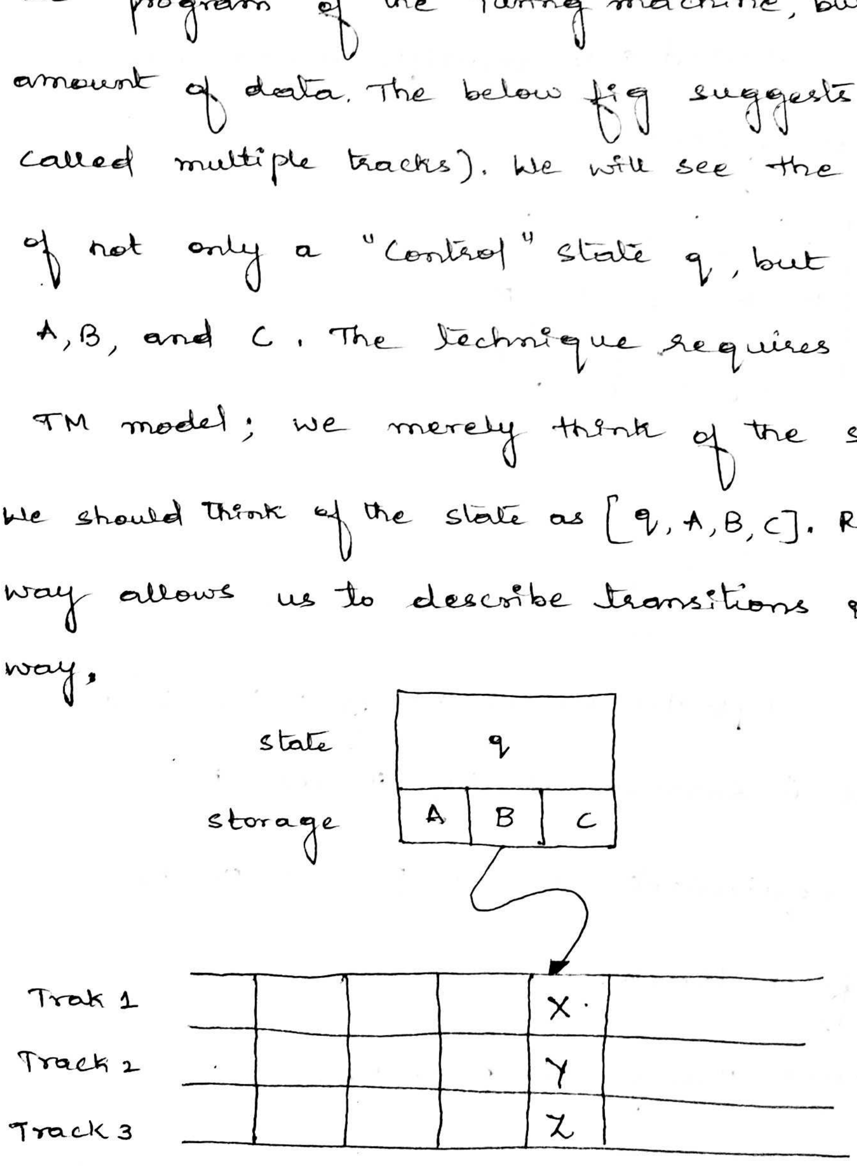
## Programming Techniques for Turing Machines:

TM is exactly as powerful as a conventional computer.

(i) Storage in the state:

We can use the finite control not only to represent a position in the "program" of the Turing machine, but to hold a finite amount of data. The below fig suggests this technique (also called multiple tracks). We will see the finite control consisting of not only a "Control" state q, but three data elements A, B, and C. The technique requires no extension to the TM model; we merely think of the state as a tuple.

We should think of the state as [q, A, B, C]. Regarding states this way allows us to describe transitions in a more systematic way.

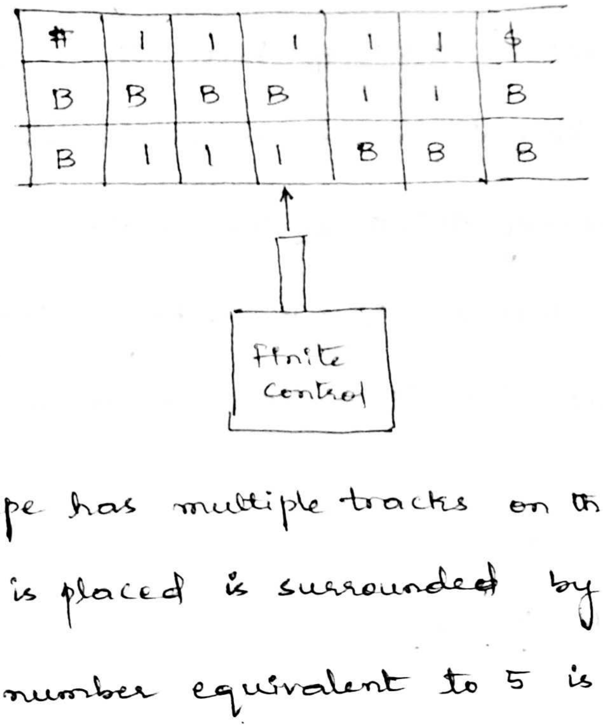


## (ii) Multiple Tracks:

Another useful "trick" is to think of the tape of a TM as composed of several tracks. Each track can hold one symbol, and the tape alphabet of the TM consists of tuples, with one component for each "track". Thus for instance, the cell scanned by the tape head contains the symbol [x, y, z]. Like the technique of storage in the finite control, using multiple tracks does not extend what the TM can do.

## (iii) [illegible]

- If the input tape is divided into multiple tracks then the input tape will be as follows -



|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| # | 1 | 1 | 1 | 1 | 1 | $ |
| B | B | B | B | 1 | 1 | B |
| B | 1 | 1 | 1 | B | B | B |

- The input tape has multiple tracks on the first track the input which is placed is surrounded by # and $.
- The unary number equivalent to 5 is placed on the input tape, on the first track.
- on the second track unary 2 is placed.

- If we construct a TM which subtracts 2 from 5 we get the answer on the third track and that is 3, in unary form. Thus this TM is for subtracting two unary numbers with the help of multiple tracks.

(iii) Subroutine :

In the high level languages use of subroutines built the modularity in the program development process. The same type of concept can be introduced in construction of TM.

We can write the subroutines as a Turing machine.

Eg:- construct a TM for the subroutine

$f(a,b) = a * b$

where a and b are unary numbers.

Soln: The unary number are represented by 1s.

That if $a=2$ and $b=3$ then

$$\underbrace{\boxed{1}\boxed{1}}_{a}\boxed{\$}\underbrace{\boxed{1}\boxed{1}\boxed{1}}_{b}\;\cdots\cdots$$

For this subroutine we will perform a copy operation. That is marking first 1 of a and copying it after Δ at b number of times, similarly marking second 1 of a and copying it at the rightmost end for b number of times. This makes the multiplication function complete.

$$\underset{\uparrow}{1}\quad 1\quad \$\quad 1\quad 1\quad 1\quad \Delta$$

$$X\quad \underset{\uparrow}{1}\quad \$\quad 1\quad 1\quad 1\quad \Delta$$ mov rt upto $

$$X\quad 1\quad \underset{\uparrow}{\$}\quad 1\quad 1\quad 1\quad \Delta$$ mov rt

$$X\quad 1\quad \$\quad \underset{\uparrow}{1}\quad 1\quad 1\quad \Delta$$ copy this 1 after Δ by marking it as Y.

```
X 1 $ Y 1 1 Δ 1 Δ      Now mov to left upto Y.
              ↑

X 1 $ Y 1 1 Δ 1 Δ
            ↑
X 1 $ Y 1 1 Δ 1 Δ
          ↑
X 1 $ Y 1 1 Δ 1 Δ
        ↑
X 1 $ Y 1 1 Δ 1 Δ      move rt
      ↑

X 1 $ Y 1 1 Δ 1 Δ      mark it as Y and copy this 1
        ↑              at the right most end.

X 1 $ Y Y 1 Δ 1 1 Δ    mov left upto Y
                ↑
X 1 $ Y Y 1 Δ 1 1 Δ
              ↑
X 1 $ Y Y 1 Δ 1 1 Δ
            ↑
X 1 $ Y Y 1 Δ 1 1 Δ
          ↑
X 1 $ Y Y 1 Δ 1 1 Δ    mov rt
        ↑

X 1 $ Y Y 1 Δ 1 1 Δ    mark it as Y and copy this 1 at
          ↑            The rt most end.

X 1 $ Y Y Y Δ 1 1 1 Δ  mov left till Y.
                  ↑
X 1 $ Y Y Y Δ 1 1 1 Δ
                ↑
X 1 $ Y Y Y Δ 1 1 1 Δ
              ↑
X 1 $ Y Y Y Δ 1 1 1 Δ
            ↑
X 1 $ Y Y Y Δ 1 1 1 Δ  The head is pointing to Y, next to
          ↑            Y is Δ that means all the
                       1's are over. We will convert these
                       Y's to 1s.
```

And repeat the same procedure for the 1 which is left to $.

X 1 $ Y Y Y Δ 1 1 1
  ↑

X X $ Y Y Y Δ 1 1 1
    ↑

X X $ Y Y Y Δ 1 1 1 → mark it as 1 and copy 1
      ↑ after rt most end.

X X $ 1 Y Y Δ 1 1 1 1 Δ mov left till Y.
                   ↑

X X $ 1 Y Y Δ 1 1 1 1
      ↑ ↑ ↑ ↑ ↑ ↑

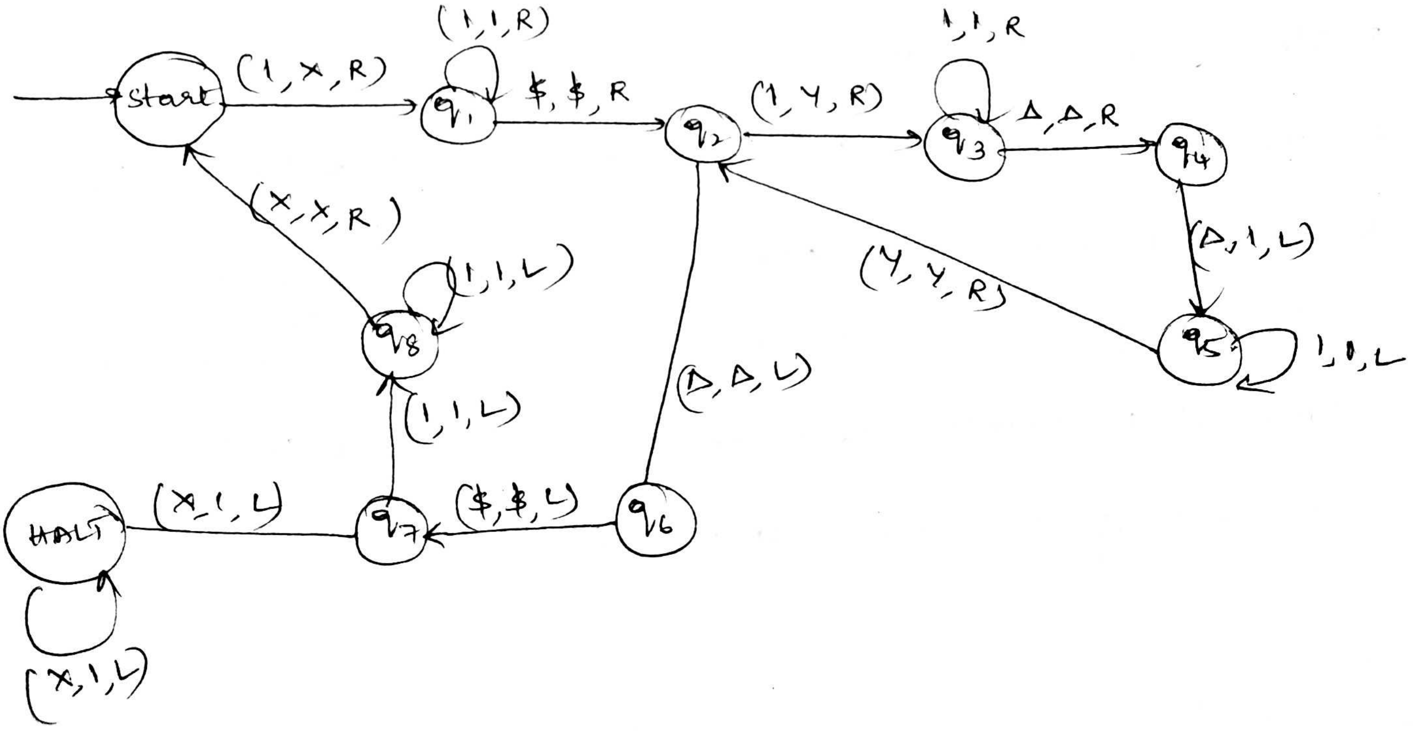
X X $ 1 1 Y Δ 1 1 1 1 1 Δ
        ↑ ↑ ↑ ↑ ↑ ↑ ↑

X X $ 1 1 1 Δ 1 1 1 1 1 1 Δ

X X $ Y Y Y Δ 1 1 1 1 1 1

Convert all X's to 1's.

$$\underbrace{1\ 1}_{a}\ \$\ \underbrace{1\ 1\ 1}_{b}\ \Delta\ \underbrace{1\ 1\ 1\ 1\ 1\ 1}_{\text{answer}}\ \Delta$$



## Extensions to the Basic Turing Machine :

- multi tape Turing machine
- Non deterministic Turing machine.

## MultiTape Turing Machine :

The device has a finite control (state) and some finite number of tapes. Each tape is divided into cells, and each cell can hold any symbol of the finite tape alphabet. As in the single tape TM, the set of tape symbols includes a blank and has a subset ~~of~~ called the input symbols, of which the blank is not a member. The set of states includes an initial state and some accepting states.

1. The input, finite sequence of i/p symbols, is placed on the first tape.
2. All other cells of the tapes hold the blank.
3. The finite control is in the initial state.
4. The head of the first tape is at the left of the input.
5. All other tape heads are at some arbitrary cell.

Since tapes other than the first tape are completely blank it does not matter where the head is placed initially; all cells of these tapes "look" the same.

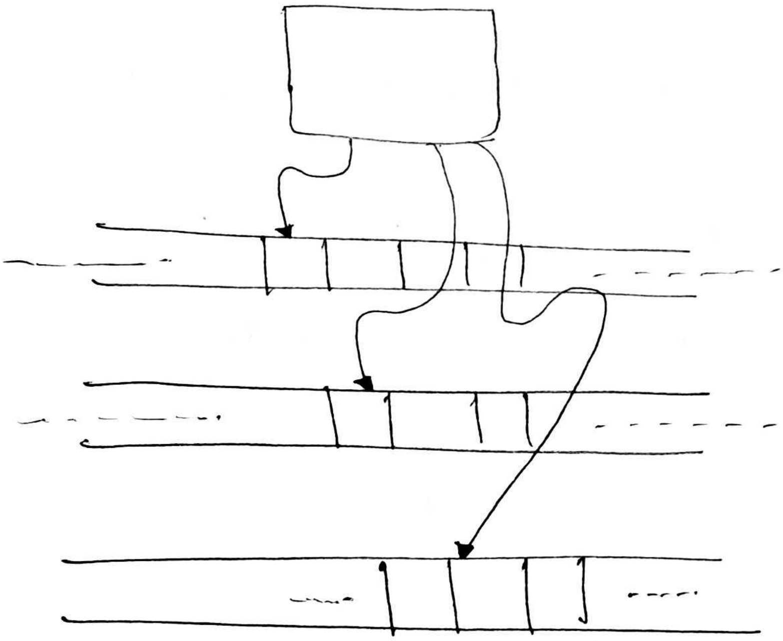
A move of the multi tape TM depends on the state and the symbol scanned by each of the tape heads. In one move, the multitape TM does the following:

(1) The control enters a new state, which could be the same as the previous state.

(2) On each tape, a new tape symbol is written on the cell scanned Any of these symbols may be the same as the symbol previously there.

