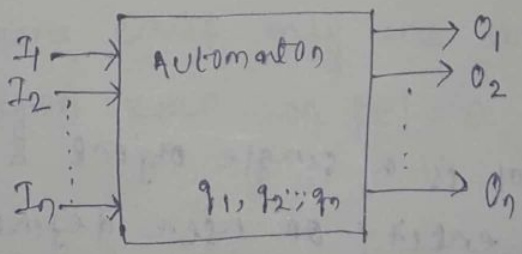
CHAPTER-1

INTRODUCTION

"Theory of computation" or "Automata Theory" is based on mathematical computations. Computation can be defined as finding a solution to the problem from the given inputs. These computations are used to represent various mathematical models. eg:- finite automata, pushdown automata, Turing Machine.

The word "Automata" is the plural of the word "Automaton", which means to automate or mechanize. Mechanization of a process means performing on a machine without human involvement.

Def":- An automaton is defined to be a system where energy, power or information are transformed or transmitted and used for performing some functions without direct participation of man.



Characteristics of Automaton / system:-

1) Input:- (Symbol)

At each of the discrete instant of time $t_1, t_2 \ldots t_n$, the input values $I_1, I_2 \ldots I_n$ each of which can take a finite no. of fixed

values from the input alphabet Σ.

(ii) Processing steps :-

└ State :- At any instant of time automaton can be in one of the states $q_1, q_2 \ldots q_n$. or

It represents the configuration of the system at any instant of time.

└ State Relation / State Transition :-

The next state of automaton at any instant of time is determined by the present state & present input.

(iii) Output (Symbol) :- $O_1, O_2 \ldots O_n$ are the finite no. of fixed values.

NOTE :-

At any instant of time the automaton is present in some state & on reading an i/p symbol, the automaton moves to a new state which is given by the state relation.

Symbol :-

A symbol is a single object & is an abstract entity or user defined entity that has no meaning by itself.

eg: letters, digits.

Alphabet :-

It's a finite, nonempty set of symbols. Conventionally 'Σ' is used for Alphabet.

eg: $\Sigma = \{0, 1\}$ → binary alphabet

$\Sigma = \{a, b \ldots z\}$ → set of all lower case letters.

STRING :-

A string (word) is a finite sequence/collection of symbols from alphabet.

eg:- 01001 is a string from the binary alphabet $\Sigma = \{0, 1\}$

Empty string :-

It is a string that consists of zero symbol & denoted by ϵ. It is also called as a null string.

Length of a string :-

The total number of symbols in a string is known as length of a string.

eg:- The string w011 has length 4.

→ The standard notation for the length of a string w is $|w|$.

→ The length of any string over the given alphabet is greater than or equal (>=) to 1

NOTE → There exist only one string 'ε' of length zero. or $|\epsilon| = 0$.

Prefix :-

Prefix of a string is any number of leading symbols of the string.

eg:- Prefix of "abc" are ε, a, ab, abc

[∵ abc is string itself]

Suffix :-

Suffix of a string is any number of trailing symbol of string.

eg:- Suffix of the string "abc" are

ϵ, c, bc, abc.

Substring :-

Any sequence of symbols over the given ~~str~~ string is called substring.

eg:- substring of the string "abc" are

ϵ, a, bc, ab, ac, abc.

Kleene closure :-

If Σ is a set of symbol of characters, then Σ^* or Kleene closure of Σ is the set of all string over the symbol Σ including ϵ.

eg:- if $\Sigma = \{a, b, c\}$

$\Sigma^* = \{\epsilon, a, b, c, ab, bc, ac, abc \ldots \text{asc} \ldots\}$

Power of an Alphabet :

if $\Sigma = \{0, 1\}$ where Σ : Alphabet set

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

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

$\Sigma^3 = \{000, 001, 010 \ldots 111\}$

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

Positive closure:-

If Σ is a set of symbols or characters then Σ^+ (positive closure) is set of all strings over symbol of Σ excluding empty string (ϵ).

or

The set of non-empty strings from alphabet Σ is denoted by Σ^+.

$\{a,b\}^+ = \{a, b, ab, aa, aaa, \ldots\}$

i.e NOTE: $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \ldots\ldots$

$\Sigma^* = \Sigma^+ \cup \Sigma^0$

$\Sigma^* = \Sigma^+ \cup \{\epsilon\}$

$\phi^+ = \epsilon$

Language:-

A language is defined as a set of string of symbols over an alphabet.

eg:- The set of strings of 0's & 1's with an equal number of each.

$\{\epsilon, 01, 10, 0011, 0101, 1001 \ldots\}$

→ ϕ : empty language.

→ ϵ : language consisting of the empty string.

Language of a Machine:-

Set of all accepted strings of the machine is the language of a machine.

Grammar:-

Set of rules used for generating strings of language over given alphabet is called as a grammar. (*)

or

Natural Language:

It's a medium of communication. It has some meaning (semantics).

eg:- Ram is a good boy.

Formal Language:-

It is also a collection of strings. Here the meaning of the string is not important but format of the string is important.

eg:- ab (one a followed by one b)

↓

format

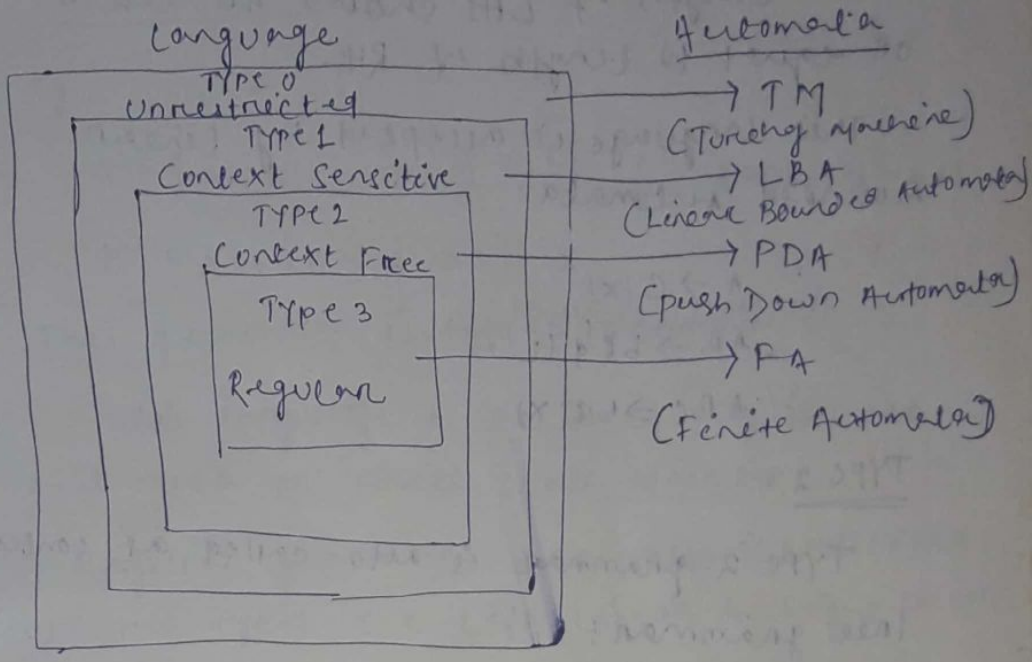
* Grammar is a set of formal rules which check the correctness of sentence.

But in this content, in automata theory A grammar will generate a string starting with a start symbol for a set of Nonterminal.

CHOMSKY HIERARCHY OF LANGUAGES

Chomsky defined 4 types of grammar i.e type 0, type 1, type 2, type 3.

The following picture shows the hierarchy of language & its relationship with automata / machine.



[: chomsky classification]

- **Type 0 :-**

Type 0 grammar is most general one & is called as unrestricted grammar / phrase structure grammar.

Rule :- $U \rightarrow V$

$U \in V^* N V^*$ &

$V = N \cup T$

N : Nonterminal

T : Terminal

This language is accepted by Turing Machine.

TYPE 1:-

Type 1 grammar is also called as context sensitive grammar or length increasing grammar.

Rule :- $U \rightarrow V$

but $|U| \leq |V|$

i.e length of LHS should be less than or equal to length of RHS.

This language is accepted by Linear Bounded Automata.

$A \rightarrow G(X)$
(length 1) (length 0)
$AB \rightarrow bCq(L)$
$ABC \rightarrow bq(X)$

TYPE 2:-

Type 2 grammar is also called as context free grammar: i.e

Rule :- $U \rightarrow V$

where $V = (N \cup T)^*$ & $U \in N$

LHS must be a single Nonterminal.

eg:- We know $G = \{V, T, P, S\}$

$V = \{S, A\}$, $T = \{a, b, c\}$ & S: Start symbol

$P = \{(S \rightarrow A), (A \rightarrow abS)\}$

So this grammar G is a type 2 or CFG.

This language is accepted by Push Down automata (PDA).

Type 3:-

Type 3 grammar is also called as regular grammar.

Rule:- The LHS is a non-terminal symbol & RHS contains atmost one non-terminal symbol which is the right most or left most symbol. This type of grammar is called as Regular grammar.

eg:- $G = \{V, T, P, S\}$

$V = \{A, B\}$, $T = \{a, b\}$, A: start symbol

$P: \{A \to aB, A \to b\}$

This grammar is type 3 grammar.

This language is accepted by Finite Automata or Finite state Machine.

Regular grammars are also classified into two types. i.e Left Linear & Right Linear.

Left Linear Grammar:

In a grammar if all productions are of the form $A \to BW$ or $A \to W$ i.e $A \to WB | W$

where A & B: variables

W: String of terminals.

Right Linear Grammar:

In a grammar if all productions are of the form $A \to WB | W$ then Right Linear grammar

eg:- $S \to 0A$

$A \to 10A | \epsilon$

$S \to 0A$

$\Rightarrow 0|0A \Rightarrow 010|0A \Rightarrow 0101|0\epsilon \Rightarrow 0101$

$\Rightarrow 0(10)^*$

Hence the language $0(10)^*$ is represented by Right linear grammar.

Expressive power of Automata (M)

The number of languages are accepted by M.

$E(FA) = 1$

$E(PDA) = 2$

$E(LBA) = 3$

$E(TM) = 4$

$E(FA) < E(PDA) < E(LBA) < E(TM)$

Language Acceptor:-

Language	Machine (Language Acceptor)
Regular Language	Finite Automata
Context Free Language	Push Down Automata
Context Sensitive Language	Linear Bounded Automata.
Recursively Enumerable Language	Turing Machine

FINITE AUTOMATA

Def^n:

The simplest kind of language recognition machine is the finite state machine.

→ Finite state machine or finite Automata is a mathematical model of a machine with finite/ discrete number of states and transitions.

→ Finite automata are computing device/ abstract machine that accept/ recognize the regular language, where input to the finite automata is a string of character & o/p is yes/ no.

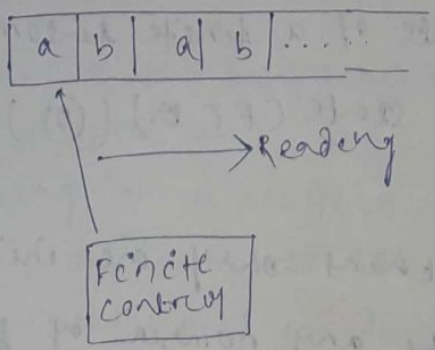
Components of Finite State Automata:

The components are

a) Finite control

b) Reading head

c) Input tape



The input tape is devided into number of cells. Each cell stores a single symbol from the alphabet Σ.