(3) Each of the tape heads makes a move, which can be either left, right or stationary.

Different heads may move in diff^t directions, and some may not move at all.



A multi tape Turing Machine

## Equivalence of one-tape & Multitape TM's.

- Every language accepted by a multitape TM is recursively enumerable.

- That is, the one tape TM and the multitape TM are equivalent.

## Running Time and Many-Tapes-to-One Construction:

- The time taken by the one tape TM to simulate $n$ moves of the K-tape TM is $O(n^2)$.

- The equivalence of the two types of TM's is good in the sense that their running times are roughly the same within polynomial complexity.

## Non Deterministic TM's.

- A non deterministic TM has multiple choices of next moves. i.e.

$$\delta(q, x) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2) \text{ ---- } (q_K, Y_K, D_K)\}$$

- The NTM is not more 'powerful' than a deterministic TM (DTM).

- If $M_N$ is NTM, then there is a DTM $M_D$ such that

$$L(M_N) = L(M_D)$$

- The equivalent DTM constructed for an NTM may take exponentially more time than the DTM.

- It is unknow whether or not this exponential slowdown is necessary!

## Restricted Turing Machines:

- The tape is infinite only to the right, and the blank cannot be used as a replacement symbol;
- The tapes are only used as stacks ("stack machines")
- The stacks are used as counters only ("counter machines")
- The above restrictions make no decrease of the original TM's power, but are useful for theorem proving.
- Undecidability of the TM also applies to these restricted TM's.

### ⇒ TM's with semi-infinite Tapes:

- Every language accepted by a TM $M_2$ is also eaccepted by a TM $M_1$ with the following restrictions.
  - $M_1$'s head never moves left of its initial position (so the tape is semi-infinite)
  - $M_1$ never writes a blank.

(ie $M_1$ and $M_2$ are equivalent).

### ⇒ Multi stack machines:

- Multi stack Machines, which are restricted versions of TM's may be regarded as extensions of pushdown automata (PDA's).
- Actually a PDA with two stacks has the same computation power as the TM.
- A K-stack machine is a deterministic PDA with K stacks.
- If a language is accepted by a TM, then it is accepted by a two stack machine.

## ⇒ Counter Machines:

- There are two ways to think of a counter machine,
- way 1: as a multistack machine with each stack replaced by a counter regarded to be on a tape of a TM.
  - A Counter holds any non negative integer.
  - The machine can only distinguish zero and non zero counters.
  - A move conducts the following operations:
    - changing the state;
    - add or subtract 1 from a counter which cannot becomes negative.

Way 2: as a restricted multistack machine with each stack replaced by a counter implemented on a stack of a PDA.

- There are only two stack symbols $Z_0$ and $X$.
- $Z_0$ is the initial symbol, like that of a PDA.
- Can replace $Z_0$ only by $X^iZ_0$ for some $i \geq 0$
- Can replace $X$ only by $X^i$ for some $i \geq 0$

## power of counter Machine:

- Every language accepted by a one-counter machine is a CFL.
-

MRCE

Computer Science and Engineering

# FORMAL LANGUAGES AND AUTOMATA THEORY

## Unit-V

Lecture Notes

III YEAR- SEM-I

By

G M SUBHANI,
Assistant Professor
CSE Department,
MRCE, Hyderabad.

- Every language accepted by a counter machine is recursive enumerable.

- Every recursive Enumerable language is accepted by a three-counter machine.

- Every recursive Enumerable language is accepted by a two-counter machine.

## Turing Machines and Computers:

- A computer can simulate a TM
- A TM can simulate a computer.

That means the real computer we use everyday is nearly an implementation of the maximal computational model under the assumptions that

- the memory space (including registers, RAM, hard disks... is infinite in size.

- the address space is infinite (not only that defined by 32 bits used in most computers today).

# UNIT-4

# TURING MACHINE

## Introduction:

Alan Turing is father of such a model which has computing capability of general purpose computer.

- Hence this model is popularly known as Turing Machine.

This machine has following features:

1. It has external memory which remembers arbitrarily long sequence of input.
2. It has unlimited memory capability.
3. The model has a facility by which the i/p at left or right on the tape can be read easily.
4. The machine can produce certain output based on its i/p.

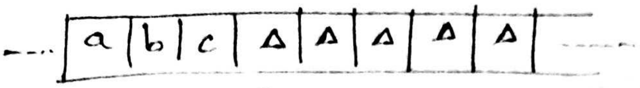
Sometimes it may be required that the same i/p has to be used to generate the o/p. so in this machine the distinction b/w i/p and o/p has been removed. Thus a common set of alphabets can be used for the turing machine.

## BASIC MODEL:

The turing machine can be modelled with the help of following representation.

1) The i/p tape having infinite no. of cells, each cell containing one i/p symbol and thus the i/p string can be placed on a tape.

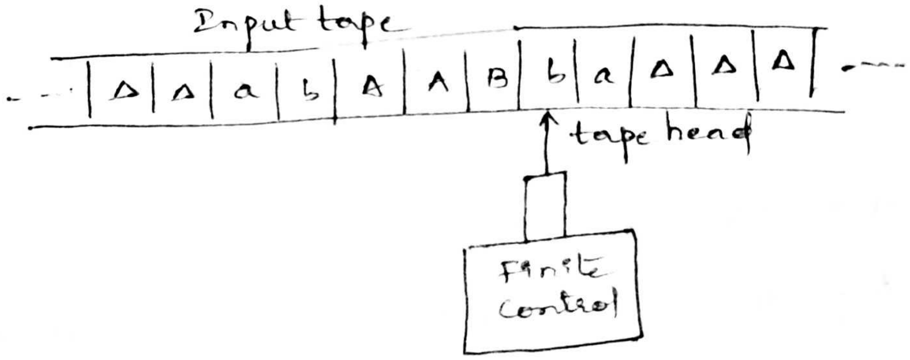
The empty tape is filled by blank characters.



Input tape

2) The finite control and the tape head which is responsible for reading the current i/p symbol. Tape head can move from to left or right.

3) Finite set of symbols called external symbols which are used in building the logic of turing machine.



## DEFINITION OF TM:

Turing machine (TM) is denoted as,

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

Where

$Q$ = finite set of states

$\Sigma$ = Set of input symbols

$\Gamma$ = Set of allowable tape symbols.

$\delta$ = mapping function.

$q_0$ = Start state

$B$ = Blank symbol.

$F$ = Set of final states.

## DESIGN OF TM:

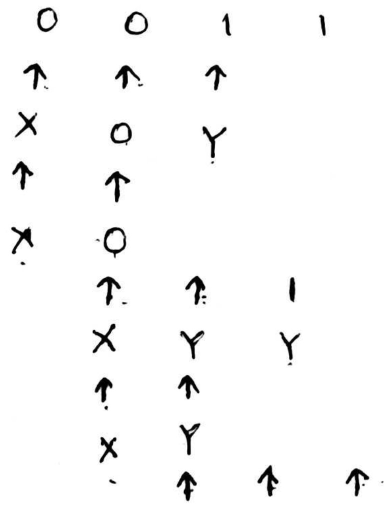
Basic guidelines for designing a TM are,

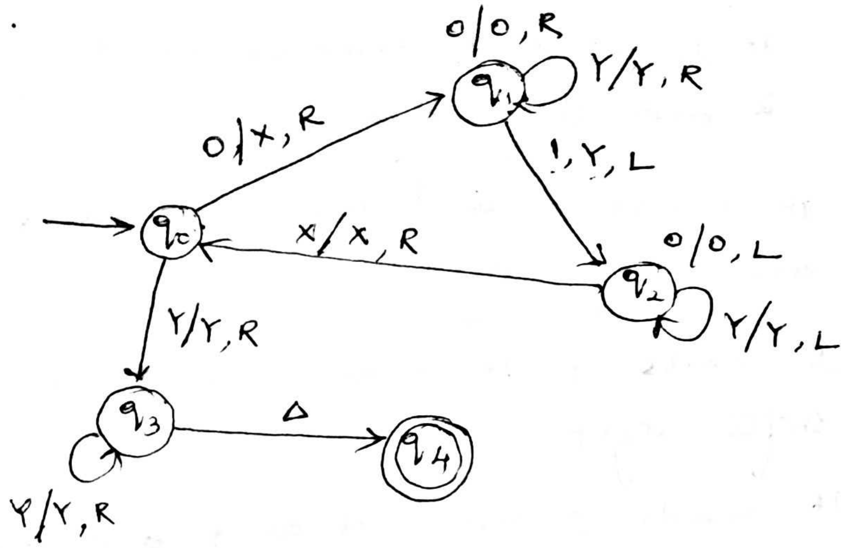
1) scanning a symbol by R/w head is to 'know' what to do in the future. The machine must remember the past symbols scanned.

2) change the states only when there is a change in the movement of R/w head.

Ex: The design of a TM M to accept the language $L = \{0^n1^n \mid n \geq 1\}$.

- Initially the tape of M contains $0^n1^n$ followed by an infinity of blanks.
- Repeatedly, M replaces the leftmost 0 by X, moves right to the leftmost 1, replacing it by Y, moves left to find the right most X, then moves one cell right to the leftmost 0 and repeats the cycle.
- If however, when searching for a 1, M finds a blank instead, then M halts without accepting.
- If after changing 1 to Y, M finds no more 0's then M checks that no more 1's remain.





The function $\delta$ is given by

| state | 0 | 1 | symbol X | symbol Y | B. |
|---|---|---|---|---|---|
| $q_0$ | $(q_1, X, R)$ | - | - | $(q_3, Y, R)$ | - |
| $q_1$ | $(q_1, 0, R)$ | $(q_2, Y, L)$ | - | $(q_1, Y, R)$ | - |
| $q_2$ | $q_2, 0, L$ | - | - | $(q_2, Y, R)$ | - |
| $q_3$ | - | - | $(q_0, X, R)$ | $(q_3, Y, R)$ | $(q_4, B, R)$ |
| $q_4$ | - | - | - | - | - |

A TM M Accepting $0^n1^n$.

Computation of M on input 0011.

~~$q_0, 0011$~~ ~~$\vdash X q_1 011$~~

~~$\vdash X0q_1 11$~~

~~$\vdash X$~~

0 0 1 1 Δ — Initially it reads 0.
↑

It is marked as X. & shift right.

X 0 1 1 Δ — It reads 0. 0 remains as it is. shifts right.
   ↑

X 0 1 1 Δ — It reads 1. It is marked as Y. shifts left.
      ↑

X 0 Y 1 Δ — It reads 0. & it remains as it is. shifts left.
   ↑

$\underset{\uparrow}{X}$ 0 Y 1 Δ It reads X. It remains as it is and shifts right

X $\underset{\uparrow}{0}$ Y 1 Δ It reads 0 and marks it as X and shifts right

X X $\underset{\uparrow}{Y}$ 1 Δ It reads Y. It remains as it is and shifts right

X X Y $\underset{\uparrow}{1}$ Δ It reads 1 marks it as Y and shifts left

X X $\underset{\uparrow}{Y}$ Y Δ It reads 'Y. It remains as it is and shifts left.

X $\underset{\uparrow}{X}$ Y Y Δ It reads X. It remains as it is and shifts right.

X X $\underset{\uparrow}{Y}$ Y Δ It reads Y remains as it is and shifts right.

X X Y $\underset{\uparrow}{Y}$ Δ It reads Y remains as it is and shifts right.

X X Y Y $\underset{\uparrow}{Δ}$ It reads Δ and halts.

Ex:2 Design turing machine to recognize the language $\{1^n 2^n 3^n \mid n \geq 1\}$.

Soln we have to follow the above procedure.

## COMPUTABLE LANGUAGES AND FUNCTIONS :

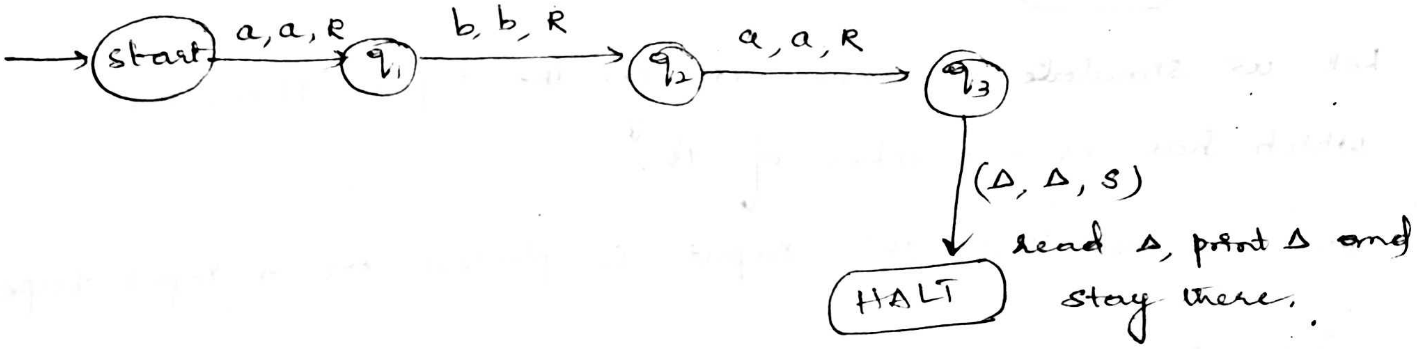
① construct a TM which accepts the language of aba over $\Sigma = \{a, b\}$.

Sol$^n$ This TM is only for $L = \{aba\}$

On the i/p tape the string aba is placed.

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| a | b | a | Δ | Δ | ---- |

The tape head will read out the sequence upto the character Δ. If aba is read out the TM will halt after reading Δ.



read Δ, print Δ and stay there.

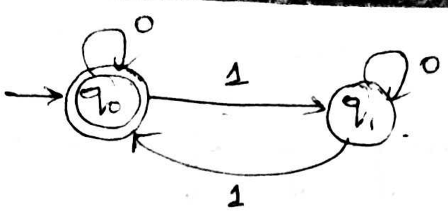
TM can be represented by transition table.

|   | a | b | Δ |
|---|---|---|---|
| start | $(q_1, a, R)$ | – | — |
| $q_1$ | – | $(q_2, b, R)$ | – |
| $q_2$ | $(q_3, a, R)$ | – | – |
| $q_3$ | - | – | (HALT, Δ, S) |
| HALT | – | – | – |

② construct TM for language consisting of strings having any no. of 0's and only even no. of 1's over the input set $\Sigma = \{0, 1\}$.

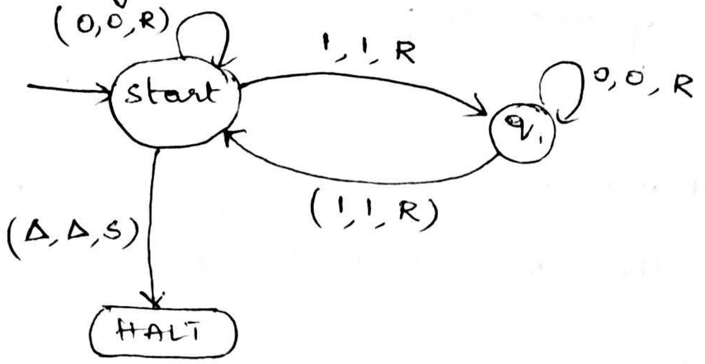
Sol$^n$ The number of 0's can be any.

But the even no. of 1's are to be allowed.



The same idea can be used to draw TM.

We always assume that at the end of input Δ is placed.



Let us simulate the above TM for the input 11010l which has even number of 1's.

We assume that this input is placed on a input tape.

I/p head  1  1  0  1  0  1  Δ
↑
state  $q_1$

1  1  0  1  0  1  Δ
↑
start

1  1  0  1  0  1  Δ
↑
start

1  1  0  1  0  1  Δ
↑
$q_1$

1  1  0  1  0  1  Δ
↑
$q_1$

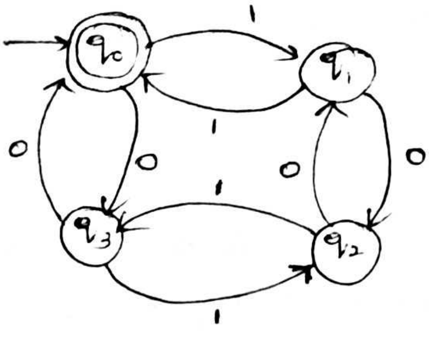
1  1  0  1  0  1  Δ
↑
start

1  1  0  1  0  1  Δ
↑
HALT.