The Reading head reads only one cell at a time & the head moves to the right side with or without changing the state.

Finite control :- This unit performs the operation of finite automata which is represented by transition diagram in which vertices represent states & edges represent the transitions.

Definition of a finite Automata :-

(Mathematical Representation)

A finite Automata is a 5-tuple notation $M = (Q, \Sigma, \delta, q_0, F)$ where

Q : finite set of states in which the finite automata exists. ($\rightarrow$ Q)

Σ : finite set of i/p alphabet which can be fed to the finite automata. ⊗

δ : Mapping function (Transition function) which maps $Q \times \Sigma \rightarrow Q$. i.e it determines next state, given the current state of finite automata & the current i/p symbol.

q_0 : initial state of a finite Automata ($q_0 \in Q$)

F : Set of final state ($F \subseteq Q$) (◎)

NOTE :-

→ Always FA contains only one initial state.

→ There may be any number of final state in given automata.

without final state in automata, it's called as empty language.

(*) states are ~~with~~ labelled with circles & written in lowercase letters i.e q or s with numeric subscripts & also by capital letters.

State:-

Behaviour of the system at any instant of time is called as state / configuration of a system at any moment. (*)

Types:-

① Initial State:-

This is the starting / initial state of a finite state Machine (M).

→ The FA reaches this state without without consuming any character. To reach any other state finite automaton consumes a character.

→ A state with an arrow from free space (not coming from any other state) is designated as an initial state. (→○)

→ Always there's only one starting state / initial state in a Finite Automaton.

② Final State:-

Final state / Accepting state produces as usable result.

→ A string 'w' is accepted by the finite Automata M if, on complete consumption / reading of w, M reaches / halts in final state.

→ The final states are shown by double circle ◎ in transition diagram & encircled ○ state in the transition table.

Sometimes a star symbol (*) precedes the state to indicate that it's the final state. (*P : P is final state)

→ There may be any no. of final state in given automata.

③ Next state :-

The state that immediately follows the current state is next state.

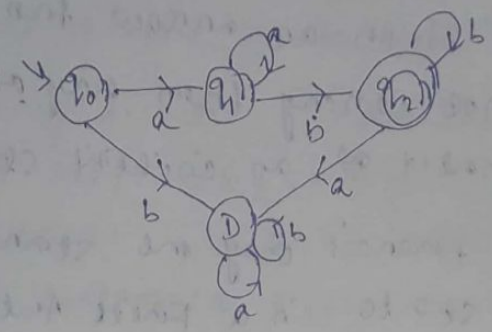
④ Dead / Trap state :-

They are nonfinal state of a finite state machine (M) whose transitions of every i/p symbol terminates on itself.

or

Once you reach this state, afterwards you can't go to any final state. So the string has to be rejected only. Such a state is known as a dead state.

eg:



Here

q_0 : initial state

q_2 : Final state

q_1 : Next state of q_0 &

q_2 is next state of q_1

D : Dead state.

Transition :-

The act of passing (changing) from one state to another state is known as transitions.

→ In finite automata there are different ways to represent transitions.

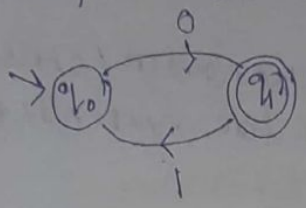
(1) Transition diagram / state diagram / transition system

(II) Transition Table / State table

(III) Transition function

Transition Diagram:

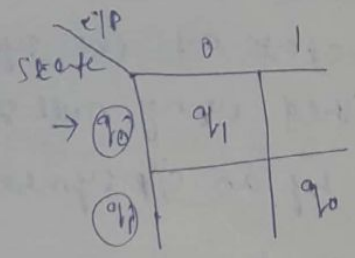
A transition diagram / transition system is a finite directed labelled graph in which each vertex (or node) represents a state & the directed edges indicates the transition of a state & the edges are labelled by i/p / output or input.



q_0 & q_1 : states

(II) Transition Table:-

It's the tabular representation of a state diagram where columns correspond to inputs & ~~each~~ rows correspond to states.



(III) Transition Function:-

The set of transition functions δ is the formal description of a finite automaton.

→ It shows how the automata will behave, given its current state & input symbol.

i.e Here $\delta(q_0, 0) = q_1$

$\delta(q_1, 1) = q_0$

Extended Transition function :-

If execution starts in any other state (other than the initial state) of the finite automaton & an i/p string is given at that state, then the behaviour of the string is defined by extended transition function.

i.e It describes the behaviour of the string when input at a state other than the initial state.

(or)

When more than one character / string are to be read, then we go for extended transition ~~system~~ function.

→ It is represented with $\hat{\delta}$ symbol. Here δ can also be used as extended transition function is a subcategory of transition function.

Properties :-

1) $\hat{\delta}(q_j, \epsilon) = q_j$

This indicates the state of the system can be changed / accomplished using null string ϵ. It can be changed only by an i/p symbol.

2) $\hat{\delta}(q_j, aw) = \hat{\delta}(\delta(q_j, a), w)$

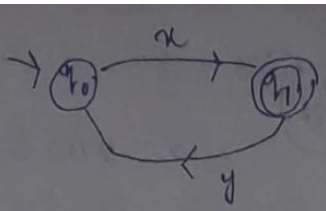
where a : input symbol

w : string of characters

This property indicates that a transition function is performed character by character in the sequence given in the input string.

eg: $\hat{\delta}(q, xy) = \hat{\delta}(\hat{\delta}(q, x), y)$

$\hat{\delta}(q, yx) = \hat{\delta}(\hat{\delta}(q, y), x)$



from above fig:

$\delta(q_0, xy) = \delta(\delta(q_0, x), y)$

$= \delta(q_1, y)$ [$\because \delta(q_0, x) = q_1$]

$= q$

Acceptability of a string by Finite Automaton:

(i.e $w \in \Sigma^*$)

If there exist a path which originates from some initial state, goes along the arrows & terminates at some final state, then the string is accepted by Finite Automaton.

i.e A string 'w' is accepted by a finite Automaton M

if $\delta(q_0, w) = q$ for some $q \in F$.

$L(M) = \{w \mid \delta^*(q_0, w) \in F\}$

eg: Consider a string xyx & check whether it's accepted by the above (fig) FA.

$\delta(q_0, xyx) \vdash \delta(q_1, yx)$

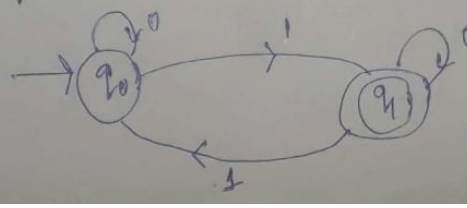
$\vdash \delta(q_0, x)$

$\vdash q_1$

∴ The given string 'xyx' is accepted by above FA because the transition is ending with the final state.

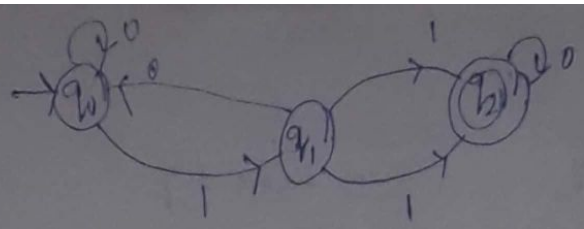
Q. Find out $Q, \Sigma, \delta, q_0, F$ & also check whether the given string is accepted or not from following Diagrams (FAs)

1.



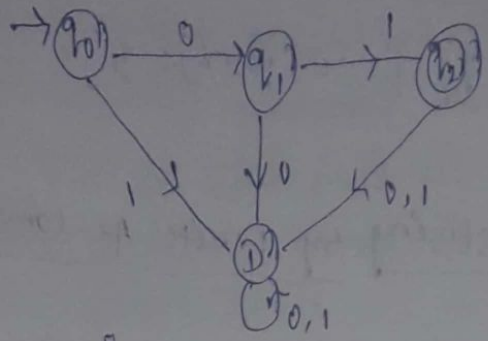
w = 01011 & 101

2)



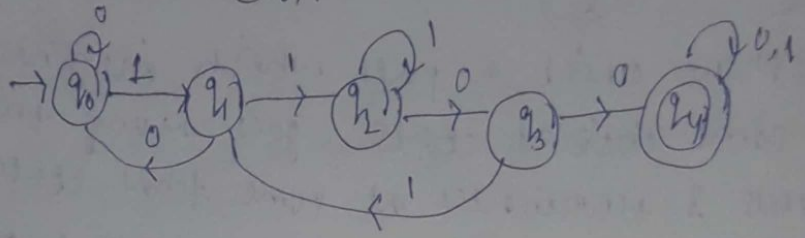
w = 011

3)



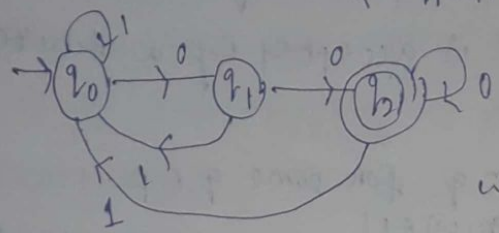
w = 001

4)



w = 1100 & 11011

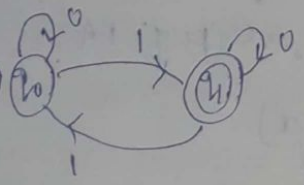
5)



w = 10100

Any:

1)



w = 01011 & 101

M = Q, Σ, δ, q0, F

Q = {q0, q1}

Σ = {0, 1}

Transition Table:-

State \ i/p	0	1
→q0	q0	q1
(q1)	q1	q0

Transition function:-

$\delta(q_0, 0) = q_0$

$\delta(q_0, 1) = q_1$

$\delta(q_1, 0) = q_1$

$\delta(q_1, 1) = q_0$

Initial state → q_0 : q_0

F : q_1

Acceptability of 01011 & 101:

$\delta(q_0, 01011) \vdash \delta(q_0, 1011)$ [$\because \delta(q_0, 0) = q_0$]

$\vdash \delta(q_1, 011)$ [$\because \delta(q_0, 1) = q_1$]

$\vdash \delta(q_1, 11)$ [$\because \delta(q_1, 0) = q_1$]

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

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

↳ final state

As transition ends with final state, so the given string is accepted by given finite Automaton.

$\delta(q_0, 101) \vdash \delta(q_1, 01)$ [$\because \delta(q_0, 1) = q_1$]

$\vdash \delta(q_1, 1)$ [$\because \delta(q_1, 0) = q_1$]

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

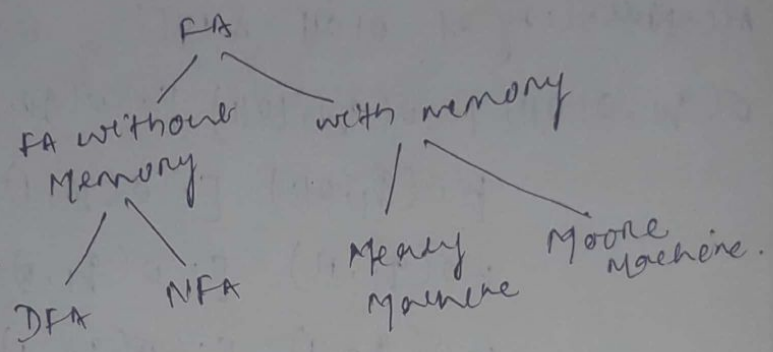
So the given string is not accepted by given FA as transition ends with non-final state.