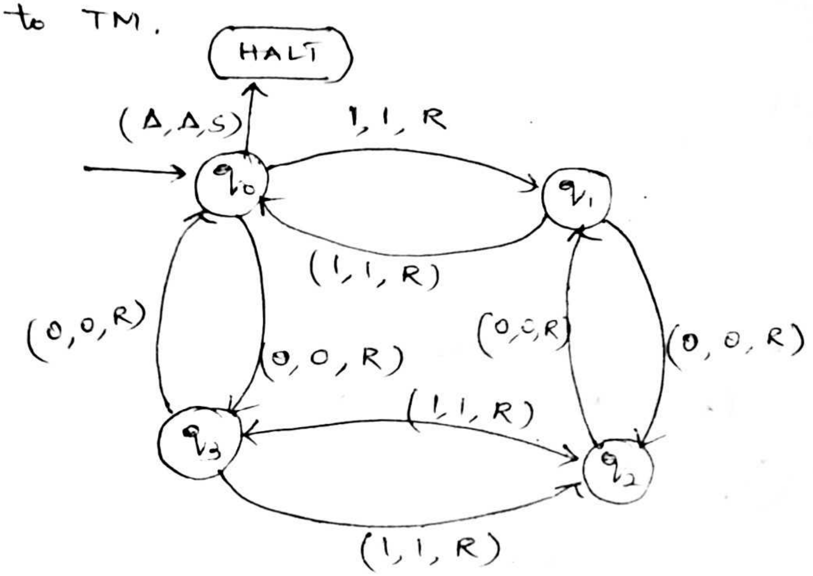
Thus the I/p is accepted by TM

③ construct a TM for the language of even no of 1's and even no of 0's over $\Sigma = \{0, 1\}$.

<u>sol^n</u> For this problem we can draw FSM as.



Now it will be very much easy to convert this FSM to TM.



④ Construct TM for the language $L = \{a^n b^n\}$ where $n \geq 1$.

<u>sol^n</u> Let us formulise the logic for $a^n b^n$ and then construct it.

a a b b Δ   Convert it to A & move right in search of b.
↑

A a b b Δ   skip it, move ahead.
   ↑

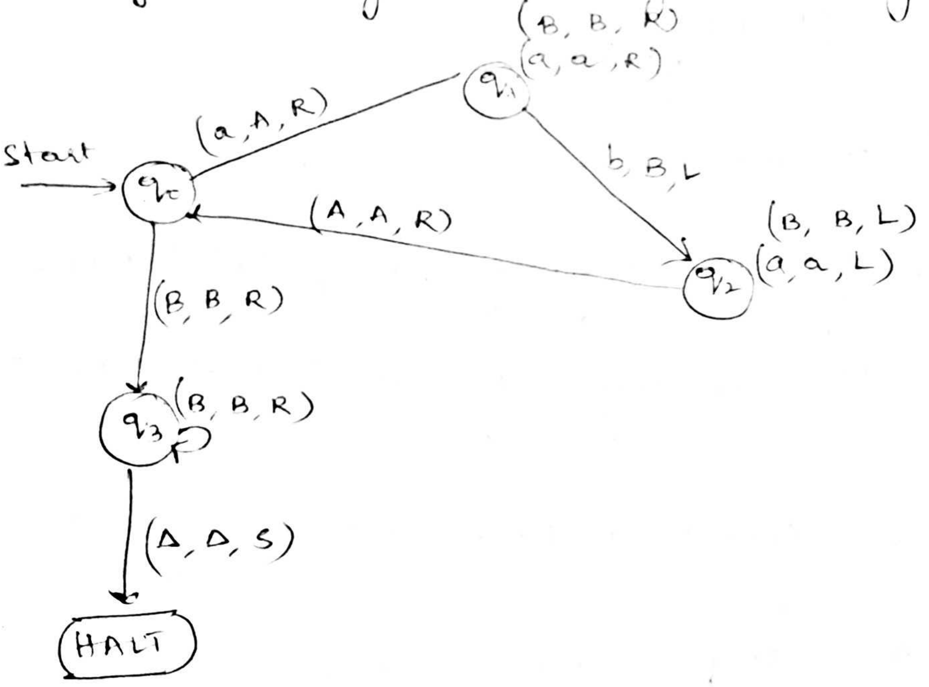
A a b b Δ   convert it to B, move left till A
      ↑

A a B b Δ   move left
   ↑

A a B b Δ   move right
↑

```
A  a  B  b  Δ      Convert a to A, move right in
   ↑               search of b.

A  A  B  b  Δ      move right
      ↑

A  A  B  b  Δ      Convert b to B, move left
         ↑

A  A  B  B         move left
      ↑

A  A  B  B  Δ      Now immediately before B is A that means
   ↑               all the a's are marked by A. So we will
                   move right to ensure that no b is present.

A  A  B  B  Δ      move right
      ↑

A  A  B  B  Δ      move right
         ↑

A  A  B  B  Δ      HALT.
            ↑
```

- Let us try to design TM using above logic.



Transition table :

| | a | b | A | B | Δ |
|---|---|---|---|---|---|
| $q_0$ | $(q_1, A, R)$ | – | – | $(q_3, B, R)$ | – |
| $q_1$ | $(q_1, a, R)$ | $(q_2, B, L)$ | – | $(q_1, B, R)$ | – |
| $q_2$ | $(q_2, a, L)$ | – | $(q_0, A, R)$ | $(q_2, B, L)$ | – |
| $q_3$ | – | – | – | $(q_3, B, R)$ | (HALT, Δ, S) |
| HALT | – | – | – | – | – |

(5) construct a TM for $L = \{a^n b^n c^n \mid n \geq 1\}$.

sol$^n$ The simulation for aabbcc can be shown below.

$\underset{\uparrow}{a}$ a b b c c Δ convert a to A move right

A $\underset{\uparrow}{a}$ b b c c Δ move right upto c

A a $\underset{\uparrow}{b}$ b c c Δ Convert b to B , move right

A a B $\underset{\uparrow}{b}$ c c Δ move right

A a B b $\underset{\uparrow}{c}$ c Δ Convert c to C , move ~~right~~ left

A a B b C $\underset{\uparrow}{c}$ Δ move left

A a B b $\underset{\uparrow}{C}$ c Δ move left

A a B $\underset{\uparrow}{b}$ C c Δ move left

A a $\underset{\uparrow}{B}$ b C c Δ move left

A $\underset{\uparrow}{a}$ B b C c Δ move left

$\underset{\uparrow}{A}$ a B b C c Δ move right

A $\underset{\uparrow}{a}$ B b C c Δ Covert a to A move right

A A $\underset{\uparrow}{B}$ b C c Δ move right

A A B b C c Δ   convert b to B , move right

↑

A A B B C c Δ   move right

↑

A A B B C c Δ   convert c to C , move left

↑

~~A A B B C C Δ~~   ~~move left~~ ✓

↑

A A B B C C Δ   move left

↑

A A B B C C Δ   move left

↑

A A B B C C Δ   move right

↑

A A B B C C Δ   move right

↑

A A B B C C Δ   move right

↑

A A B B C C Δ   move right

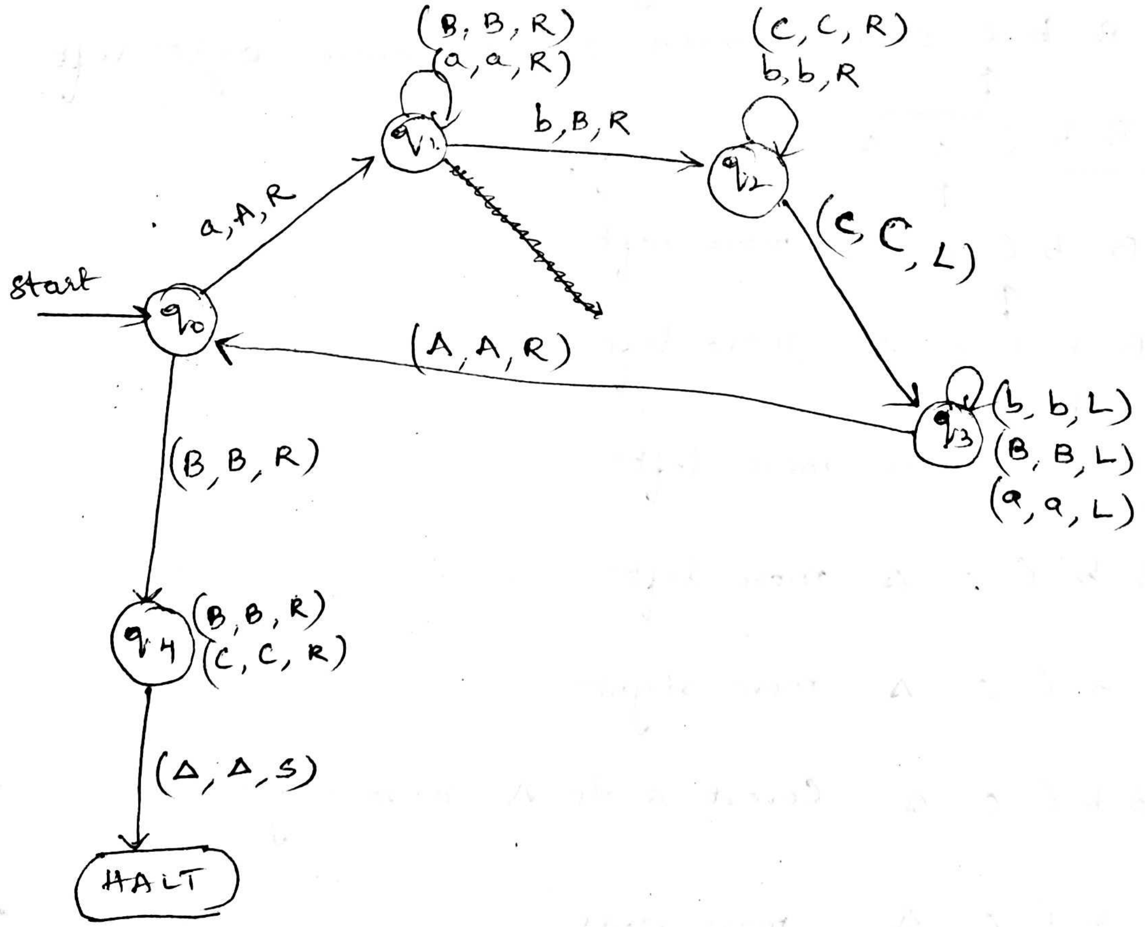
↑

A A B B C C Δ   move right

↑

A A B B C C Δ   since Δ is read , TM will reach to HALT state

↑



(6) construct a TM for checking the palindrome of the string of even length.

Soln The logic is that we will read first symbol from left and then we compare it with the first symbol from right to check whether it is the same.

- Again we compare second symbol from left with the second symbol from right.
- We repeat this process for all the symbols.
- If we found any symbol not matching, we cannot lead the machine to HALT state

```
a b a b b b a b a Δ    mark it by *, move to right till Δ.
↑
* b a b b b a b a Δ    move right
  ↑
* b a b b b a b a Δ    move right
    ↑
* b a b b b a b a Δ    move right
      ↑
* b a b b b a b a Δ    move right
        ↑
* b a b b b a b a Δ    move right
          ↑
* b a b b b a b a Δ    move right
            ↑
* b a b b b a b a Δ    move right
              ↑
* b a b b b a b a Δ    move to left ~~upto~~ ~~*~~, check if it is a
                  ↑
* b a b b b a b a Δ    ~~[illegible]~~ ~~[illegible]~~ if it is a, replace it by Δ, move left
                ↑                                                          upto *.
* b a b b b a b ▲ Δ    move left
              ↑
* b a b b b a b ▲ Δ    move left
            ↑
* b a b b b a b ▲ Δ    move left
        ↑
* b a b b b a b ▲ Δ    move left.
      ↑
```

```
* b a b b a b Δ Δ      move left
    ↑
* b a b b a b Δ Δ      move left
  ↑
* b a b b a b Δ Δ      move right
↑
* b a b b a b Δ Δ      convert it to * , move right
  ↑
* * a b b a b Δ        move right upto Δ
    ↑
* * a b b a b Δ        move right
      ↑
* * a b b a b Δ        move right
        ↑
* * a b b a b Δ        move right
          ↑
* * a b b a b Δ        move right
            ↑
* * a b b a b Δ        move left, if left symbol is b Convert it to
              ↑                                               Δ.
* * a b b a b Δ        convert it to Δ, move left
            ↑
* * a b b a Δ Δ        move left till *
          ↑
* * a b b a Δ          move left
        ↑
* * a b b a Δ          move left
      ↑
* * a b b a Δ          move left
    ↑
* * a b b a Δ          Goto right
  ↑
* * a b b a Δ          replace a by * , move right. upto Δ
    ↑
* * * b b a Δ          move right
      ↑
* * * b b a Δ          move right
        ↑
* * * b b a Δ          move right
          ↑
* * * b b a Δ          move left, if it is a replace it by Δ &
            ↑                                           move left
```

```
* b a b b a b Δ Δ ·   move left
    ↑
* b a b b a b Δ Δ     move left
  ↑
* b a b b a b Δ Δ     move right
↑
* b a b b a b Δ Δ     convert it to * , move right
  ↑
* * a b b a b Δ       move right upto Δ
    ↑
* * a b b a b Δ       move right
      ↑
* * a b b a b Δ       move right
        ↑
* * a b b a b Δ       move right
          ↑
* * a b b a b Δ       move right
            ↑
* * a b b a b Δ       move left, if left symbol is b Convert it to
              ↑                                               Δ.
* * a b b a b Δ       convert it to Δ, move left
            ↑
* * a b b a Δ Δ       move left till *
          ↑
* * a b b a Δ         move left
        ↑
* * a b b a Δ         move left
      ↑
* * a b b a Δ         move left
    ↑
* * a b b a Δ         Goto right
  ↑
* * a b b a Δ         replace a by * , move right. upto Δ
    ↑
* * * b b a Δ         move right
      ↑
* * * b b a Δ         move right
        ↑
* * * b b a Δ         move right
          ↑
* * * b b a Δ         move left, if it is a replace it by Δ &
            ↑                                        move left
```

* * * b b a Δ replace it by Δ, move left
↑

* * * b b Δ move left
↑

* * * b b Δ move left
↑

* * * b b Δ move right
↑

* * * b b Δ Convert it to *, move right
↑

* * * * b Δ move right
↑

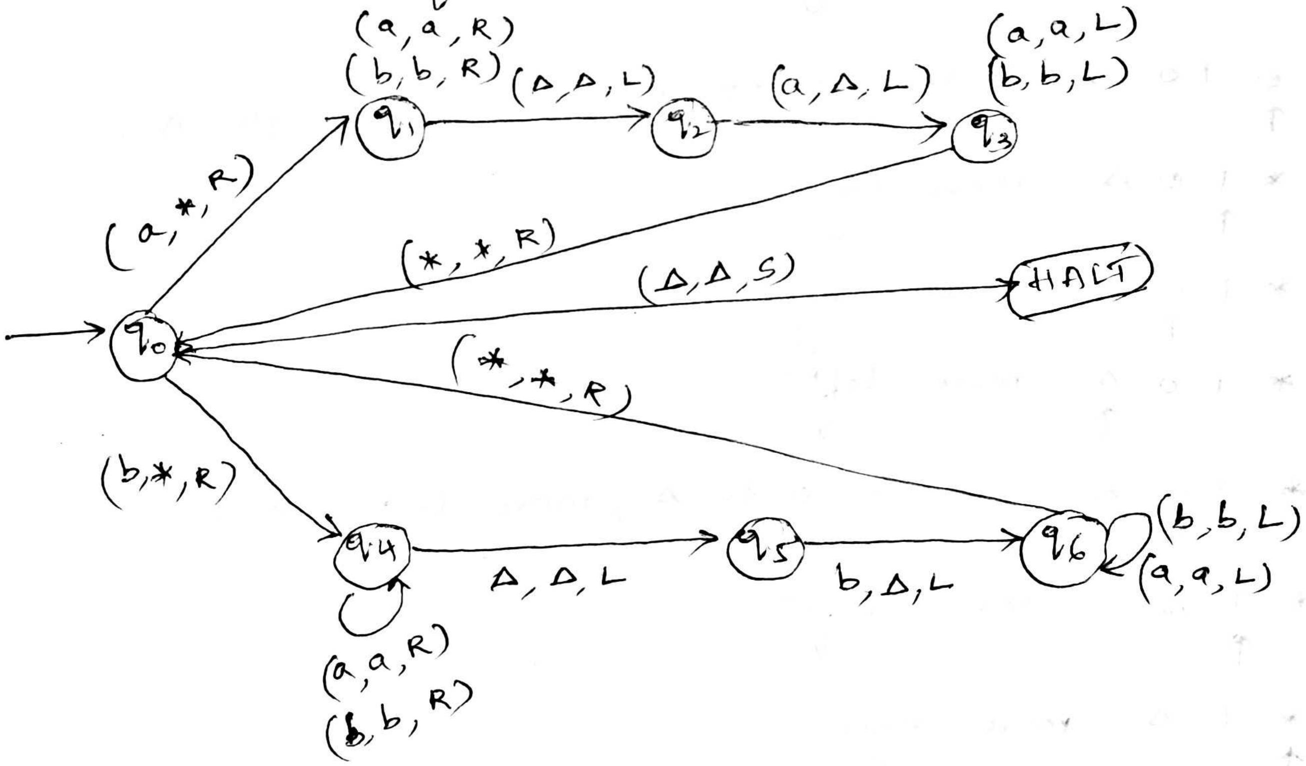
* * * * b Δ move left
↑

* * * * b Δ convert it to Δ, move left
↑

* * * * Δ Δ move right, check whether it is Δ
↑

* * * * Δ Goto HALT state.
↑

Let us draw TM for it.



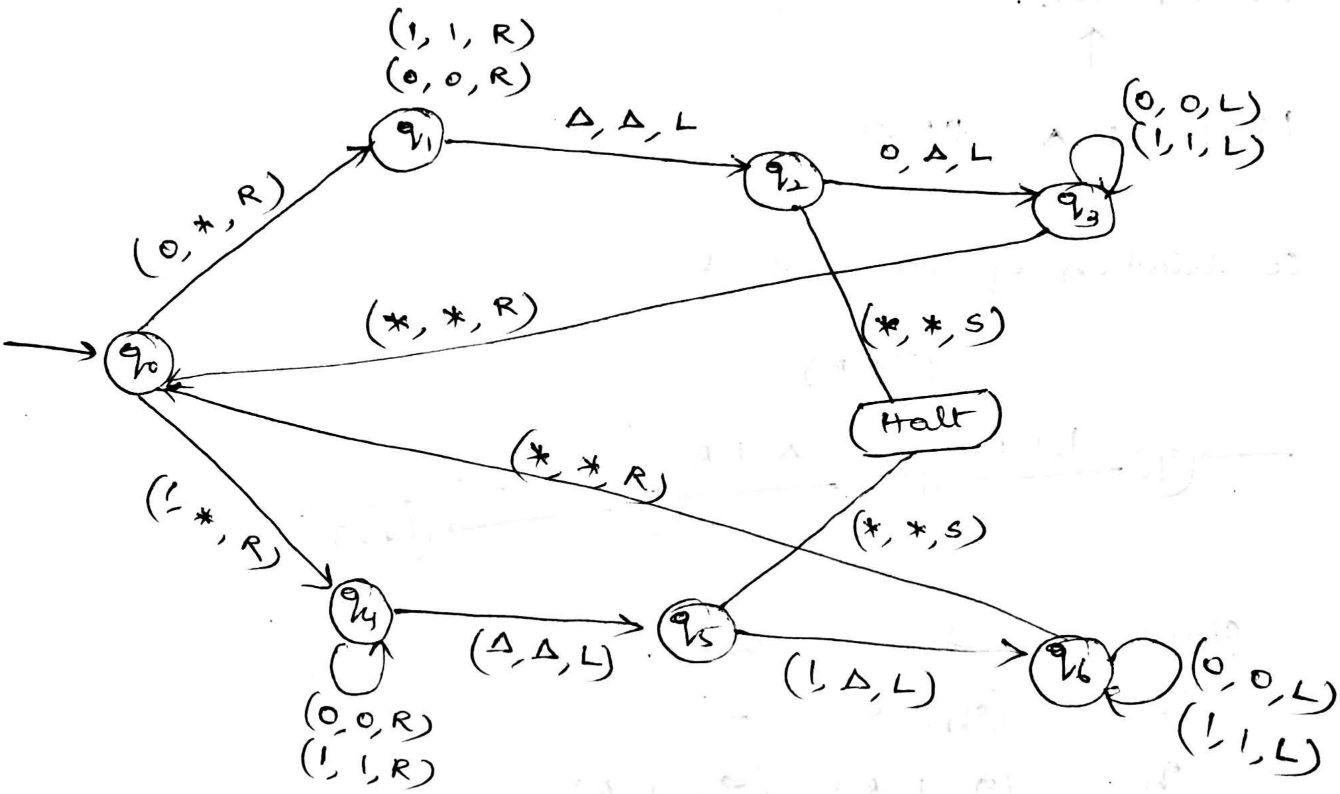
(7) Construct a TM for checking the palendrome of a string of odd palendrome for Σ = {0, 1}.

Another string :- 01010Δ

Soln : for eg:- if the string is

```
0 0 1 0 0 Δ     move right by making 0 to *
↑

* 0 1 0 0 Δ     move right upto Δ
  ↑

* 0 1 0 0 Δ     move left, replace 0 by Δ
          ↑

* 0 1 0 0 Δ     replace 0 by Δ, move left upto *
        ↑

* 0 1 0 Δ       move left
      ↑

* 0 1 0 Δ       move left
    ↑

* 0 1 0 Δ       move left
  ↑

* 0 1 0 Δ       move right
↑

* 0 1 0 Δ       Convert 0 to *, move right upto Δ
  ↑

* * 1 0 Δ       move right
    ↑

* * 1 0 Δ       move right
      ↑

* * 1 0 Δ       move left
        ↑

* * 1 0 Δ       Convert 0 to Δ, move left till *
      ↑

* * 1 Δ         move left
    ↑

* * 1 Δ         move right
  ↑

* * 1 Δ         Convert 1 to *, move right
    ↑

* * * Δ         move left
      ↑
```

```
* * * Δ    Since it is *, goto Halt state.
    ↑
```



(8) Construct a TM for a successor function for a given unary number i.e f(n) = n+1.

sol<sup></sup> Logic : We just ~~make~~ go on moving towards right most end of i/p string. The End marker of the string Δ is replaced by 1.

For eg:- the i/p tape consists of 4, the successor function will give o/p as 5.

```
1 1 1 1 Δ    move to Extreme right by keeping all
↑            1's as it is.

1 1 1 1 Δ    move right
  ↑

1 1 1 1 Δ    move right
    ↑
```

1 1  1 1 Δ    move right

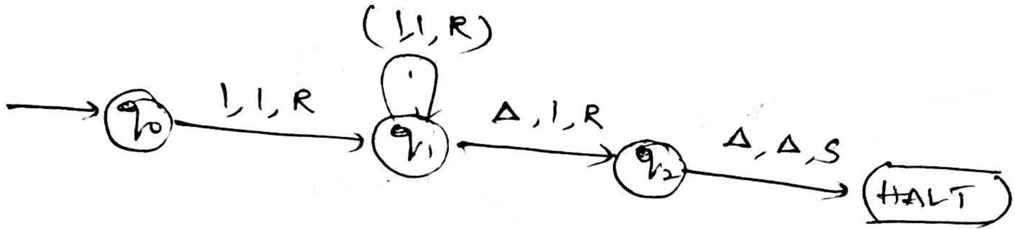
↑

1 1  1 1  Δ Δ   convert Δ to 1

↑

1 1  1 1  1 Δ    HALT

Construction of TM will be



| States | 1 | Δ |
|---|---|---|
| $q_0$ | $(q_1, 1, R)$ | – |
| $q_1$ | $(q_1, 1, R)$ | $(q_2, 1, R)$ |
| $q_2$ | - | (HALT, Δ, S) |
| HALT | – | – |

(9) construct TM to add two given integers.

soln In this TM we are going to perform addition of two unary numbers.

For ex: 2+3

i,e 11 + 111

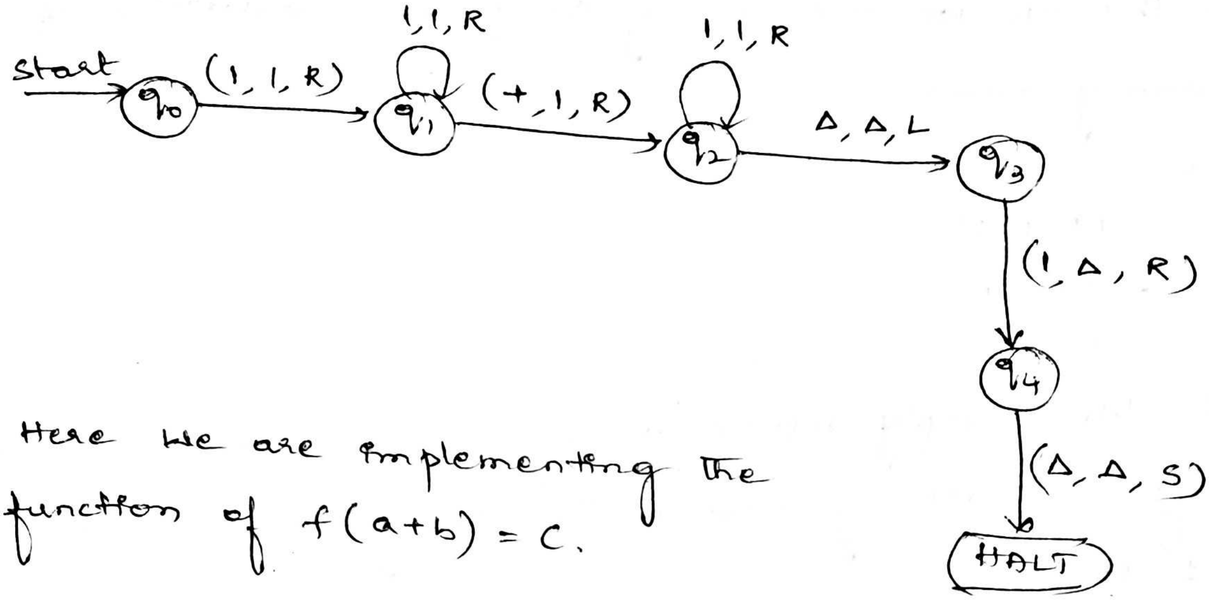
= 11111

Logic: We simply replace '+' by 1 and move ahead right for searching end of the string. We will convert last 1 to Δ.

The input is 3+2.

```
1 1 1 + 1 1 Δ     move right upto + sign
↑
1 1 1 + 1 1 Δ     move right
  ↑
1 1 1 + 1 1 Δ     move right
    ↑
1 1 1 + 1 1 Δ     Convert '+' to 1 and move right
      ↑
1 1 1 1 1 1 Δ     move right
        ↑
1 1 1 1 1 1 Δ     move right
          ↑
1 1 1 1 1 1 Δ     Now Δ has encountered so just move left
            ↑
1 1 1 1 1 1 Δ     convert 1 to Δ
          ↑
1 1 1 1 1 Δ Δ     Thus the tape now consists of the addition
                  of two unary numbers.
```

The TM will look like this,



Here we are implementing the function of $f(a+b) = c$.

(10) Construct TM for performing subtraction of two unary numbers $f(a-b) = c$ where a is always greater than b.

Sol

Soln Here we have assumption as first number is greater than second one.

Let us assume $a = 3$, $b = 2$, so the i/p tape will be

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | - | 1 | 1 | Δ |

We will move right to '-' symbol as perform reduction of number of 1's from first number.

Let us look at the simulation for understanding the logic.

1 1 1 - 1 1 Δ    move right upto '-'
↑

1 1 1 - 1 1 Δ    move right
  ↑

1 1 1 - 1 1 Δ    move right
    ↑

1 1 1 $\underset{\uparrow}{-}$ 1 1 Δ ~~convert~~ move right.

1 1 1 – $\underset{\uparrow}{1}$ 1 Δ convert 1 to *, move left

1 1 1 $\underset{\uparrow}{-}$ * 1 Δ move left

1 1 $\underset{\uparrow}{1}$ – * 1 Δ convert 1 to *, move right

1 1 * $\underset{\uparrow}{-}$ * 1 Δ move right till 1

1 1 * – $\underset{\uparrow}{\ast}$ 1 Δ move right

1 1 * — * $\underset{\uparrow}{1}$ Δ convert 1 to *, move left till 1

1 1 * – $\underset{\uparrow}{\ast}$ * Δ move left

1 1 * $\underset{\uparrow}{-}$ * * Δ move left

1 1 $\underset{\uparrow}{\ast}$ – * * Δ move left

1 $\underset{\uparrow}{1}$ * – * * Δ convert 1 to *, move right

1 * $\underset{\uparrow}{\ast}$ – * * Δ move right

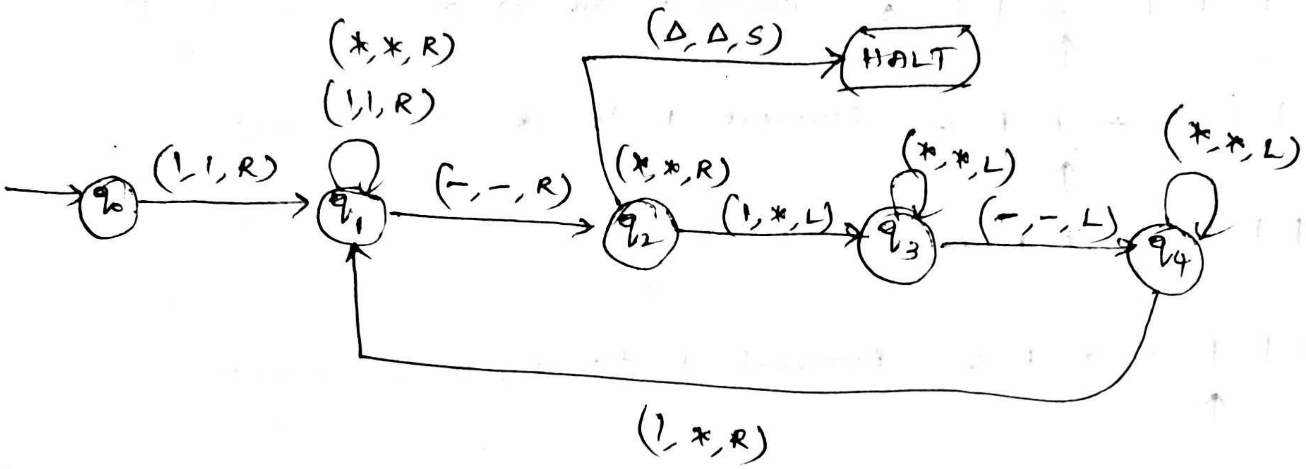
1 * * $\underset{\uparrow}{-}$ * * Δ move right

1 * * – $\underset{\uparrow}{\ast}$ * Δ move right

1 * * – * $\underset{\uparrow}{\ast}$ Δ move right

1 * * – * * $\underset{\uparrow}{\Delta}$ HALT

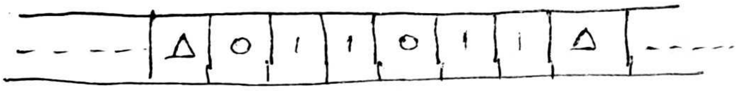
Thus we get 1 on the i/p tape as answer for f(3-2).