Types of FA :-

NOTE :-

→ An automaton in which the output depends upon only on the input is called as automaton without memory.

→ An automaton in which the output depends on the states as well as input is called as automaton with a memory.



FA Without Memory :-

Here FA is devided into 2 types.

(i) Deterministic FA (DFA)

(ii) Non-deterministic FA (NDFA / NFA)

DFA :-

The term "deterministic" refers to the fact that on each input symbol, there is one & only one state to which the automaton can transit from its current state.

1. A state doesn't contain more than one transition for same input symbol.
2. Each and every state has to consume all the input symbols present in Σ.

i.e

From any state for every i/p symbol, there must be exactly one transition.

Formal Definition of DFA :-

A DFA 'M' is formally defined by a 5-tuple notation as $M = (Q, \Sigma, \delta, q_0, F)$

where

Q : Finite set of states

Σ : finite set of i/p symbols.

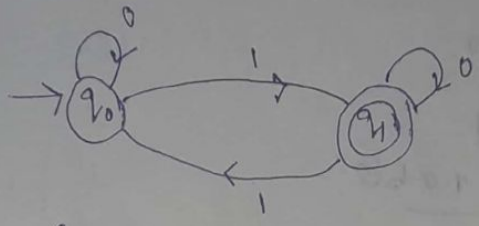
δ : Transition function, $Q \times \Sigma \to Q$

i.e it takes a state & an i/p symbol as argument & returns a state.

q_0 : initial state

F : set of final state. ($F \subseteq Q$)

eg :-



Here $Q : \{q_0, q_1\}$, $\Sigma = \{0, 1\}$

$q_0 : q_0$, $F : q_1$

It is a DFA as both states q_0 & q_1 consume all input symbols in Σ.

Design Procedure of Finite Automata (DFA) :-

1) Understand the language properties for which DFA has to be designed.
2) Determine the set of states required.
3) Identify the initial, final & dead state of DFA.
4) For each state decide the transition to be made for each character of the i/p string.
5) Obtain the transition table & diagram for DFA.
6) Test the DFA obtained by taking short string.

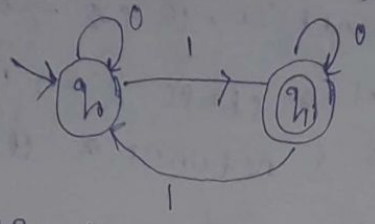
Q
1. Design a FA (DFA) that accepts odd no. of 1's over $\Sigma = \{0, 1\}$

Ans:-

$L(M) = \{1, 01, 001, 1010, 010 \ldots\}$

where M is a DFA of

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



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

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

$q_0 : q_0$

$F : q_1$

Transition Table:-

state \ i/p	0	1
$\rightarrow q_0$	q_0	q_1
(q_1)	q_1	q_0

Transition Function:-

$\delta(q_0, 0) = q_0$

$\delta(q_0, 1) = q_1$

$\delta(q_1, 0) = q_1$

$\delta(q_1, 1) = q_0$

Test for 10101

$\delta(q_0, 10101) \vdash \delta(q_1, 0101)$

$\vdash \delta(q_1, 101)$

$\vdash \delta(q_0, 01)$

$\vdash \delta(q_0, 1)$

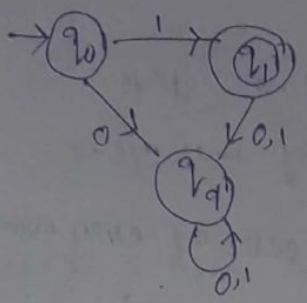
$\vdash q_1$ which is a final state.

so accepted.

2. Design a DFA that accepts odd no. of 1 only, $\Sigma = \{0, 1\}$

Ans: $L(M) = \{1\}$

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



where Q: $\{q_0, q_1, q_d\}$ q_d: dead state

Σ: $\{0, 1\}$

q_0: q_0

F: q_1

Transition Table:-

State / I/P	0	1
$\rightarrow q_0$	q_d	q_1
(q_1)	q_d	q_d
q_d	q_d	q_d

Transition Functions:-

$\delta(q_0, 0) = q_d$ $\quad$ $\delta(q_1, 1) = q_d$

$\delta(q_0, 1) = q_1$ $\quad$ $\delta(q_d, 0) = q_d$

$\delta(q_1, 0) = q_d$ $\quad$ $\delta(q_d, 1) = q_d$

Test for 10:-

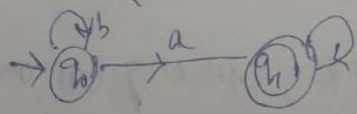
$\delta(q_0, 10) \vdash \delta(q_1, 0)$

$\vdash q_d$ which is not final state i.e. dead state

so it's not accepted.

3. Design a DFA which accepts a string containing atleast one 'a' $\Sigma = \{a, b\}$

$L(M) = \{a, ba, bab, ababa \ldots\}$



Test for babb:

$\delta(q_0, babb) \vdash \delta(q_0, abb)$

$\vdash \delta(q_1, bb)$

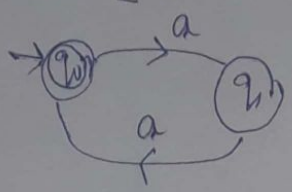
$\vdash \delta(q_1, b)$

$\vdash q_1$ which is a final state.

So this string is accepted by the m/c.

4. Design a DFA that accepts set of even numbers of a's, $\Sigma = \{a\}$

Ans: $L(M) = \{aa, aaaa, aaaaaa \ldots\}$



Test for aaaa:

$\delta(q_0, aaaa) \vdash \delta(q_1, aaa)$

$\vdash \delta(q_0, aa)$

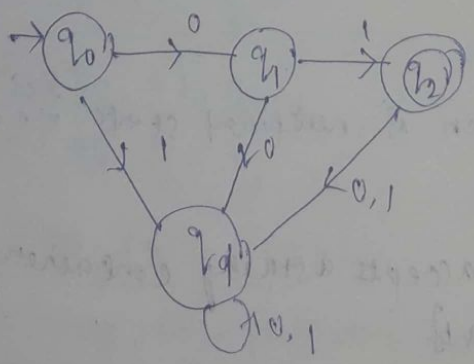
$\vdash \delta(q_1, a)$

$\vdash q_0$ which is a final state.

So this string is accepted.

5. Design a DFA which accepts string 01 only.
$\Sigma = \{0, 1\}$

Ans: $L(M) = \{01\}$

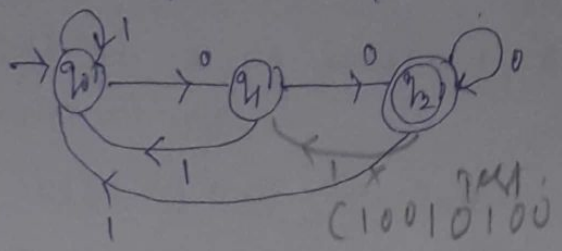


Test for 01 & 101:

6. Design a DFA which accepts string 011 only.
$\Sigma = \{0, 1\}$

7. Design a DFA that accepts string ending with 00 only, $\Sigma = \{0, 1\}$

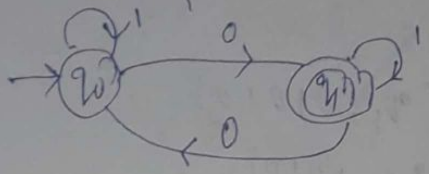
$L(M) = \{00, 000, 100, 0100, 110100 \ldots\}$



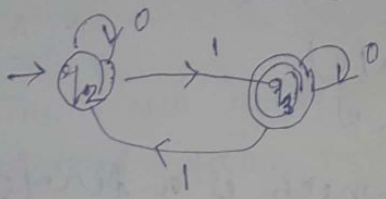
8. Design a DFA which accepts string containing odd no. of 0's & odd no. of 1's, $\Sigma = \{0, 1\}$

Ans: odd no. of 0's

$L(M) = \{0, 000, 10, 1000, 0110 \ldots\}$.



odd no. of 1's



Now we can merge / concatenate above 2 DFA to get one final DFA that accepts strings of odd no. of 0's & odd no. of 1's.

Merger Table:

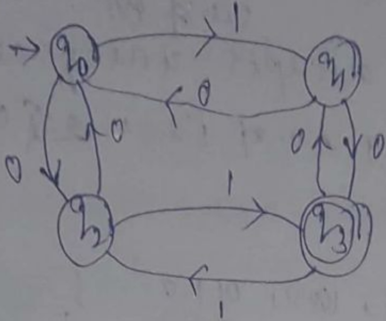
State \ i/p	0	1
→ q_0q_2	q_1q_2	q_0q_3
q_0q_3	q_1q_3	q_0q_2
q_1q_2	q_0q_2	q_1q_3
q_1q_3	q_0q_3	q_1q_2

Let

$q_0q_2 : q_0$ $\quad q_1q_2 : q_2$

$q_0q_3 : q_1$ $\quad q_1q_3 : q_3$

state \ i/p	0	1
$\rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
(q_3)	q_1	q_2



Test for 101010

$\delta(q_0, 101010) \vdash \delta(q_1, 01010)$

$\vdash \delta(q_3, 1010)$

$\vdash \delta(q_2, 010)$

$\vdash \delta(q_0, 10)$

$\vdash \delta(q_1, 0)$

$\vdash q_3$ which is a final state

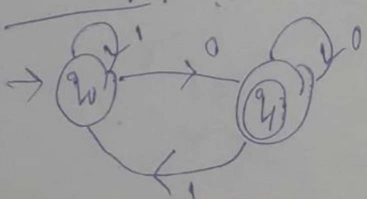
so accepted.

9. Construct FA which accepts all the binary nos divisible by 2.

Ans: $L(M) = \{000, 010, 100, 110, 1110 \ldots\}$

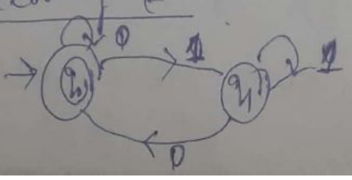
Here $\Sigma = \{0, 1\}$ as binary nos.

Excluding ε :



Mark all string ending with 0

Including ε



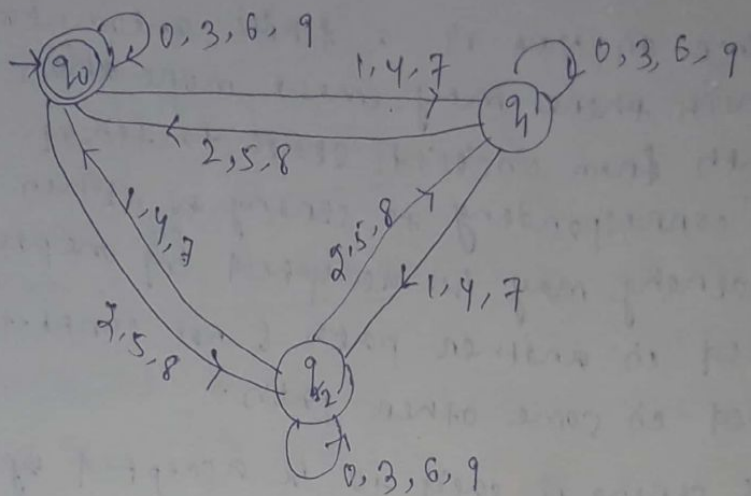
q_0, initial state = final state

NOTE:-
To accept all the strings / binary nos which are divisible by n convert n states.