## TWO WAY INFINITE TAPE:

- The TM is widely accepted as a model of computation.
- The TM can perform string operations, arithmetic computations, recognizing languages either regular or non regular languages.

- The tape head as usual can move in forward and backward direction.



Two way infinite tape.

- The TM with infinite tape can also be denoted by

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

$Q$ = finite non Empty set of states.

$\Gamma$ = set of external symbols used.

$\delta$ = mapping function.

$q_0$ = Initial state

$B$ = Blank symbol.

$F$ = Final state.

① Construct a TM for a language having equal number of a's and b's in it over the input set $\Sigma = \{a, b\}$.

Sol

Soln Here we will clearly take the advantage of infinite two ends of the input tape.

| --- | Δ | b | a | a | b | Δ | ---- |
|---|---|---|---|---|---|---|---|

Logic: If we get a, we will replace it by A and move right in search of b, if we get b we will replace it by B and move left.

- We will move to the leftmost end (i.e Δ) and again repeat the same process of marking a by A and b by B.
- We may have string starting with a or b, so from initial state only we will have two seperate paths shown for a and b.

$\Delta\ \underset{\uparrow}{b}\ a\ a\ b\ \Delta$ Convert b to B, move right in search of a.

$\Delta\ B\ \underset{\uparrow}{a}\ a\ b\ \Delta$ a has to be converted to A, move left.

$\Delta\ \underset{\uparrow}{B}\ \underset{\uparrow}{A}\ a\ b\ \Delta$ skip B, move left

$\underset{\uparrow}{\Delta}\ B\ A\ a\ b\ \Delta$ Keep Δ as it is, move right.

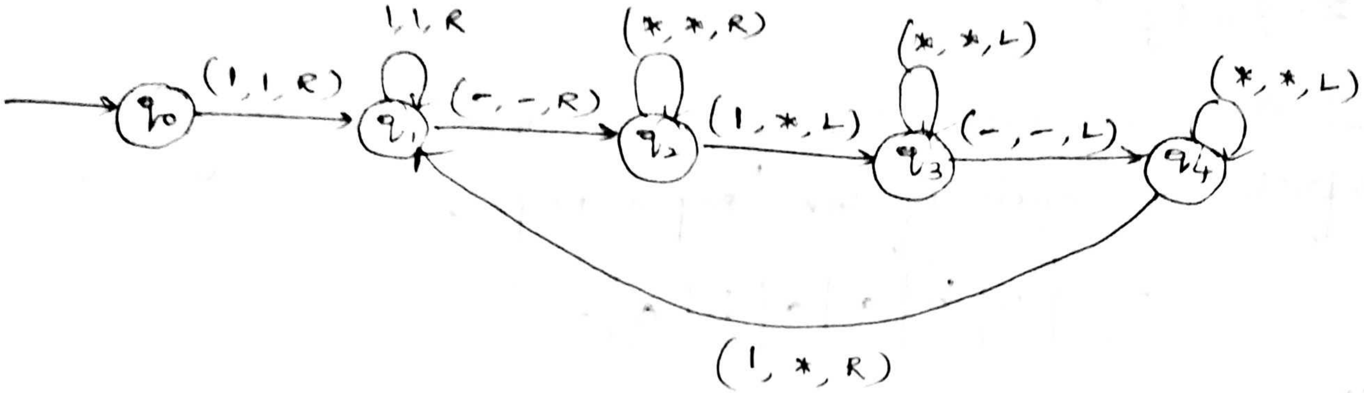
$\Delta\ \underset{\uparrow}{B}\ A\ a\ b\ \Delta$ move right

$\Delta\ B\ \underset{\uparrow}{A}\ a\ b\ \Delta$ move right

$\Delta\ B\ A\ \underset{\uparrow}{a}\ b\ \Delta$ Convert a to A, move right

$\Delta\ B\ A\ A\ \underset{\uparrow}{b}\ \Delta$ convert b to B, move left

$\Delta\ B\ A\ \underset{\uparrow}{A}\ B\ \Delta$ move left let us draw TM for it.



Δ B A A B Δ   move left
↑

Δ B A A B Δ   move left
↑

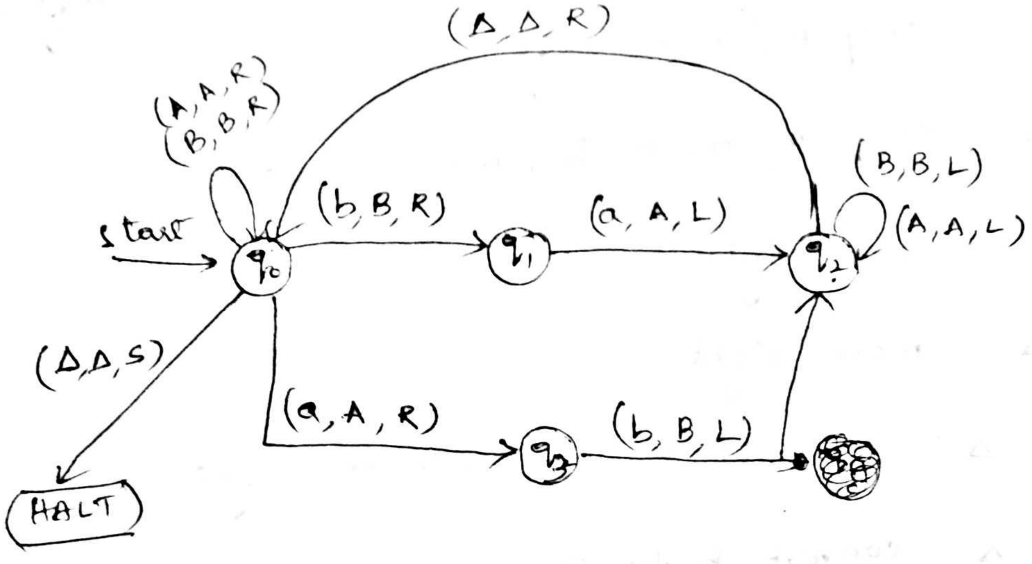
Δ B A A B Δ   If Δ comes move to right by ignoring all A's and B's.
↑

Δ B A A B Δ   move right
↑

Δ B A A B Δ   move right
↑

Δ B A A B Δ   move right
↑

Δ B A A B Δ   Δ is reached, goto HALT state
↑



IIIrdly if the string is

```
Δ a b a b Δ   convert a to A, move right
  ↑
Δ A b a b Δ   Convert b to B, move left
    ↑
Δ A B a b Δ   move left
  ↑
Δ A B a b Δ   move right
↑
Δ A B a b Δ   move right
  ↑
Δ A B a b Δ   move right
    ↑
Δ A B a b Δ   Convert a to A, move right
      ↑
Δ A B A b Δ   Convert b to B, move left
        ↑
Δ A B A B Δ   move left
      ↑
Δ A B A B Δ   move left
    ↑
Δ A B A B Δ   move left
  ↑
Δ A B A B Δ   move right
↑
Δ A B A B Δ   move right
  ↑
Δ A B A B Δ   move right
    ↑
Δ A B A B Δ   move right
      ↑
Δ A B A B Δ   move right
        ↑
Δ A B A B Δ   goto HALT
          ↑
```

② Construct TM for checking well formness of the parenthesis

Sol$^n$ Δ ( ) ( ) Δ   move right in search of ')' skipping '('.
↑

Δ ( ) ( ) Δ   mark it as *, move left in search of '('
↑

Δ ( $\overset{*}{)}$ ( ) Δ   Now '(' has encountered mark it as *. move ~~left~~ right in search of '(' mark it *
↑

Δ * * ( ) Δ   move right, skip *
↑

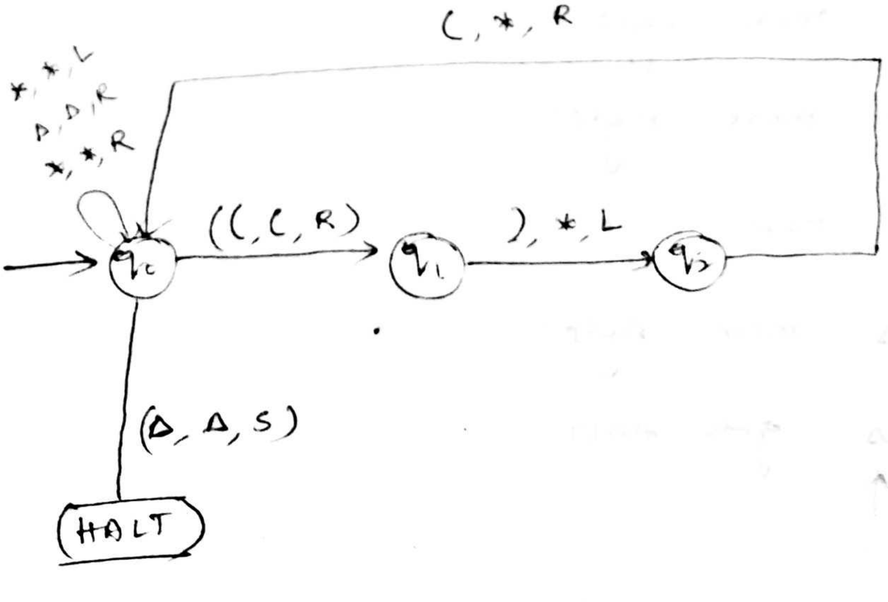
Δ * * ( ) Δ   move right
↑

Δ * * ( ) Δ   Convert to *, move left
↑

Δ * * ( * Δ   Convert ( to *
↑

Δ * * * * Δ   move right till Δ
↑

Δ * * * * Δ   HALT
↑



## Properties of Recursive and Recursively Enumerable Languages.

- The language accepted by FM is called a regular language, the language accepted by PDA is a context free language.

But TM defines two classes of languages such as recursively enumerable language and recursive lang.

When a string belonging to Σ runs on TM then it results in three states

i) Acceptance of string

ii) Loops for ever on receiving string

iii) Rejects the given string

## Recursively Enumerable Language:

The language L ∈ Σ is called recursively Enumerable language if there is a Turing machine T that accepts every word in L and either rejects or loops for every word in the complement of L.

## Recursive Language:

A language L over input set Σ is called recursive if there is a TM that accepts every word in L and rejects every word in L'.

From these two languages following observations are made -

1. Every recursive lang is recursively enumerable lang.

2) Not all recursively enumerable languages are recursive.

## CHURCH'S HYPOTHESIS:

- Hypothesis means proposing certain facts.
- The church's Hypothesis or church's Turing thesis can be stated as,

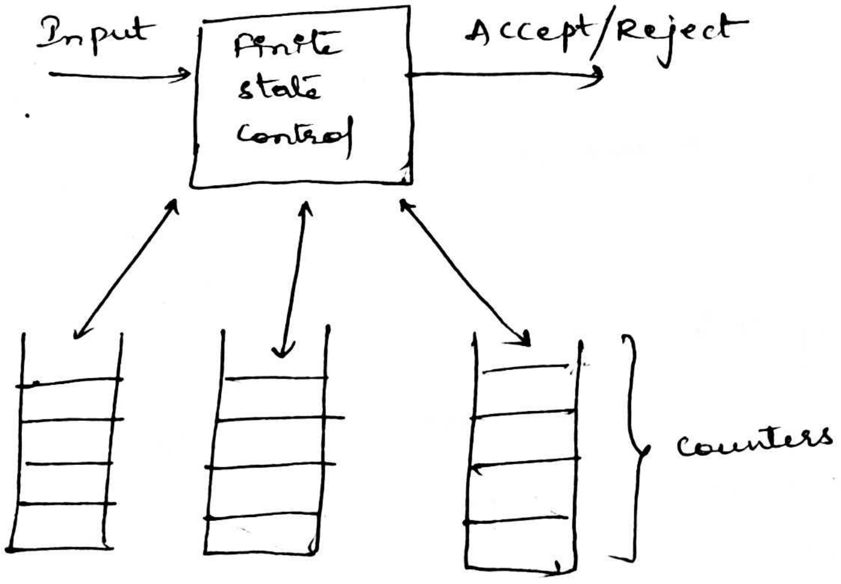
"The assumption that the intuitive notion of computable functions can be identified with partial recursive functions".

However, this hypothesis can not be proved.
The computability of recursive functions is based on following assumptions.

1) Each elementary function is computable.
2) Let f be the computable function and g be another function which can be obtained by applying an elementary operation to f, then g becomes a computable function.
3). Any function becomes computable if it is obtained by rule 1) and 2).

# COUNTER MACHINE.

A counter machine can be rep^td by following model.



Counter machine

There are 2 ways of representing counter machine.

1) The counter machine is similar to a multistack TM, But the only difference b/w them is that in place of each stack there is a counter.

- The counters contain non negative integers.
- Each move of counter machine depends on its state, i/p symbo
- In one move counter machine can:

i) change state

ii) Add or subtract 1 from any of its counters.

- The negative counters are not allowed at all.

2) The counter machine is ||^{rl} to restricted multistack machine. These restrictions are

i) There are only two stack symbols : $Z_0$ and X

ii) The $Z_0$ is the bottom of stack marker.

It is initially on each stack.

iii) Replace $z_0$ only by string of the form $x^i z_0$ where $i \geq 0$

iv) Replace $x$ only by $x^i$ for $i \geq 0$. ie

The $z_0$ appears on the bottom of each stack and all other stack symbols are $x$.

## TYPES OF TM :

Various types of TM's are

(i) Turing machine with two dimensional tapes

(ii) Turing machines with multiple tapes

(iii) Turing machines with multiple heads

(iv) Turing machines with infinite tape

(v) Non deterministic turing machines

- computationally all these TMs are equally powerful. That means one type can compute the same that other can.

- However the efficiency of computation may vary.

(1) TM with two-dimensional tapes

This type of TM has two finite controls

1. Read/write head
2. Two dimensional tape.

- This tape has infinite extension to right and down.
- It is divided into small squares formed due to corresponding rows and columns.

- TM with one dimensional tape is equally powerful to that of two dimensional tape j →

i ↓

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 6 | 7 | 15 | 16 | – |
| 3 | 5 | 8 | 14 | 17 | 26 | – |
| 4 | 9 | 13 | 18 | 25 | – | – |
| 10 | 12 | 19 | 24 | – | – | – |
| 11 | 20 | 23 | -- | – | -- | – |
| 21 | 22 | – | – | – | – | – |
| – | – | – | -- | -- | – | – |

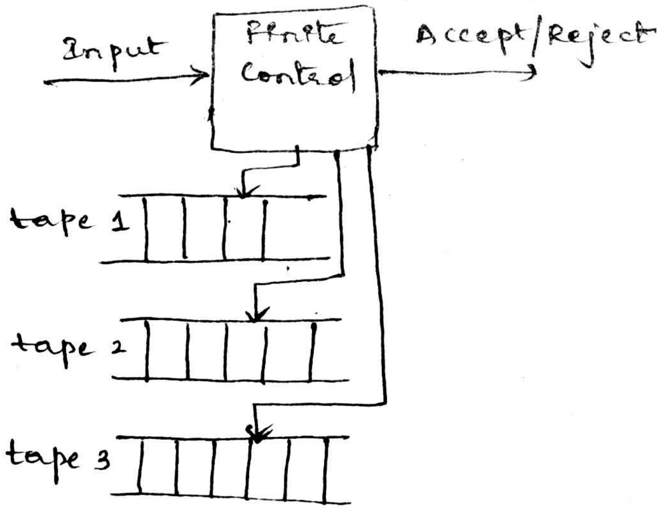
Two dimensional tape

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| j | 1 | j | 2 | 3 | i | 4 | 5 | 6 | j | 7 | 8 | 9 | 10 | i | 11 | — |

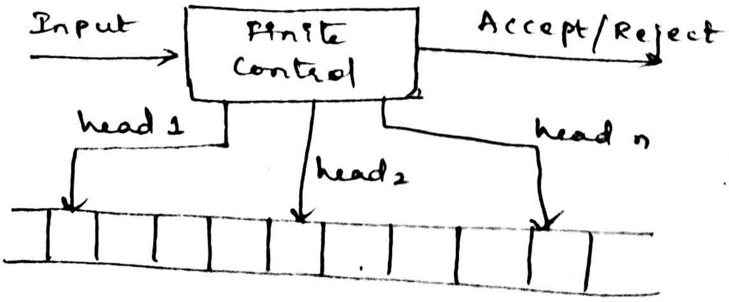
The head of two dimensional tape moves one square up, down, left or right.