10. Design a FA which is divisible by 3 on decimal no.

Ans:- L(M): {0, 3, 6, 9, ...}

Σ: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}



NFA:-
The term 'nondeterministic' refers to the fact that from any state for every i/p symbol, there may or may not be a transition.
→ NFA can have zero, one or more transitions from a state on the same input symbol.

NOTE:-
It's not compulsory that all the states have to consume all input symbol in Σ for transition.

Formal Definition of NFA:
NFA can be formally defined by 5-tuple notation. M = (Q, Σ, δ, q0, F)
where Q: finite set of states
Σ: finite set of i/p alphabets
δ: Transition function which maps
$Q \times \Sigma \rightarrow 2^Q$ (power set)

o/p set of states may be empty, may contain exactly one state, or may contain more than one state. so δ maps $Q \times \Sigma \to 2^Q$ (power set of Q)

q_0 : initial state ($q_0 \in Q$)

F : set of final states ($F \subseteq Q$)

Acceptance of String by NFA:

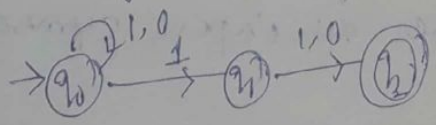
Since an NFA is a finite automaton in which there may exist more than one path from initial state to final state corresponding to string w, if there the string may be accepted by one path, rejected in another path & not completely consumed in some other path.

The string is said to be accepted by NFA if among the various alternative paths, there is atleast one path that leads to a final state.

or

Acceptability of a string is defined as if any one complete path ends with final state, then it will be accepted otherwise rejected.

eg:-



string : 0101

$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0$

$\quad\quad 0 \to q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2$ → final state

so string 0101 is accepted by

$q_0 \to q_0 \to q_0 \to q_0 \to q_1 \to q_2$

from above dig.

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

Σ: $\{0, 1\}$

q_0: q_0

F : q_2

Transition Table :-

state \ i/p	0	1
$\rightarrow q_0$	q_0	$\{q_0, q_1\}$
q_1	q_2	q_2
q_2	-	-

Transition Function :-

$\delta(q_0, 0) = q_0$

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

$\delta(q_1, 0) = q_2$

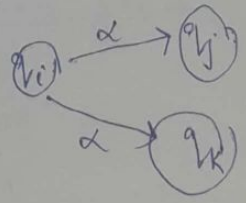
$\delta(q_1, 1) = q_2$

$\delta(q_2, 0) = -$

$\delta(q_2, 1) = -$

NOTE :-

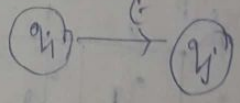
NFA must contain a non deterministic transition i.e



$[q_j \neq q_k]$

so 1) Ambiguity problem

2) Empty Transition



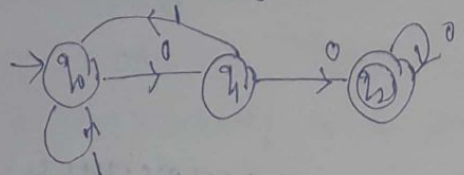
Design of NFA :-

1) Understand the language properties for which NFA has to be designed.
2) Determine set of states & symbols required
3) Identify the initial & final state of NFA (no dead state as state may not have any transition also)
4) Obtain the transition to be made for each state on each character of the input string.
5) Draw the transition table & diagram for NFA.

Q. 1: Design an NFA that accepts a set of all strings ending in 00 over $\{0,1\}$

Ans: $L(M) = \{00, 100, 0100, 000, 1000 \ldots\}$

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



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

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

$q_0: q_0$

$F: q_2$

Transition Table :-

State \ i/p	0	1
→ q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	$- \{\phi\}$

Transition Functions:

$\delta(q_0, 0) = q_1$

$\delta(q_0, 1) = q_0$

$\delta(q_1, 0) = q_2$

$\delta(q_1, 1) = q_0$

$\delta(q_2, 0) = q_2$

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

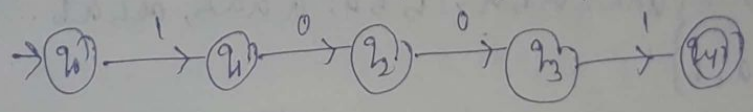
Test for 0 1 0 1 0 0

$q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{0} q_2$

where q_2 is a final state. so this strong is accepted.

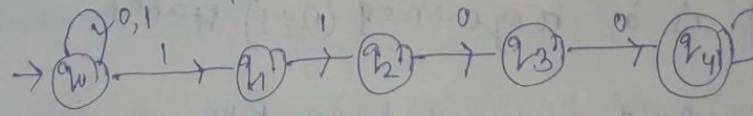
2 Design an NFA that accepts set of all strings containing 1001 only, $\Sigma = \{0, 1\}$

Ans: $L(M) = \{1001\}$, $\Sigma = \{0, 1\}$



3. Design an NFA to accept all strong containing 1100 as a substring over $\Sigma = \{0, 1\}$.

Ans: $L(M) = \{1100, 01100, 011001 \ldots\}$



Test for:

0 1 1 0 0 1

$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0$

$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1$

$\xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \xrightarrow{0} q_3 \xrightarrow{0} q_4 \xrightarrow{1} q_4$

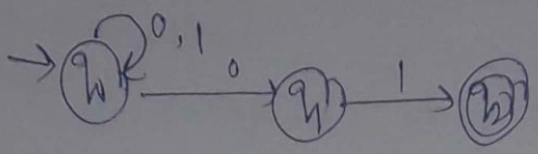
$\xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \xrightarrow{0} q_3 \xrightarrow{0} q_4 \xrightarrow{1} q_4$

So as q_4 is final state, the strong 011001 is accepted by $q_0 \to q_0 \to q_0 \to q_1 \to q_2 \to q_3 \to q_4 \to q_4$

3. Design an NFA to accept set of all strings that end with 01.

Ans:- L(N) = {01, 001, 101, 0001, 1001, ...}

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



Test for 101:-

$q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0$

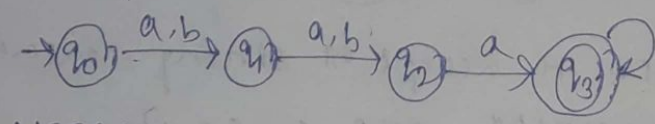
$q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2$ (final state)

So, 101 string is accepted by NFA as q_0-q_0-q_1-q_2 path.

4. construct NFA to accept set of all strings having 3rd symbol from LHS of string is a over a,b

Ans:- L(N) = {aaa, aba, bba, aaab, abab ...}

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

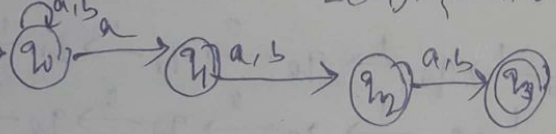


NOTE:-

If it's nth symbol from LHS of the string is a' requires (n+1) states.

Where 3rd symbol from RHS of the string is a:

L(N) = {aaa, aba, abb, aaab ...}

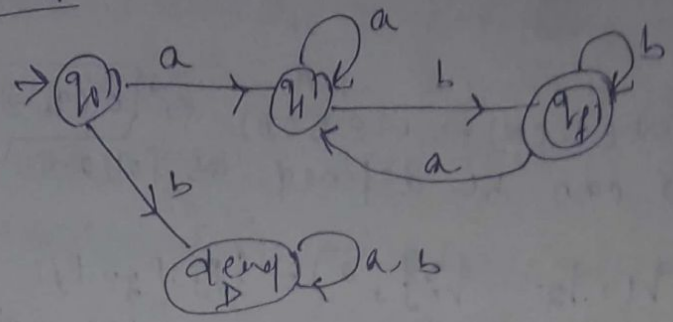


If nth symbol from RHS of the string is a' requires (n+1) states.

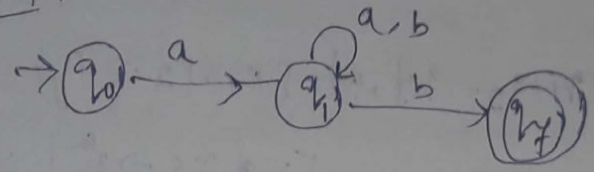
5 Construct FA & NFA that accepts the strings that starts with a & ends with b over a, b.

Ans: L(M) = {ab, aab, abb, abab ...}

DFA:-



NFA:-



EQUIVALENCE OF NFA & DFA:-

FOR all (∀) NFA there exist (∃) a DFA.
i.e if a language 'L' is accepted by NFA, then ∃ a DFA which also accepts 'L'.

Proof:-

Let N = (Q, Σ, δ, q0, F) be a NFA

L(N): Language accepted by NFA.

Here $\delta: Q \times \Sigma \to 2^Q$

& D = (Q', Σ, δ', q0', F') be a DFA

L(D): Language accepted by DFA.

We've to prove/ construct $\delta: Q \times \Sigma \to Q$

L(N) = L(D)

→ Let states of D are: $Q' = 2^Q$ i.e all the states of D are subset of the set of states N.

→ q0' = {q0} i.e initial state of D is initial state of N.

→ The final state of D will be any state of D that contains a final state N. i.e as final state can be more than one & in N we've 2^Q no. of states so we can have more than one final state so the final state of DFA must be including with final state of NFA.

→ Let $w \in L(N) \Rightarrow w \in L(D)$ so $\hat{\delta}'(q_0, w) = F$ s.t. $\hat{\delta}(q_0, w)$

→ Now δ' can be defined as follows.

$$\delta'([q_1, q_2 \cdots q_i], a) = \{p_1, p_2 \cdots p_j\} \quad [\because \alpha' \times \Sigma \to \alpha'] \quad \text{--- (1)}$$

$$\text{iff } \delta(\{q_1, q_2 \cdots q_i\}, a) = \{p_1, p_2 \cdots p_j\} \quad [\because \alpha \times \Sigma \to 2^\alpha]$$

i.e iff we apply δ to each $\{q_1, q_2 \cdots q_i\}$ & taking the union we get some new set of states like $\{p_1, p_2 \cdots p_j\}$, which is for NFA. (for NFA as $\delta(q_0, w, a)$)

If we combine the set of states $\{q_1, q_2 \cdots q_i\}$ as a single state & imposing 'a' we'll move to another single state as $\{p_1, p_2 \cdots p_j\}$ which is of DFA. as here $\delta(q_0, a) = q_j$ so will be for DFA.

From the above definition

$\delta'([q_0], w) = \{p_1, p_2 \cdots p_j\}$ iff $\delta(q_0, w) = \{p_1, p_2 \cdots p_j\}$

Provided $\{p_1, p_2 \cdots p_j\} \Rightarrow$ Final state

Now we prove this by help of induction above statement

for some input string 'w',

i.e $\delta'(q_0, w) = \{p_1, p_2 \cdots p_j\}$ iff $\delta(q_0, w) = \{p_1, p_2 \cdots p_j\}$

Induction method:

Basis:

The result is true for $|w| = 0$, if $w = \epsilon$ because $\hat{\delta}'(q_0, \epsilon) = q_0$ iff $\hat{\delta}(q_0, \epsilon) = q_0$

[∵ Property 1 of extended transition function]

Hypothesis:

Let us assume that the result is true for $|w| = n$ length of string = n

Hence $\hat{\delta}'(q_0, w) = \{p_1, p_2 \cdots p_j\}$ iff $\hat{\delta}(q_0, w) = \{p_1, p_2 \cdots p_j\}$.

[i.e on imposing a string to state in NFA let's go to some states so when that string could be computed in states of DFA then its all states must be subset of states of NFA state]

Induction:

Now let us assume that this result is true for any string of length $n+1$ i.e $|w| > n$

Let $w = xa$, $x \in \Sigma^*$ so $|x| = n$

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

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

(from hypothesis as $q_0 x = p_1 p_2$ ... where $|x| = n$)

iff $\delta(\{p_1, p_2 \cdots p_j\}, a) = \{q_1, q_2 \cdots q_i\}$ by using ①

Hence the result is true for $|w| = n+1$, when length of string is n.

So if anyone of the set $\{q_1, q_2 \cdots q_i\}$ contains a final state F, then it becomes a final states of F'. i.e $\delta'(q_0', w) \in F'$ iff $\delta(q_0, w) \in F$

So $\boxed{L(N) = L(D)}$ (Proved)

Conversion from NFA to DFA :-

Procedure :-

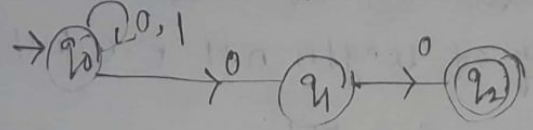
→ First make the starting state of NFA as starting state of DFA. Apply symbols of Σ as input to q_0 & keep the o/ps in second & third column soon.

→ Next take the new generated states which are placed in second & third column, keep them in first column. Again apply input symbols. Then on basis of transition keep the new o/ps in 4th & 5th column.

→ Repeat the process until no new state is left.

Q. Convert following NFA to DFA

1.



Ans: Step 1 :

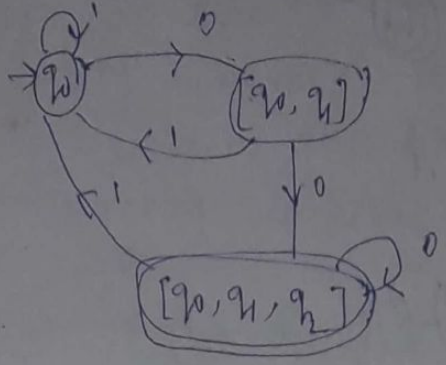
Transition Table (for above NFA)

State \ i/p	0	1
→ q_0	$\{q_0, q_1\}$	q_0
q_1	q_2	-
(q_2)	-	-

Step 2 :

i/p	0	1
→ q_0	$\{q_0, q_1\}$	q_0
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	q_0
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	q_0
~~$\{q_0, q_1\}$~~		

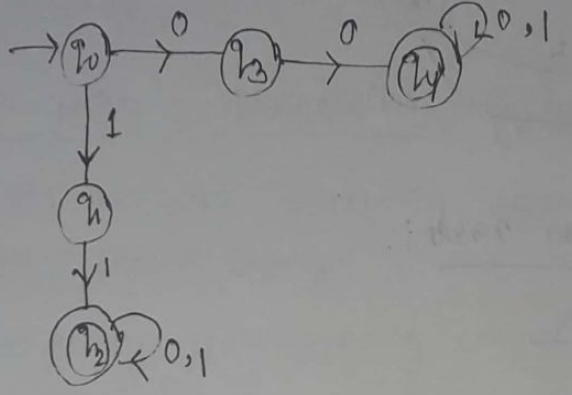
Step 3: Draw DFA:



NOTE:

As q_2 is final state in NFA, so the set of states where q_2 is present will be the final state of DFA.

2.



Ans:

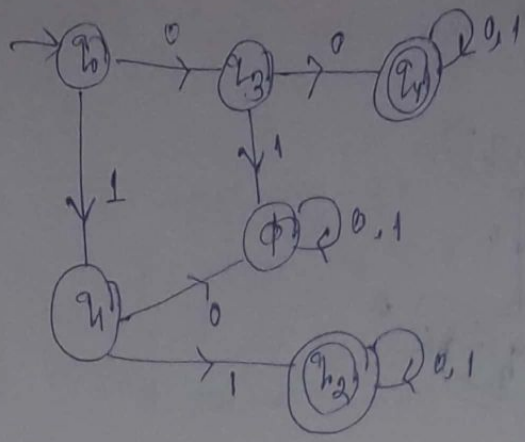
Step 1:

Transition Table

state \ i/p	0	1
→ q_0	q_3	q_1
q_1	ϕ	q_2
(q_2)	q_2	q_2
q_3	q_4	ϕ
(q_4)	q_4	q_4

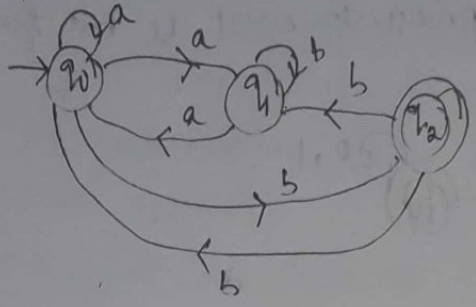
Step 2: Conversion Table:

state \ i/p	0	1
→ q_0	q_3	q_1
q_3	q_4	ϕ
q_1	ϕ	q_2
(q_4)	q_4	q_4
(q_2)	q_2	q_2



As ϕ is a dead state, it must consume all input symbols of Σ, since it's DFA.

3.



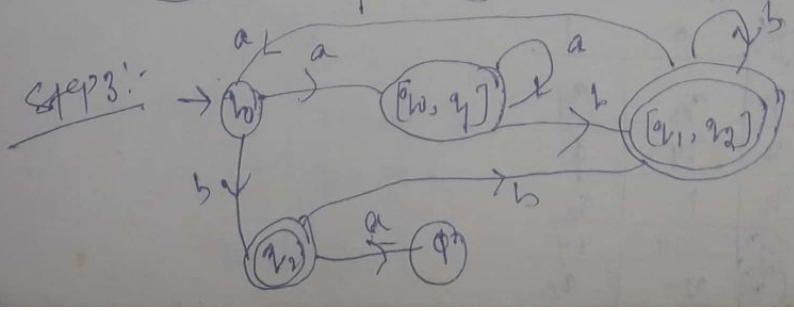
Step1:- Transition table:-

i/p / State	a	b
→ q_0	$\{q_0, q_1\}$	q_2
q_1	q_0	q_1
(q_2)	ϕ	$\{q_1, q_2\}$

Step2:- conversion table:-

i/p / State	a	b
→ q_0	$[q_0, q_1]$	q_2
$[q_0, q_1]$	$[q_0, q_1]$	$[q_1, q_2]$
(q_2)	ϕ	$[q_1, q_2]$
($[q_1, q_2]$)	q_0	$[q_1, q_2]$

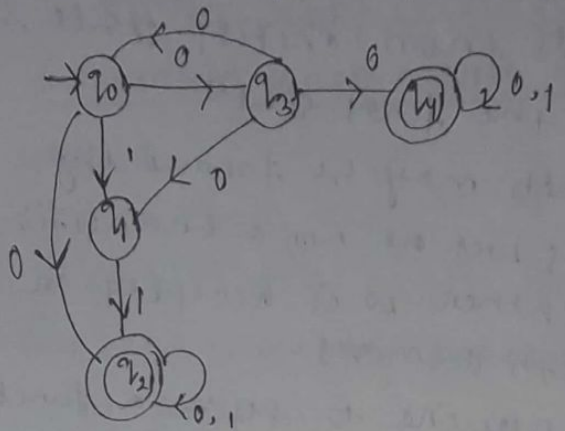
Step3:-



NOTE:-

A DFA can simulate the behaviour of NFA by increasing number of states & the state which contains the final state of NFA will be considered as final state of DFA.

4.

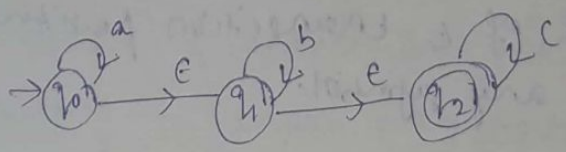


NFA with ∈-Transition:- (∈-NFA)

It's an NFA including transitions on the empty input symbol i.e ∈ (epsilon).

∈ → empty i/p / string with 0 length.

eg:-



Formal Definition:-

∈-NFA is defined as 5-tuple m/c.

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

Q: Finite set of states

Σ: finite set of i/p alphabets

δ: Transition function which maps

$Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$ i.e $Q \times \Sigma^* \to 2^Q$

q_0: initial state ($q_0 \in Q$)

F: set of final state.

Acceptance of a string by ε-NFA:

A string w in Σ^* will be accepted by ε-NFA (NFA with ε-transition), if there exist atleast one path corresponding to w, which starts from initial state & ends on one of the final states.

Since this path may be formed by ε-transition as well as non ε-transition to find out whether w is accepted or not by NFA with ε-moves.

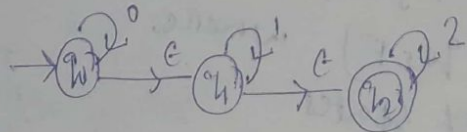
So, here we require to define a function ε-closure(q_i) where q_i is a state of the automata.

ε-closure(q_0) is pronounced as epsilon closure q_0 is defined as the set of all possible states which can be reachable from the state q_0 using a sequence of ε-transition / without reading any symbol.

or

set of all possible reachable states using a sequence of ε-transition from q_0.

eg:-

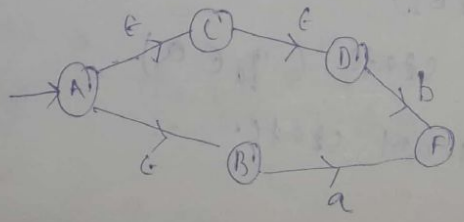


ε-closure(q_0) = {q_0, q_1, q_2}

ε-closure(q_1) = {q_1, q_2}

ε-closure(q_2) = {q_2}

eg:



ϵ-closure(A) : {A, B, C, D}

ϵ(B) : {B, C}

ϵ(C) : {C, D}

ϵ(D) : {D}

ϵ(F) : {F}

Conversion from ϵ-NFA to NFA :-

Steps :

1. Find the ϵ-closure of each state in NFA.
2. Remove the empty Transition.
3. Every non-empty transition must be modified as follows

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

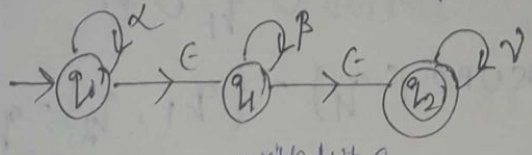
After converting ϵ-NFA to NFA

⇓

- Number of states are same
- Starting state is same.
- final states & transitions are different.

By reading ϵ from any state reachable to final state, those states are final.