2) Turing machine with multiple tapes:

This is a kind of TM with one finite control and with more than one tape having its own read/write heads.



3) Turing machine with multiple heads

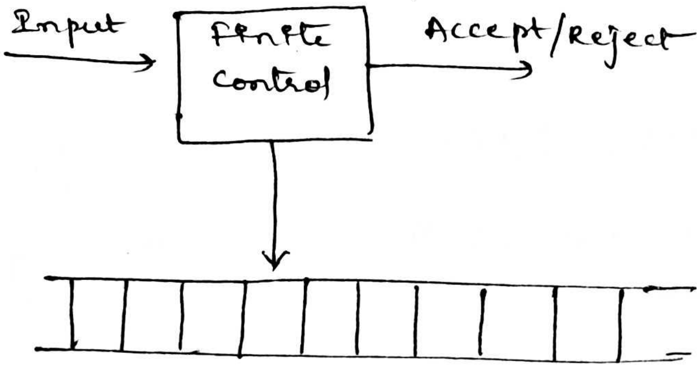


There are n heads, but in any state, only one head can move.

This type of TMs are as powerful as one tape TMs.

(4) Turing machines with infinite tape

This is a TM that have one finite control and one tape which extends infinitely in both directions.



This type of TMs are as powerful as one tape TMs whose tape has a left end.

5) Non deter ministic turing machines

The concept of non deterministic TM is similar to the NFA.

- For any state and any i/p symbol it can take any action from a set rather than a definite predetermined action.

For Example the languages like $L = \{ww \mid w \in (a+b)^*\}$ can be shown by non deterministic TM.

- The non deterministic TM is as powerful as deterministic TM.

# UNIT- 5

# COMPUTABILITY THEORY

Introduction:

A mathematical problem is said to be computable if it can be solved by a computing device.

- The computable problems are also called as "solvable", decidable or "recursive".
- In 1930's Godel, Turing and church showed that there are some mathematical problems which are not solvable.
- Around 1936, Turing and church independently designed a computing machine (called Turing machine) which performs computations using some algorithm.
- But there are some problems which are not computable.
- such problems are said to be "unsolvable" or "undecidable".

Thus in simple words we can state that the class of problems which can be answered as 'yes' are called solvable or decidable on the other hand the class of problems that can be answered as no are called unsolvable or undecidable

## Chomsky's Hierarchy

The chomsky's hierarchy represents the class of languages that are accepted by diff$^t$ machine.

- The category of languages in chomsky's hierarchy is given below.

| Language class | Language | Grammar | Machine | Eg: |
|---|---|---|---|---|
| Type 3 | Regular | Regular grammar | NFA or DFA | $a^*b^*$ |
| Type 2 | Context free | Context free grammar | PDA | $a^n b^n$ |
| Type 1 | Decidable Languages | Context sensitive grammar | LBA | $a^n b^n c^n$ |
| Type 0. | Computable languages | Un restricted grammar | TM | $n!$ |

Type 3 : Regular languages

- Regular languages are those languages which can be described using regular expressions.
- These languages can be modelled by NFA or DFA.

Type 2 : context free languages

Context free languages are the languages which can be rep$^{td}$ by context free grammar.

The prod$^{n}$ rule is of the form

$A \longrightarrow \alpha$

Where A is any single non-terminal and
$\alpha$ is any comb$^{n}$ of terminals and non terminals

- A NFA or DFA cannot recognize strings of this language because these automata are not having "stack" to memorize. Instead of it the PDA can be used to represent these languages.

## Type 1: Context sensitive languages

The context sensitive grammars are used to represent context sensitive languages.

The CSG follows the following rules :-

1) The CSG may have more than one symbol on the left hand side of their prodn rules.

2) The no. of symbols on the left hand side must not exceed the no. of symbols on the right hand side.

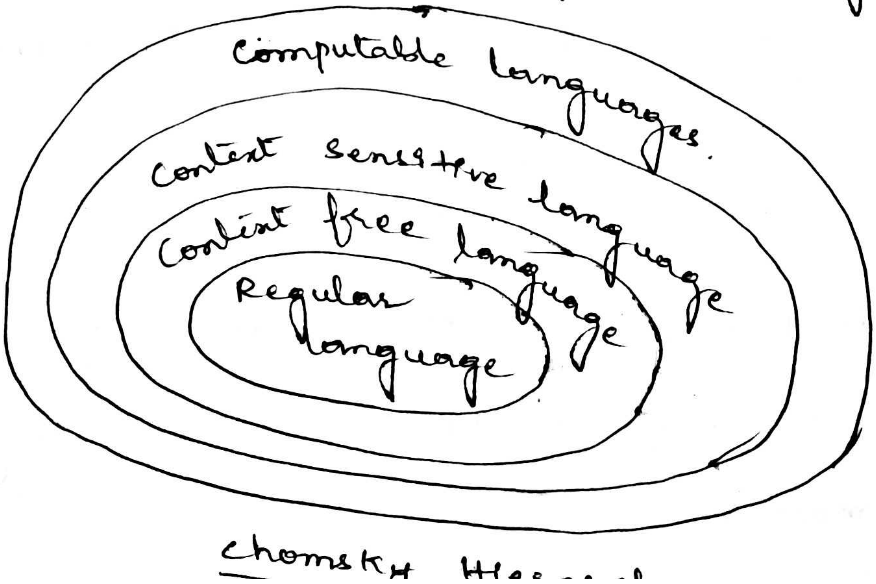
3) The rule of the form A → ε is not allowed unless A is a start symbol.

- The automaton which recognizes CSLs is called linear bounded automaton.

## Type 0: Un restricted languages

There is no restriction on the grammar rules of these type of languages.

- These languages can be effectively modeled by TMs.



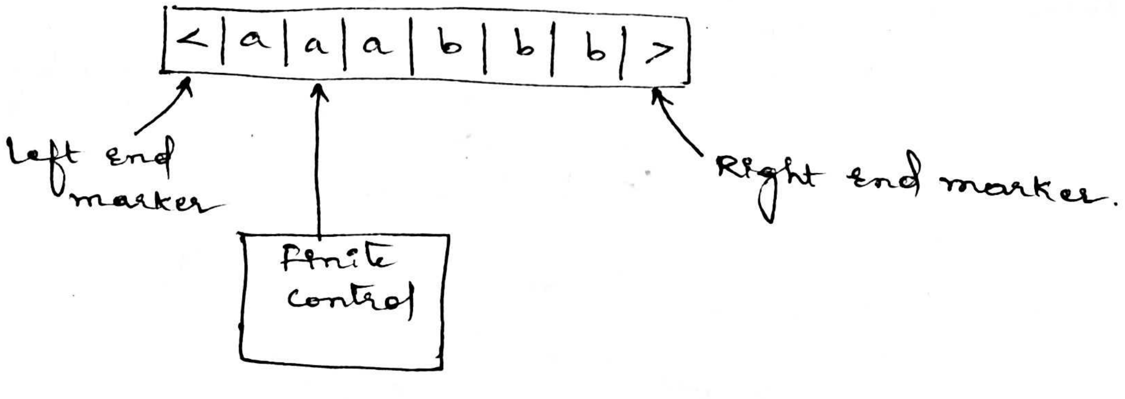
chomsky Hier

## Linear Bounded Automata

The LBA is a model which was origonally developed as a model for actual computers rather than model for computational process.

- A LBA is a restricted form of non deterministic TM.
- A Linear bounded automaton is a multi track TM which has only one tape and this tape is exactly of same length as that of i/p.

- The LBA accepts the string in the sim manner as that of TM does. For LBA halting means accepting. In LBA computation is restricted to an area bounded by length of the i/p. This is very much sim to programming envt where size of variable is bounded by its data type.



- LBA is powerful than NPDA but less powerful than TM.
- The i/p is placed on the i/p tape with beginning and end markers.

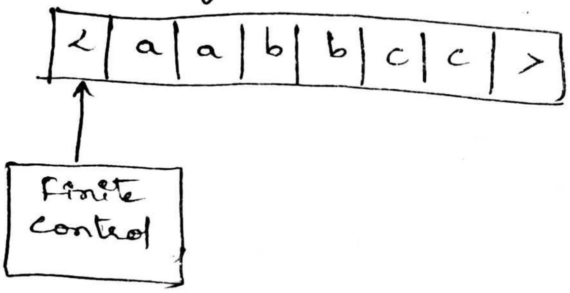
A LBA com be formally defined as -

It is 7 - tuple non deterministic TM with

$M = (Q, \Sigma, T, \delta, q_0, q_{accept}, q_{reject})$ having

1. Two extra symbols of left end marker and right end marker which are not elements of $T$.
2. I/p lies b/w these end markers.
3. The TM cannot replace < or > with anything else It cannot move the tape head left of < or right of >.

For eg: We can construct a language $L = \{a^n b^n c^n / n \geq 1\}$ using LBA as follows.



The input is placed on the input tape which is enclosed within left end marker and right end marker.

Simulation: consider input aabbcc

< a a b b c c > move right
↑

< a a b b c c > convert to A, move right
  ↑

< A a b b c c > move right
    ↑

< A a b b c c > convert to B, move right
      ↑

< A a B b c c > move right
        ↑

< A a B b c c > convert to C, move left.
          ↑

$< A\ a\ B\ \underset{\uparrow}{b}\ C\ c >$ move left

$< A\ a\ \underset{\uparrow}{B}\ b\ C\ c >$ move left

$< A\ \underset{\uparrow}{a}\ B\ b\ C\ c >$ move left

$< \underset{\uparrow}{A}\ a\ B\ b\ C\ c >$ move right

$< A\ \underset{\uparrow}{a}\ B\ b\ C\ c >$ convert to A, move right

$< A\ A\ \underset{\uparrow}{B}\ b\ C\ c >$ move right

$< A\ A\ B\ \underset{\uparrow}{b}\ C\ c >$ Convert to B, move right

$< A\ A\ B\ B\ \underset{\uparrow}{C}\ c >$ move right

$< A\ A\ B\ B\ C\ \underset{\uparrow}{c} >$ convert to C, move left

$< A\ A\ B\ B\ \underset{\uparrow}{C}\ C >$ move left till A

$< A\ \underset{\uparrow}{A}\ B\ B\ C\ C >$ move right

$< A\ A\ \underset{\uparrow}{B}\ B\ C\ C >$ move right

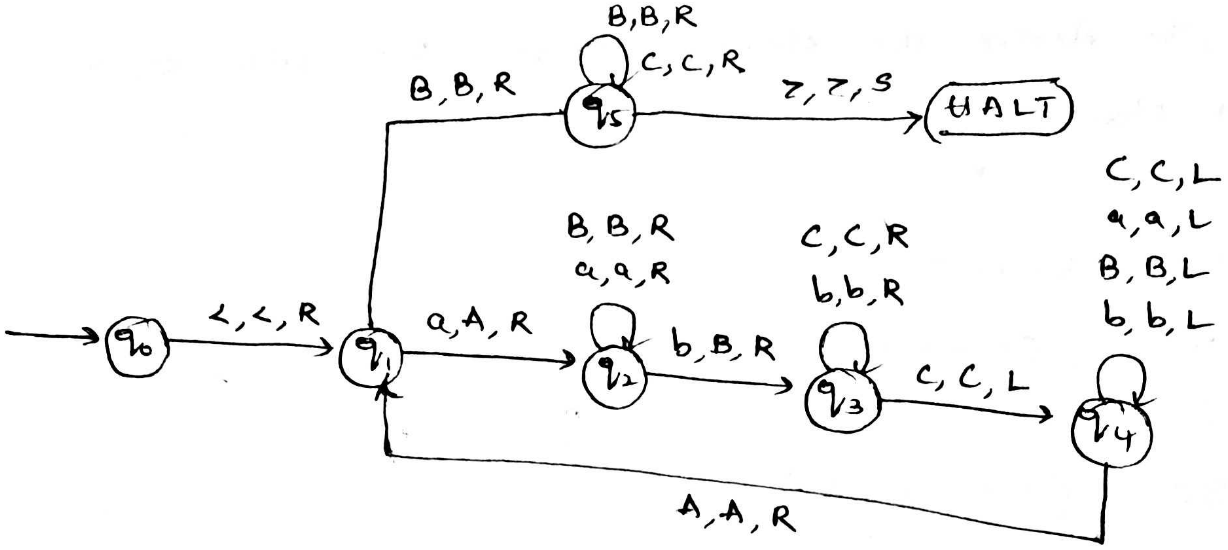
$< A\ A\ B\ \underset{\uparrow}{B}\ C\ C >$ move right

$< A\ A\ B\ B\ \underset{\uparrow}{C}\ C >$ move right

$< A\ A\ B\ B\ C\ \underset{\uparrow}{C} >$ move right

$< A\ A\ B\ B\ C\ C\ \underset{\uparrow}{>}$ We got right end marker '>' then we HALT by accepting the input aabbcc.

Thus in LBA length of tape is exactly equal to the i/p string.



## CONTEXT SENSITIVE LANGUAGES :

The context sensitive languages are the languages which are accepted by LBA.

- These type of languages are defined by CSG.
- In this grammar more than one terminal or nonterminal symbol may appear on the left hand side of the prod$^n$ rule.

The classic example of context sensitive language is $L = \{a^n b^n c^n / n \geq 1\}$

The context sensitive grammar can be written as.

$S \rightarrow aBC$
$S \rightarrow SABC$
$CA \rightarrow AC$
$BA \rightarrow AB$
$CB \rightarrow BC$
$aA \rightarrow aa$
$aB \rightarrow ab$
$bC \rightarrow bc$
$bB \rightarrow bb$
$cC \rightarrow cc$

Now, to derive the string aabbcc we will start from start symbol.

$S$

$\underline{S}ABC \quad (S \to aBC)$

$aB\underline{CA}BC \quad (CA \to AC)$

$aBA\underline{CB}C \quad (CB \to BC)$

$a\underline{BA}BCC \quad (BA \to AB)$

$\underline{aA}BBCC \quad (aA \to aa)$

$a\underline{aB}BCC \quad (aB \to ab)$

$aa\underline{bB}CC \quad (bB \to bb)$

$aab\underline{bC}C \quad (bC \to bc)$

$aabb\underline{cC} \quad (cC \to cc)$

$aabbcc$

## DECIDABILITY OF PROBLEMS:

A problem is said to be decidable if there exists some algorithm which solves the problem.

- If no such algorithm exists then the problem is said to be un decidable.

- Generally, if the problem is recognized by a TM then it is called decidable problem otherwise they are called undecidable problems.

- If a language is recursive then it is called decidable language otherwise it is called undecidable.

## Undecidable problem about TM:

1) Reduction
2) Empty and NonEmpty Language.
3) Rice's Theorem.

## REDUCTION:

Reduction is a technique in which, if a problem P1 is reduced to a problem P2 then any solution of P2 solves P1.

- In general, if we have an algorithm to convert an instance of a problem P1 to instance of a problem P2 that have the same answer, then it is called as P1 reduces P2.
- Hence if P1 is not recursive, then P2 is also not recursive.
- If P1 is not recursively enumerable, then P2 is also not recursively Enumerable.

Theorem: If P1 is reduced to P2 then,

i) If P1 is undecidable then P2 is also undecidable

ii) If P1 is non R.E then P2 is also non RE

Proof: (i) Consider an instance w of P1. Then construct an algorithm such that the algorithm takes instance w as input and converts it into another instance x of P2.