Q. Convert the following ϵ-NFA from to NFA.



without ε

Ans: Let ϵ-NFA is a 5 tuple M(

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

Q' : {ϵ-closure(q_0), ϵ-closure(q_1), ϵ-closure(q_2)}

Σ : {α, β, γ}

q_0' : q_0

Step 1:

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

ϵ-closure(q_1) = $\{q_1, q_2\}$ = q_y

ϵ-closure(q_2) = $\{q_2\}$ = q_z

Step 2:

Modifying each nonempty transition.

$\delta'(q_x, \alpha)$ = ϵ-closure($\delta(q_x, \alpha)$)

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

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

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

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

$\delta'(q_x, \beta)$ = ϵ-closure($\delta(q_x, \beta)$)

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

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

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

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

$\delta'(q_x, \nu)$ = ϵ-closure($\delta(q_x, \nu)$)

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

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

= ϵ-closure($\phi \cup \phi \cup q_2$)

= ϵ-closure(q_2) = $\{q_2\}$ = q_z

$\delta'(q_y, \alpha)$: ϵ-closure $(\delta(q_y, \alpha))$

: ϵ-closure $(\delta(q_1, q_2), \alpha)$

: ϵ-closure $(\delta(q_1, \alpha) \cup \delta(q_2, \alpha))$

: ϵ-closure $(\phi \cup \phi)$: $\boxed{\phi}$

$\delta'(q_y, \beta)$: ϵ-closure $(\delta(q_y, \beta))$

: ϵ-closure $(\delta(q_1, q_2), \beta)$

: ϵ-closure $(\delta(q_1, \beta) \cup \delta(q_2, \beta))$

: ϵ-closure $(q_1 \cup \phi)$

: ϵ-closure (q_1) : $\{q_1, q_2\}$ = $\boxed{q_y}$

$\delta'(q_y, \gamma)$: ϵ-closure $(\delta(q_y, \gamma))$

: ϵ-closure $(\delta(q_1, q_2), \gamma))$

: ϵ-closure $(\delta(q_1, \gamma) \cup \delta(q_2, \gamma))$

: ϵ-closure $(\phi \cup \phi)$: q_2 : $\boxed{q_y}$

$\delta'(q_z, \alpha)$: ϵ-closure $(\delta(q_z, \alpha))$

: ϵ-closure $(\delta(q_2, \phi))$

: ϵ-closure (ϕ) : $\boxed{\phi}$

$\delta'(q_z, \beta)$: ϵ-closure $(\delta(q_z, \beta))$

: ϵ-closure $(\delta(q_2, \phi))$

: ϵ-closure (ϕ) : $\boxed{\phi}$

$\delta'(q_z, \gamma)$: ϵ-closure $(\delta(q_z, \gamma))$

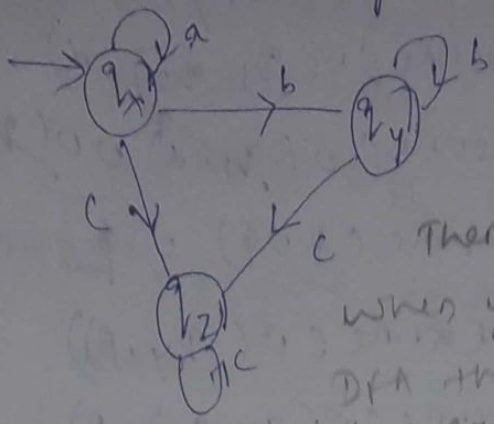
: ϵ-closure $(\delta(q_2, \gamma))$

: ϵ-closure (q_2) : q_2 : $\boxed{q_z}$

So transition table for DFA without ϵ will be

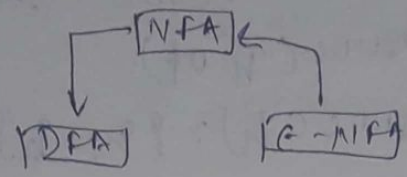
δ	α	β	γ
$\rightarrow^* q_x$	q_x	q_y	q_z
q_y	ϕ	q_y	ϕ
q_z	ϕ	ϕ	q_z

So transition diagram is as follows



There's no ϕ as state when you convert it into DFA there'll be state of ϕ b'cz in NFA you may have transition or may not also.

NOTE!:



MINIMIZATION OF DFA:-

For all DFA, it has a unique language but the reverse is not true.

→ for a given language there may be more than 1 DFA, that accepts it.

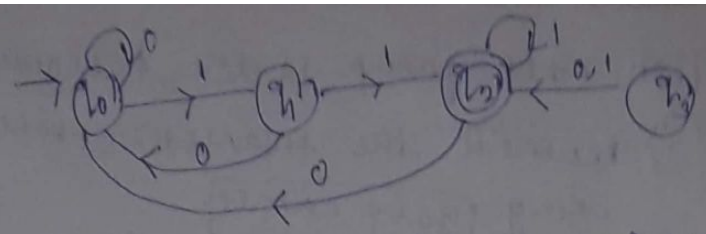
To understand minimization of DFA we've to clearly understand of

(i) dead state
(ii) inaccessible state
(iii) indistinguishable state
(iv) distinguishable state

Inaccessible State:-

All those states which can never be reached from initial state are called inaccessible state. or

There's no in-degree but goes/reaches to any one of the state which is accessible.



q_3: Unreachable / Inaccessible state.

Indistinguishable State:-

Two states q & r of DFA are called indistinguishable if

$\hat{\delta}(q, w) \in F \Rightarrow \hat{\delta}(r, w) \in F$

$\hat{\delta}(q, w) \notin F \Rightarrow \hat{\delta}(r, w) \notin F$

$\forall w \in \Sigma^*$

Distinguishable states:-

Two states q_1 & q_2 of DFA are called distinguishable if

$\hat{\delta}(q_1, w) \in F$ & $\hat{\delta}(q_2, w) \notin F \; \forall w \in \Sigma^*$

or vice versa.

Steps of Minimization of states of DFA:-

We can merge some of states to form a composite state. So that no. of steps or transitions get reduced.

Step 1: Remove inaccessible or unreachable states from the given DFA.

2: Draw the transition table for rest states.

3: Divide the rows of transition table in two states.

(i) set of final states

(ii) set of nonfinal states.

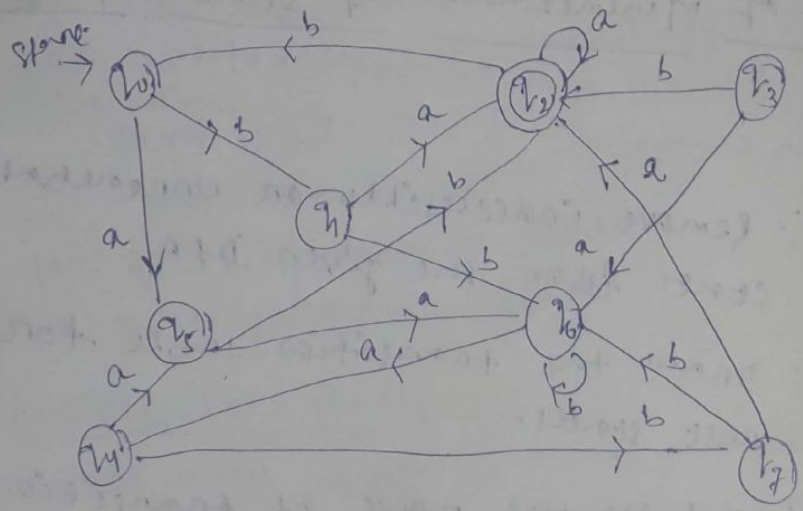
4:

4: For each group do the following:
i) Rewrite the transition table using group names.
ii) Find identical rows.
iii) For identical rows, create a group of states & the other states will form another group.

5: Repeat step 4 until no more division is possible.
[If no. of steps is prime no., it can be decided]

6: Using the composible states redraw the transition table to find the min^m states of DFA.

Q1. Minimise the DFA given below.



Step 1:-
Remove the unreachable state from the given DFA. Here q_3 is unreachable state. So remove it.

Step 2:- Draw the transition table for reff states after removing q_3 (unreachable state)

q \ Σ	a	b
→ q_0	q_5	q_1
q_1	q_2	q_6
* q_2	q_2	q_0
~~q_3~~		
q_4	q_5	q_7
q_5	q_6	q_2
q_6	q_4	q_6
q_7	q_2	q_6

Step 3:- Divide the rows of transition table in 2 sets.

Set 1:- set of final states.

Gr II :-

	a	b
q_2	q_2	q_0

on q_2 q_2 q_0

set 2:- Gr I:- set of nonfinal states.

q_0 q_5 q_1

q_1 q_2 q_6

q_4 q_5 q_7

q_5 q_6 q_2

q_6 q_4 q_6

q_7 q_2 q_6

Step 4:-

Step 4:

q_0 q_5 q_1 → row 1

q_1 q_2 q_6 → row 2

q_4 q_5 q_7 → row 3

q_5 q_6 q_2 → row 4

q_6 q_4 q_6 → row 5

q_7 q_2 q_6 → row 6

As row 2 & row 6 are similar as q_1 & q_7 transit to same states on a & b. So skip one of them, let skip row 6. So q_7 is replaced by q_1 in the rest.

Step 5: Write the next table as follows:

q_0 q_5 q_1 → row 1

q_1 q_2 q_6 → row 2

q_4 q_5 q_1 → row 3

q_5 q_6 q_2 → row 4

q_6 q_4 q_6 → row 5

Repeat step 4 on the next table of Set 1 till you are finding similar rows & write next table.

Step 6:

q_0 q_5 q_1

q_1 q_2 q_6

q_4 q_5 q_2

q_6 q_4 q_6

Now there are no two rows same so this transition table is minimised set

Step 7: Do same process for set 2 as

Grp II: q_2 q_2 q_6

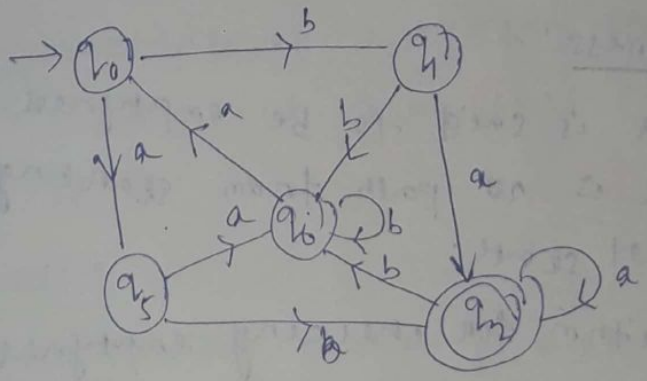
So only single row, already minimised.