- Then apply that algorithm to check whether x is in P2. If algorithm answers "yes" then that means x is in P2. IIIrly we can also say that w is in P1.

Since we have obtained P2 after reduction of P1.

$\text{111}^{rly}$ if algorithm answers "no" then x is not in P2, that also means w is not in P1.

This proves that if P1 is undecidable then P1 is also undecidable.

ii) We assume that P1 is non-RE but P2 is RE.

Now construct an algorithm to reduce P1 to P2, but by this algorithm P2 will be recognized.

That means there will be a TM that says "yes" if the i/p is P2 but may or may not halt for the i/p which is not in P2.

- Apply a TM to check whether x is in P2, if x is accepted that also means w is accepted.
- This procedure describes a TM whose language is P1 if w is in P1 then x is also in P2 and if w is not in P1 then x is also not in P2.

This proves that if P1 is non RE then P2 is also non-RE.

## (ii) Empty and Non Empty Language:

There are two types of languages empty and non-Empty.

Let $L_e$ denotes an empty language and $L_{ne}$ denotes non. Empty language.

Let w be a binary string and $M_i$ be a TM.

If $L(M_j) = \phi$ then $M_i$ does not accept any input then w is in $L_e$.

Illrly if $L(M_i)$ is not the empty language then w is in $L_{ne}$. Thus we can say that

$$L_e = \{M \mid L(M) = \phi\}$$

$$L_{ne} = \{M \mid L(M) \neq \phi\}$$

Both $L_e$ and $L_{ne}$ are complement of one another.

Theorem: $L_{ne}$ is recursively enumerable.

Proof: To prove this, We simply need to prove that there exists some TM which accepts $L_{ne}$.

To do this We need to construct non deterministic TM M that can be converted to deterministic TM.

The Turing machine M accepts another TM $M_i$ as i/p. With the non deterministic capability M can guess the i/p w that can be accepted by $M_i$ if $M_i$ accepts w then M also accepts the input $M_i$.

Thus if at all $M_i$ accepts even one i/p, the M will guess that string and will accept $M_i$.

But if $L(M_i) = \phi$ that means M can make no guess about the input string and therefore cannot accept $M_i$.

This proves that there should be such a TM whose language $L(M) \neq \phi$, thus $L(M) = L_{ne}$.

## (3) RICE'S THEOREM:

Theorem: Every non trivial property of RE language is undecidable.

Proof: property of a language is a set of strings for which the Turing machines can be drawn which ultimately represents certain class.

For eg: a property of "context free language" in TM contains all the codes for the TM M such that all the L(M) are context free languages.

- Let P be any non trivial property of RE languages. That means atleast one language has the property and atleast one language does not.

- Initially we will assume $\phi$ an empty language is not having the property P.

- As P is the non trivial, there should be some language L having property P ie $L = L_p$ and therefore there exists a coded TM accepting language L.

Let us call such TM as $M_L$. Let us reduce a Universal language $L_u$ to $L_p$ and as $L_u$ is undecidable $L_p$ is also undecidable.

- The Turing machine M' can be produced on reduction. for the reduction algorithm the pair (M, w) can be given as input.

- Basically M' is a two tape Turing machine such that

- One tape is used to simulate TM M for the i/p w. Then input (M, w) is used for designing the transitions of M'.

- The second tape is used to simulate $M_L$ on input x. Again a pair (M', x) is used to build the transitions of M'.

The TM M' does the following things -

1) The TM M is simulated for the i/p w. Then TM M' acts as a Universal Turing machine by accepting the pair (M, w).

2) There exists two cases either M accepts w or M does not. If M accepts w then M' does nothing else.
M' cannot have its own input x, in such case $L(M') = \phi$.
But as we have assumed $\phi$ is not in property P that also means code for M' is not in $L_p$.

3). If M accepts w then M' simulates $M_L$ for its own input x and hence M' actually accepts the language of $M_L$.

From above three steps we cannot decide whether code for M' is in $L_p$.

This proves that property P is undecidable.

## DECIDABILITY OF CFGs

The questions about undecidable context free languages are

1) whether or not two diff^t CFLs define the same lang.
2) Whether given CFL is ambiguous or not?
3) Whether complement of given CFL is context free lang?
4) Whether the intersection of two context free languages is context free?

There is no algorithm to answer these questions.

But there are a certain set of questions for which the answers can be obtained. Such questions are about context free languages that are decidable. following is a list of those questions -

1) Does the CFG G generates any string (emptiness)
2) Is the language generated by G is finite or not? (finiteness)
3) Suppose G is a grammar and W is a string does G generates W? (membership).

## EMPTINESS :

Theorem: There exists an algorithm which can determine whether or not the given CFG can generate any word at all.

proof: The algorithm to determine whether or not a grammar generates a word is given below -

i) Put the dot (·) above every terminal symbol. If there is any null string ε then put dot over ε as well. These symbols occur at right hand side of prod$^n$ rule.

ii) If all the symbols of right hand side of the prod$^n$ rule are dotted then dot the corresponding non terminal (LHS) throughout the grammar.

Repeat this step as long as possible.

(iii) If the starting symbol gets dotted then answer is yes otherwise answer is no.

Let us apply this algorithm on some grammar

Let,

$S \rightarrow PQ$

$P \rightarrow AP \mid AA$

$A \rightarrow a$

$Q \rightarrow BQ \mid BB$

$B \rightarrow b.$

Now if we want to put dots we will put over terminals at RHS of rule.

step 1:

1) $S \rightarrow PQ$

2) $P \rightarrow AP \mid AA$

3) $A \rightarrow \dot{a}$

4) $Q \rightarrow BQ \mid BB$

5) $B \rightarrow \dot{b}$

step 2:

As the RHS of rule 3) and RHS of rule 5) is dotted, we will dot A and B throughout the grammar.

1) $S \rightarrow PQ$

2) $P \rightarrow \dot{A}P \mid \dot{A}\dot{A}$

3) $\dot{A} \rightarrow \dot{a}$

4) $Q \rightarrow \dot{B}Q \mid \dot{B}\dot{B}$

5) $\dot{B} \rightarrow \dot{b}$

Now in rule 2) the alternate prod$^n$ at RHS is dotted ($\dot{A}\dot{A}$) Hence dot P.

Similarly dot Q

Hence,

1) $S \rightarrow \dot{P}\dot{Q}$

2) $\dot{P} \rightarrow \dot{A}\dot{P} \mid \dot{A}\dot{A}$

3) $\dot{A} \rightarrow \dot{a}$

4) $\dot{Q} \rightarrow \dot{B}Q \mid \dot{B}\dot{B}$

5) $\dot{B} \rightarrow \dot{b}$

Thus in rule 1) the RHS is dotted.

Hence dot S. But since S being dotted is a start symbol, the algorithm generates answer yes.

- Hence we can prove that an algorithm exists which determines whether or not the given grammar generates any string.

## FINITENESS :

**Theorem:** There exists an algorithm to determine whether the given context free grammar generates a finite or infinite language.

**proof:** We will write following algorithm.

1). Make the grammar simplified by removing unit prod$^{ns}$ and useless symbols.

2) Check whether there is any self embedded non terminal or not by following steps.

a) underline some non terminal say x which is at LHS of prod$^{n}$ rule.

b) Then put dot over all the x which are at RHS throughout the grammar.
The dotted x and underlined x are treated as two diff$^{t}$ symbols.

c) Put dot over any non terminal at LHS whose RHS contains any dotted symbols. Dot this non terminal throughout the grammar.

d). If underlined x gets dotted then x is called self embedded otherwise it is not self embedded.

3) If the grammar contains any self embedded non terminal then it generates infinite languages otherwise it generates finite languages.

Let us take one grammar

$S \rightarrow PQa \mid bPZ \mid b$ $\qquad$ $Z \longrightarrow ZPbP$

$P \longrightarrow Xb \mid bZa$

$Q \longrightarrow bPP$

$X \longrightarrow aZa \mid aaa$

The above grammar has Z as useless symbol.

Hence by removing it we get,

1) $S \to PQa \mid b$

2) $P \to Xb$

3) $Q \to bPP$

4) $X \to aaa$

Now we will mark X of rule 4) underlined and put dot over X in rule 2)

1) $S \to PQa \mid b$

2) $P \to \dot{X}b$

3) $Q \to bPP$

4) $\underline{X} \to aaa$

As RHS of rule 2) contains one dotted symbol, dot P throughout the grammar.

$S \to \dot{P}Qa \mid b$

$\dot{P} \to \dot{X}b$

$Q \to b\dot{P}\dot{P}$

$\underline{X} \to aaa$

Then Q gets dotted.

$S \to \dot{P}\dot{Q}a \mid b$

$\dot{P} \to \dot{X}b$

$\dot{Q} \to b\dot{P}\dot{P}$

$\underline{X} \to aaa$

Then S gets dotted.

$\dot{S} \to \dot{P}\dot{Q}a \mid b$ $\quad$ $\underline{X} \to aaa$

$\dot{P} \to \dot{X}b$

$\dot{Q} \to b\dot{P}\dot{P}$

Now all the non terminals are dotted but X is not.

That means X is not self embedded. symbol.

Hence the grammar generates finite language

MEMBERSHIP:

There exists an algorithm which tells whether given string belongs to given grammar.

- To check whether the given grammar generates desired string the derivation tree can be drawn.

Consider the grammar.

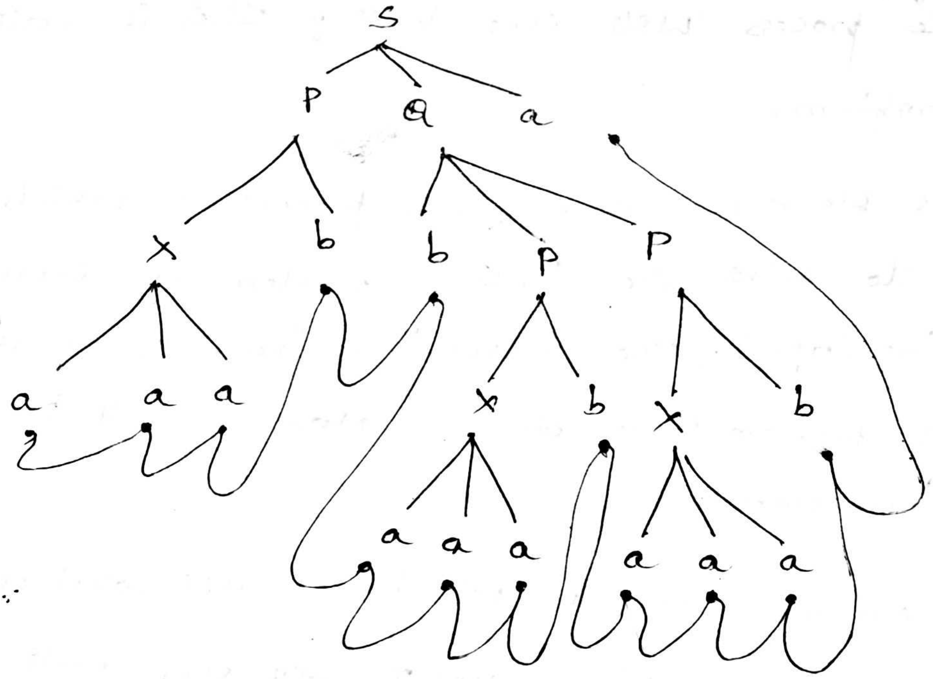
$S \rightarrow PQa \mid b$

$P \rightarrow Xb$

$Q \rightarrow bPP$

$X \rightarrow aaa$

check whether it generates string aaabbaaabaaaba



The string generated is aaabbaaabaaaba.

HALTING PROBLEM:

- This is a famous undecidable problem of TM.
- To state halting problem we will consider the given configuration of a TM.
- The o/p of TM can be

(i) Halt : The machine starting at this configuration will halt after a finite number of states.

(ii) No Halt : The machine starting at this configuration never reaches a halt state, no matter how long it runs.

Now the question arises based on these two observation: given any functional matrix, i/p data tape and initial configuration, then is it possible to determine whether the process will ever halt ? This is called halting problem.

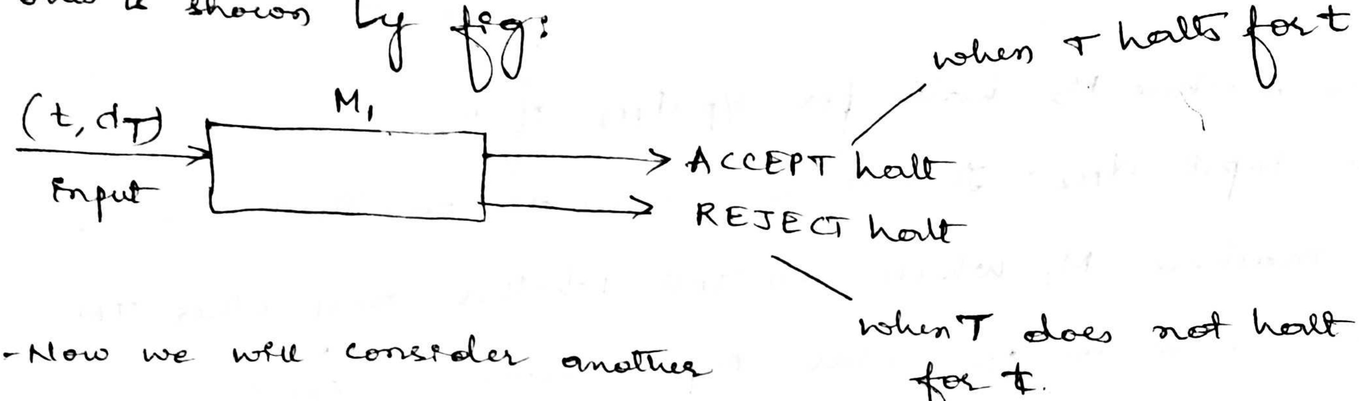
That means we are asking for a procedure which enable us to solve the halting problem for every pair (machine, tape). The answer is "no". That is the halting problem is unsolvable. Now we will prove why its is unsolvable.

- Let, there exists a TM $M_1$ which decides whether or not any computation by a TM T will ever halt when a description $d_T$ of T and tape t of T is given. [Input to machine $M_1$ will be (machine, tape)]

Then for ~~any~~ every i/p $(t, d_T)$ to $M_1$ if T halt for input t, M, also halts which is called accept halt.

111ly if T does not halt for i/p t then M, will halt which is called reject halt.

This is shown by fig:

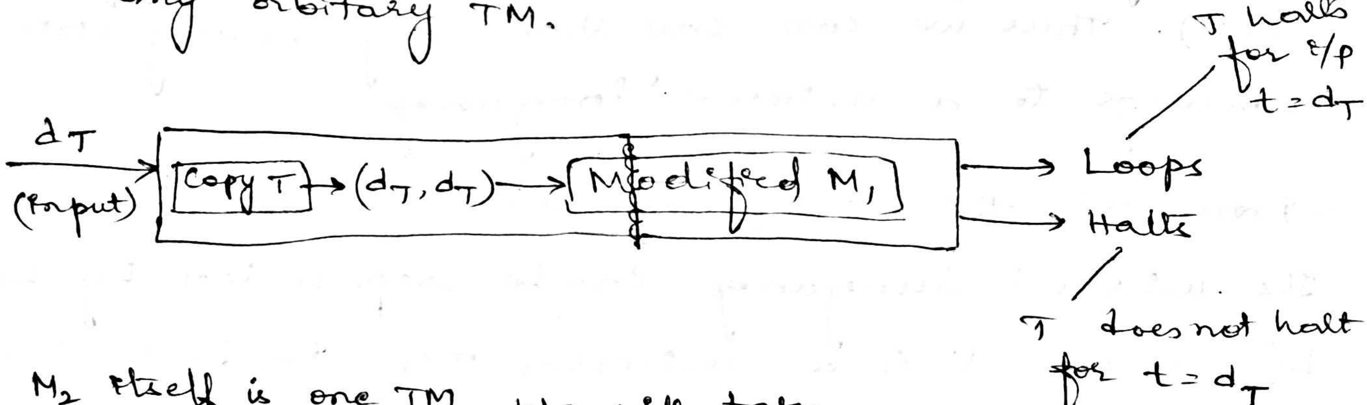


- Now we will consider another TM $M_2$ which takes an input $d_T$.

It first copies $d_T$ and duplicates $d_T$ on its tape and then this duplicated tape of is given as i/p to machine $M_1$.

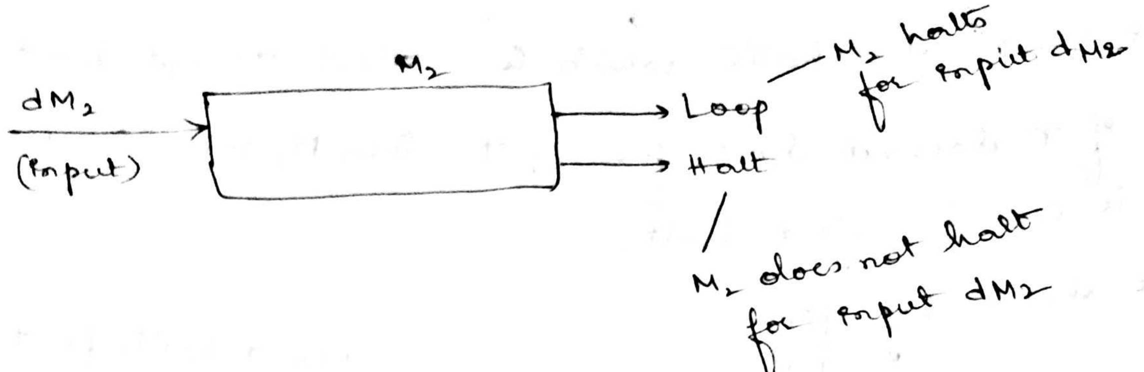
But machine $M_1$ is a modified machine with the modification that whenever $M_1$ is supposed to reach an accept halt, $M_2$ loops forever. Hence behavior of $M_2$ is as given. It loops if T halts for i/p $t = d_T$ and halts if T does not halt for $T = d_T$.

- T is any aribitary TM.



As $M_2$ itself is one TM we will take $M_2 = T$.

That means we will replace T by $M_2$ from above given machine.


$M_2$
$dM_2$
(Input)
Loop
Halt
$M_2$ halts for input $d_{M2}$
$M_2$ does not halt for input $dM_2$

Thus machine $M_2$ halts for i/p $d_{M2}$ if $M_2$ does not halt for input $d_{M2}$. This is a contradiction. That means a machine $M_1$ which can tell whether any other TM will halt on particular input does not exist.

- Hence halting problem is unsolvable.

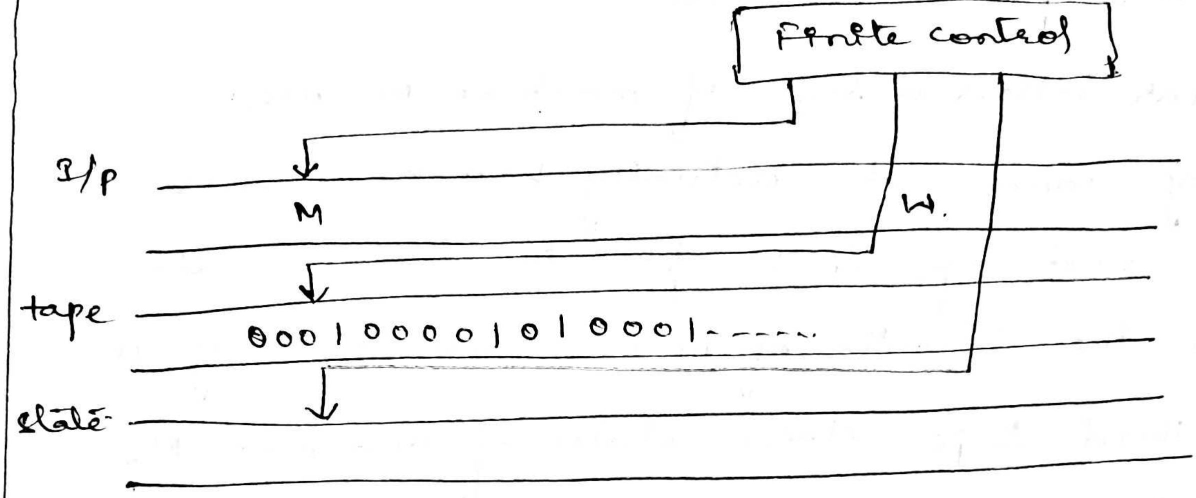
UNIVERSAL TM :

The universal language Lu is a set of binary strings which can be modeled by a TM. the universal language can be represented by a pair (M, w) where M is a TM that accepts this language and w is a binary string in $(0+1)^*$ such that w belongs to L(M). Thus we can say that any binary string belongs to a universal language.

- From this the concept of universal TM come up. The universal language can be represented by Lu = L(U) where U is a universal TM. In fact U is a binary string. This binary string represents various codes of many TMs.

- Thus the universal TM is a TM which accepts many TMs.

The TM U is a multitape TM which is shown below.



- The universal TM U accepts TM M.
- The transitions of M are stored initially on first tape along with the string w.
- On the second tape the simulated tape of M is placed. Here the tape symbol $X_i$ of M will be repd by $0^i$ and tape symbols are seperated by single 1's.
- On the third tape various states of machine M are stored. The state $q_i$ is represented by i 0's.

Operations on TM:

1. The finite tape is observed carefuley to check whether the code for M is valid or not.

If the code is not valid then U halts without accepting. As invalid codes are assumed to represent the TM with no moves, such TM accepts no inputs and halts.

2. Initialize the second tape for placing the tape values of M is encoded form that means for each

of the i/p w place 10 on second tape and for each 1 place 100 there.

3. The blank letters of tape of machine M are rep^td by 1000. But actually blank appears at the end of i/p w for the machine U. However the blanks of M are simulated by 1000.

4. As the third tape stores states of machine M, place 0 there as the start state of machine M. The head of third tape will now position at start state (ie at 0) and the head of third tape will now position at start state (ie at 0) and the head for the tape 2 will position at the first simulated cell of 2nd tape.

5. Now the move of M can be simulated. As on the first tape transitions are stored U first searches the tape 1 to know the transitions which are always given in the form

$0^i 10^j 10^k 10^l 10^m$ .

$0^i$ means current state

$0^j$ means current i/p symbol.

$0^k$ means changed state.

$0^l$ means changed i/p tape symbol.

$0^m$ means direction in which the head is to be moved.

Now to simulate the move of M change the content of tape 3 to $0^k$ that is, simulate the state change of

M. Then replace $0^i$ on tape 2 by $0^l$, that is simulate the change in i/p. Then move the tape head on tape to the position of next 1. The tape head moves to left or right depending upon the value of m. that means if m=1 then move left and if m=2 move right. Thus U simulates the move of M to left or to the right.

6. If M has no transition then no transition is performed. Therefore M halts for the simulated configuration and hence U halts.

7. If M enters in accept state, then U accepts.

## Post's Correspondence Problem:

The undecidability of strings is determined with the help of Post's Correspondence problem (PCP).

The PCP consists of two lists of strings that are of equal length over the input Σ.

- The two lists are $A = w_1 w_2 w_3 \dots w_n$ and

$B = x_1 x_2 x_3 \dots x_n$

then there exists a non empty set of integers $i_1, i_2, i_3 \dots i_n$ such that $w_1 w_2 w_3 \dots w_n = x_1 x_2 x_3 \dots x_n$.

Ex(1): Consider the correspondence system as given below. $A = (1, 0, 010, 11)$ and $B = (10, 10, 01, 1)$ The i/p set is $\Sigma = \{0, 1\}$. Find the solution.

Soln: A solution is 1 2 1 3 3 4. That means

$w_1 w_2 w_1 w_3 w_3 w_4 = x_1 x_2 x_1 x_3 x_3 x_4$

The constructed string from both lists is

101 010 010 11

| | | | | | |
|---|---|---|---|---|---|
| $w_1$ | $w_2$ | $w_1$ | $w_3$ | $w_3$ | $w_4$ |
| 1 | 0 | 1 | 010 | 010 | 11 |
| $x_1$ | $x_2$ | $x_1$ | $x_3$ | $x_3$ | $x_4$ |
| 10 | 10 | 10 | 01 | 01 | 1 |

Ex(2): obtain sol$^n$ for the following correspondence system.

A = { ba, ab, a, baa, b } , B = { bab, baa, ba, a, aba }.

The i/p set { a, b }.

sol$^n$ $w_1 w_5 w_2 w_3 w_4 w_4 w_3 w_4 = x_1 x_5 x_2 x_3 x_4 x_4 x_3 x_4$

This sol$^n$ gives a unique string

bababaabaaabaa

Ex:(3) obtain the sol$^n$ for the following system of PCP.

A = { 100, 0, 1 } , B = { 1, 100, 00 }

sol$^n$ The sol$^n$ is 1 3 11 3 22

A1 A3 A1 A1 A3 A2 A2 = 1001100100100

B1 B3 B1 B1 B3 B2 B2 = 1001100 100 100

Ex:(4) Does PCP with two lists $x = (b, bab^3, ba)$ and $y = (b^3, ba, a)$ have a solution?

sol$^n$ sequence is 2 1 1 3

2 1 1 3. = 2 1 1 3

$bab^3\, b\, b\, b\, a$ = $ba\, b^3\, b^3\, a$

Ex: 5 Find whether the PCP $P = \{(10,101), (011, 11), (101, 011)\}$ has a match.

Give the solution.

Soln Let $P = \{(\underline{10}, 101), (\underline{011}, 11), (\underline{101}, 011)\}$

$W_1 = \{10, 011, 101\}$

$W_2 = \{101, 11, 011\}$.

The PCP has no solution.

## TURING REDUCIBILITY

Reduction is a technique in which if a problem A is reduced to problem B then any soln of B solves A.

- In general if we have an algorithm to convert some instance of problem A to some instance of problem B that have the same answer then it is called A reduces to B.

YES
NO
YES
NO

Reduction

Definition of Turing Reducibility:

Let A and B be the two sets such that

$A, B \subseteq N$ of natural numbers.

Then A is Turing reducible to B and denoted as

$A \leq_T B$.

- If there is an oracle machine that computes the characteristic function of A when it is executed with -

oracle machine for B.

This is also called as A is B-recursive and B-computable.

- The oracle machine is an abstract machine used to study decision problem.

- It is also called as Turing machine with black box.

- we say that A is Turing equivalent to B and write $A \equiv_T B$ if $A \leq_T B$ and $B \leq_T A$.

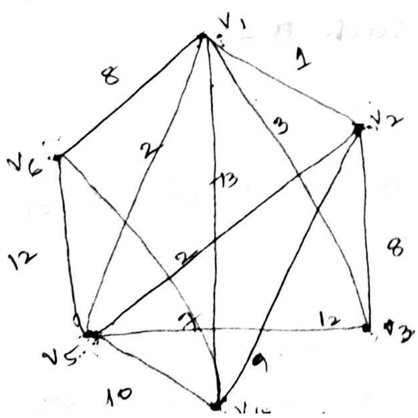
## The classes P and NP problems :

### The class of Languages P :

If the TM has some polynomial $P(n)$ and the machine M never makes more than $P(n)$ moves when an input of length n is given to M then such a TM is said to be polynomial time TM.

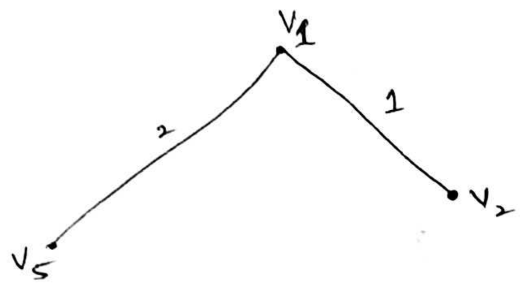
- The class P is set of languages accepted by polynomial-time TM.

For eg:- Kruskal's algorithm for minimum spanning tree.

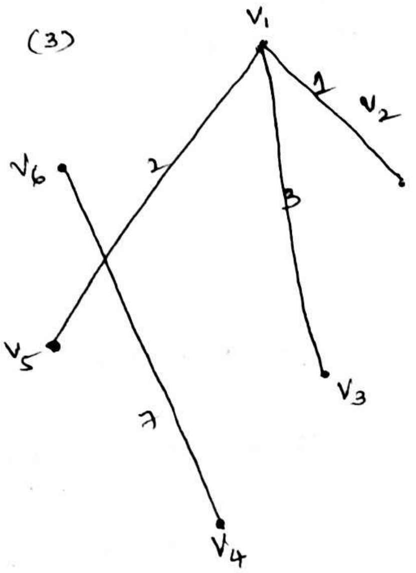


⇒

(2)



(3)



Weight = 13.

In Kruskal's Algorithm the minimum weight is obtained. In this algorithm also the circuit should not be formed. Each time the edge of minimum weight has to be selected, from the graph. It is not necessary in this algorithm to have edges of minimum weights to be adjacent. Let us solve one example by Kruskal's algorithm.

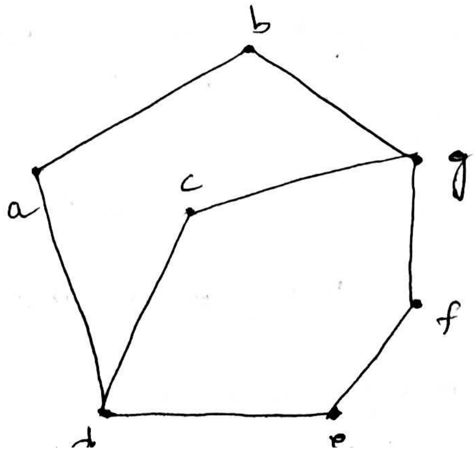
The class of Languages NP (NP complete and NP Hard problems).

If the non deterministic Turing Machine M has some polynomial P(n) and the machine M never makes more than P(n) moves when an input of length n is given to M then such a TM is said to be polynomial time non-deterministic Turing machine (There are many problems in NP which are not in P if we can't resolve P=NP question, We can atleast demonstrate that certain problems in NP are hardest, in the sense that if any one of them were in P, Then P=NP. such problems are also called as NP complete problems.

Travelling salesman's Problem:

Given a set of cities, and a cost to travel between each pair of cities, determine whether there is a path that visits every city once and returns to the first city, such that the cost travelled is less.

This problem is a NP problem as there may exist some path with shortest distance between the cities. If you get the solution by applying certain algorithm then Travelling salesman problem is NP complete problem. If we get no solution at all by applying an algorithm then that traveling salesman problem belongs to NP hard class.



If P is a class of problems which can be solved in polynomial time whereas NP is a class of problems for which it is not possible to determine whether they are solved in polynomial time ~~whereas NP is a class~~ of ~~problems for which it is~~ or not. Hence NP class problems can be modeled by Deterministic Turing Machines.

The NP class problems are of two kinds NP complete and NP hard problems. If for solving some problems there may be some algorithm existing then such set of problems belong to NP complete problem.

- On the other hand a problem is assigned to the NP class if it is solvable in polynomial time by a non deterministic, Turing machine.

—One can also say P⊆NP. A problem is NP-hard (Non deterministic Polynomial time Hard). If solving it in polynomial time would make it possible to solve all problems in class NP in polynomial time. The problem is called NP complete if it may or may not be solved in polynomial-time. Therefore NP problems are undecidable problems.

(6) PCP.

eg:- $A = (\underset{x_1}{abb}, \underset{x_2}{aa}, \underset{x_3}{aaa})$ $B = (\underset{y_1}{bba}, \underset{y_2}{aaa}, \underset{y_3}{aa})$

$x_2 x_1 x_3 = y_2 y_1 y_3$

$A = x_2 x_1 x_3$ $B = y_2 y_1 y_3$

$A = aaabbaaa$ $B = aaabbaaa$

$\therefore A = B.$

$\therefore$ PCP has a solution for this problem.