Step 8: Now combine grp I & II as

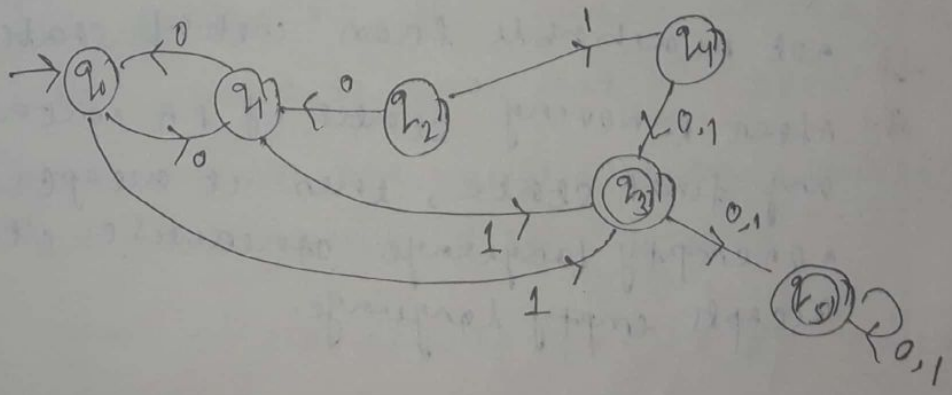
Σ	a	b
$\rightarrow q_0$	q_5	q_1
q_1	q_2	q_6
q_5	q_6	q_2
q_6	q_6	q_6
$*q_2$	q_2	q_6

[Transition table of minimised DFA]

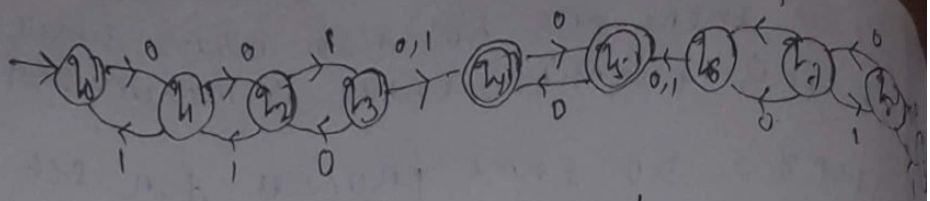
Step 9: Transition dig. of minimised DFA.



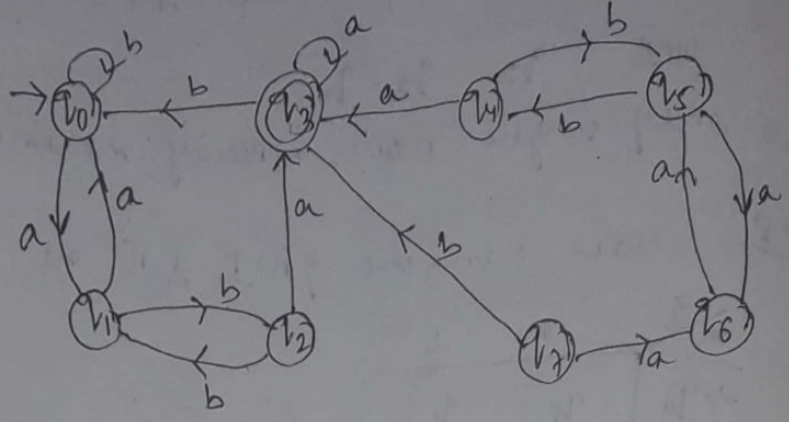
2.



3



4.



DECISION PROPERTIES OF FA :- / Decidability Problems

The decision properties of FA include

(1) Emptyness
(2) Finiteness
(3) Equivalence

Emptyness :-

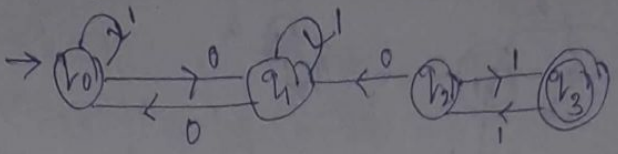
A FA is said to be emptyness if there is no path from starting state to final state.

Algorithm for checking emptyness:

1. Remove all the states which are not reachable from start state.
2. After removing states if FA contains any final state, then it accepts nonempty language otherwise it accepts empty language.

eg: check whether the following FA accepts empty lang. or not.

1.



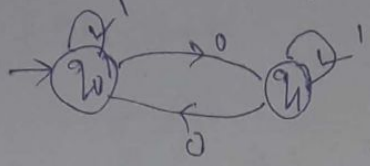
Ans:

Step 1:

Remove unreachable state q_2 & q_3

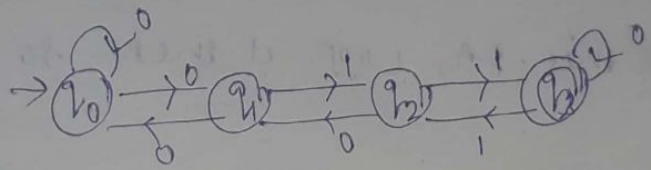
Step 2:

After removing



As it does'nt contain any final state so it accepts empty language.

2.



Ans:

It accepts all nonempty language as we can reach from initial state to final state.

Finiteness:

Algorithm for checking finiteness is:

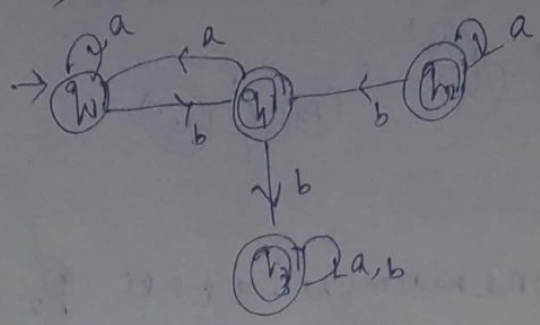
Step 1: Remove all the states which are not reachable from initial state.

2: Remove all the states from which we can't reach final states.

3: Then in remaining FA, if it contains any loops, then the problem is infinite otherwise finite.

check whether following FA accepts finite lang. or not

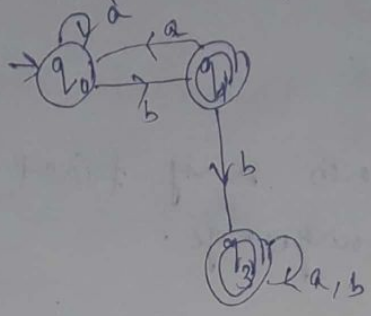
eg:- →



Ans:

Step 1:-

After removing unreachable state



so as in this FA, loop is there so it is infinite.

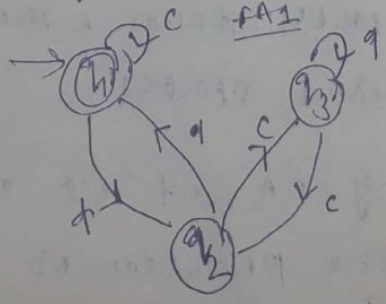
Equivalence:-

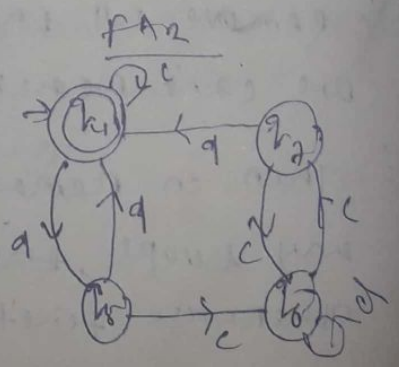
Two FA are equivalent if they accept the same set of strings over Σ.

TWO FAs are not equivalent if there's same string w over Σ accepted by one automata (comes to final state) whereas other automata do not accept w. (Conversely)

eg. check whether the following 2 FA are equal or not.

1.



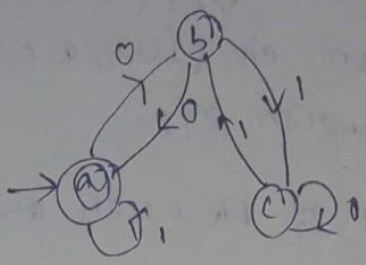


comparision Table:-

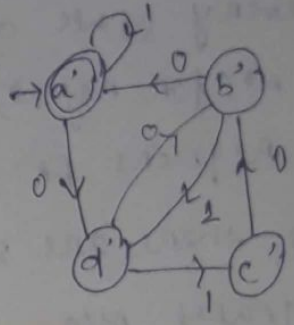
a	Σ c	d
→[q_1, q_4]	(q_1, q_4)	(q_2, q_5)
(q_3, q_5)	(q_3, q_6)	(q_1, q_4)
(q_3, q_6)	(q_2, q_7)	(q_3, q_6)
(q_2, q_7)	q_3, q_6	q_1, q_4

Here both the FA are equal because in every step it'll go to final state or non final state.

2.



FA1



FA2

Ans:

comparision table:

Applying the above procedure

a Σ	0	1
a, a'	(b, d')	(a, a')
(b, d')	(a, b')	(c, c')

(a, b') contains a as final & b' is not final

Not equivalent as

Finite Automata with Output :-

The finite Automata which we considered in the earlier sections have binary outputs i.e they accept the string or reject the string. This acceptability of string was decided on the basis of reachability of the final state from initial state. But if we remove this restriction & we need to specify output at every state other than yes (or accept) or no (or reject), then in such a case we require FA alongwith O/P.

So there are 2 types of FA with output

(i) Mealy M/c
(ii) Moore m/c.

Mealy M/c :-

A mealy m/c is a finite state machine where the O/Ps are determined by the current state & i/p.

Mealy m/c is a 6-tuple m/c.

$M : Q, \Sigma, \Delta, \delta, \lambda, q_0$

where Q : Finite set of states

Σ : Finite set of i/p Alphabets

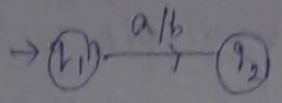
Δ : O/p Alphabet

δ : Transition function $(Q \times \Sigma \to Q)$

λ : output function $(Q \times \Sigma \to \Delta)$

q_0 : initial state $(q_0 \subseteq Q)$

Transition Dig:-

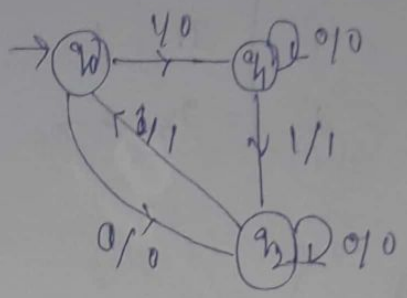


a: input
b: output
q_1 & q_2 : state

Transition Table:-

present state	next state a: 0		next state a: 1	
	state	o/p	state	o/p

Eg:-



Transition table:-

present state	next state i/p = 0		i/p = 1	
	state	o/p	state	o/p
→ q_0	q_2	0	q_1	0
q_1	q_1	0	q_2	1
q_2	q_2	0	q_0	1

For i/p string 1001 o/p will be 0001.

As 1 0 0 1

→ q_0 q_1 q_1 q_1 q_2 - since it is i/p 1

(0) (0) (0) (1)

* i/p = 0 1 0 1 0

q_0 q_2 q_0 q_2 q_0 q_2

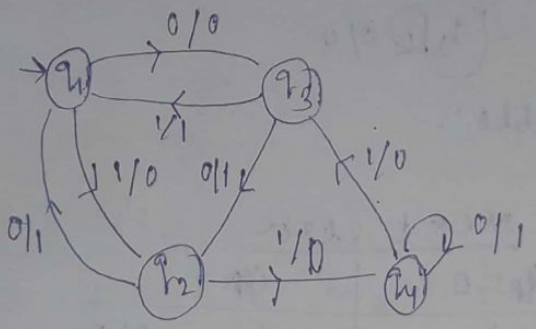
(0) (1) (0) (1) (0) → o/p.

NOTE:-

If i/p is of length n, then o/p sequence is also of length n.

eg2:-

present state	Next state			
	i/p = 0		i/p = 1	
	state	o/p	state	o/p
→ q_1	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0



i/p:- 0 0 1 1

q_1 q_3 q_2 q_4 q_3

(0) (1) (0) (0)

Moore Machine:-

A moore m/c is a finite state machine where o/ps are determined by its current state.

Moore machine is a 6-tuple machine.

$M = Q, \Sigma, \Delta, \delta, \lambda, q_0$

where Q: finite set of states

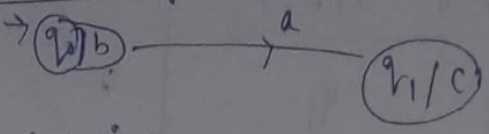
Σ: " " " i/p alphabets

Δ: " " " o/p "

δ: Transition function $(Q \times \Sigma \to Q)$

λ: O/P function ($Q \to \Delta$). Also known as machine function.

Transition diagram:-



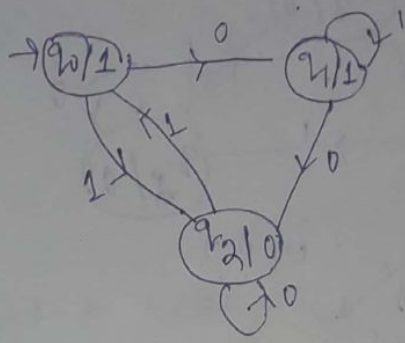
a : c/p

b & c : o/p q_0 & q_1 : states.

Transition Table:-

present state	Next state c/p = 0	Next state c/p = 1	o/p

eg:-



Transition Table:-

Present state	Next state c/p = 0	Next state c/p = 1	o/p
→ q_0	q_1	q_2	1
q_1	q_2	q_1	1
q_2	q_2	q_0	0

for the string 0110, o/p will be 1110

c/p :- 0 1 1 0

q_0 q_1 q_1 q_1 q_2

(1) (1) (1) (1) (0)

1 1 1

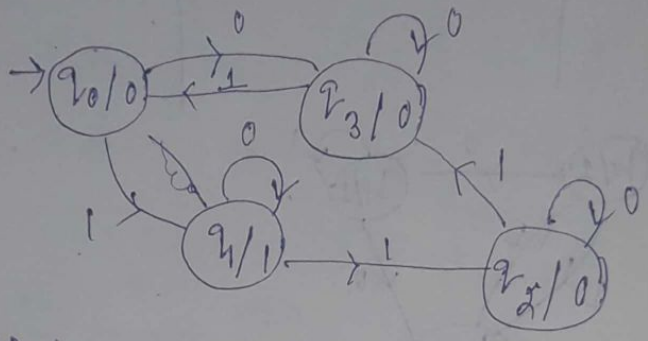
q_0 q_2 q_0 q_2

(1) (0) (1) 0

NOTE:- If length of c/p string is n, o/p string has length n+1.

eg 2:-

Present state	Next state i/p:0	Next state i/p:1	o/p
→ q_0	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0



i/p :- 0 1 1 1 o/p :

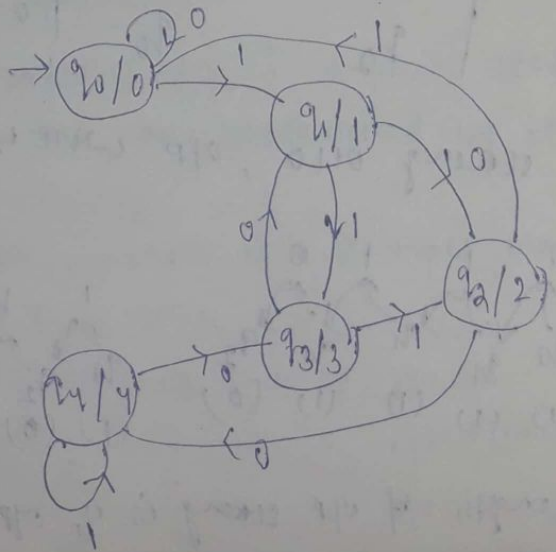
o/p : $q_0 \xrightarrow{0} q_3 \to q_0 \to q_1 \to q_2$

(0) (0) 0 (1) (0)

o/p :- 0 0 0 1 0

Q Construct a Moore m/c for a set of binary nos divisible by 5 over $\Sigma = \{0,1\}$, $\Delta = \{0,1,2,3,4\}$

Ans :-



Present state	Next state i/p:0	Next state i/p:1	o/p
→ q_0	q_0	q_1	0
q_1	q_2	q_3	1
q_2	q_4	q_0	2
q_3	q_1	q_2	3
q_4	q_3	q_4	4

i/p: 0 1 0 1 => 5

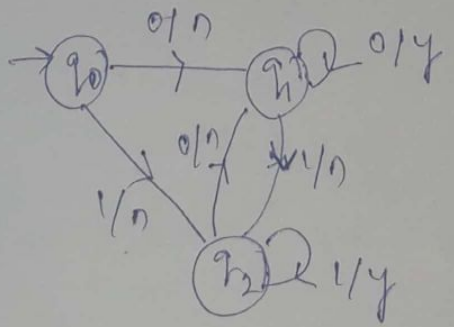
$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{1} q_0$

(0) (0) (1) (2) (0)

└ final 0/1 as remainder : 0 when decoded as 5.

Q. Construct a Mealy m/c to accept the string that ends with 00 or 11.

Ans:



Present state	i/p:0 state	i/p:0 o/p	i/p:1 state	i/p:1 o/p
→ q_0	q_1	y	q_2	y
q_1	q_1	y	q_2	y
q_2	q_1	y	q_2	y

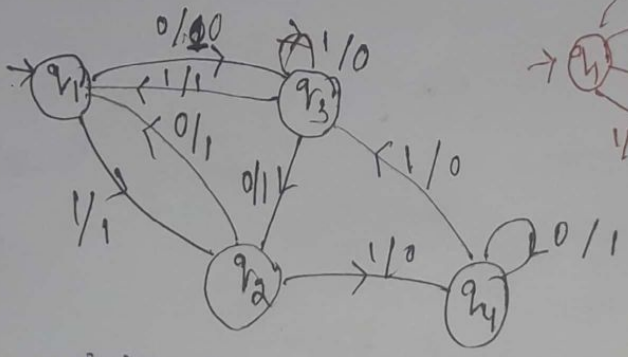
Equivalence of Moore & Mealy machine:-

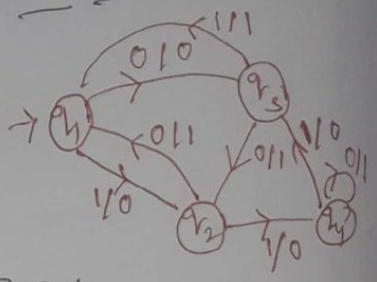
The meaning of equivalence is two machines accept the same lang.

But here mealy m/c & Moore m/c are not equivalent because the length itself is different in both the machines i.e length of moore m/c is one longer than mealy m/c for the given input.

So we can't convert equivalent m/c but we can convert from mealy m/c to moore m/c or viceversa.

Conversion of Mealy M/C to Moore M/C:-





Transition table:-

Present state	Next state: input = 0		input = 1	
	state	o/p	state	o/p
→ q_1	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

step 1

NOTE:-

- Here we've to split the state q_i into several diff. states.

The no. of such state being equal to the number of diff. o/ps associated with q_i.

$q_1 \to 1$

$q_3 \to 0$

$q_2 \to 0 \to q_{20}$

$\to 1 \to q_{21}$

$q_4 \to 0 \to q_{40}$

$\to 1 \to q_{41}$

Step 2:-

Redraw the transition table using splitted states.

present state	Next state i/p = 0	o/p	i/p = 1	o/p
$\to q_1$	q_3	0	q_{20}	0
q_{20}	q_1	1	q_{40}	0
q_{21}	q_1	1	q_{40}	0
q_3	q_{21}	1	q_1	1
q_{40}	q_{41}	1	q_3	0
q_{41}	q_{41}	1	q_3	0

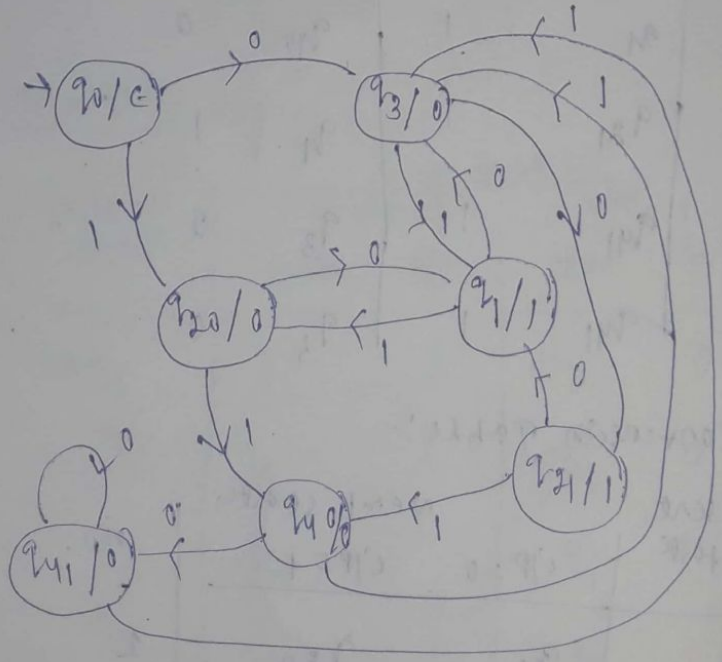
Step 3:- Conversion Table:-

present state	Next state i/p = 0	i/p = 1	o/p
$\to q_1$	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0

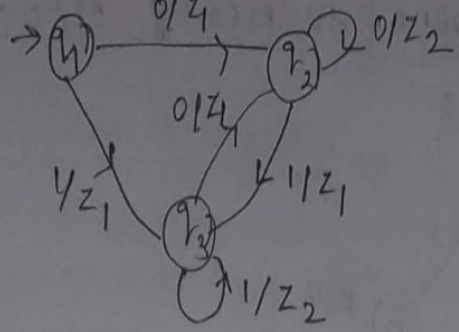
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

Step 4:- Add dummy state in transition table.

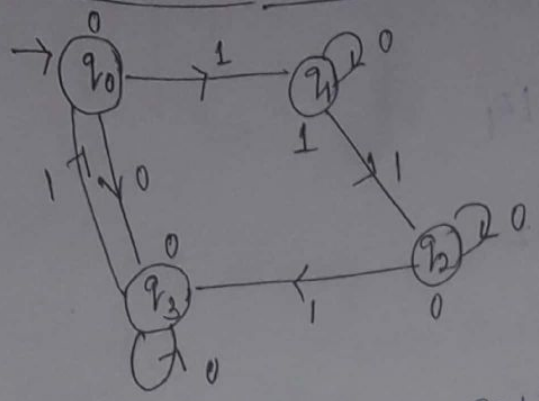
present state	next state i/p=0	next state i/p=1	o/p
→ q_0	q_3	q_{20}	∈
q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	0



Q



Conversion from Moore to Mealy M/C :-



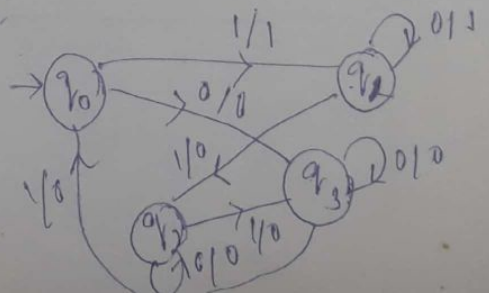
Moore m/c transition table :-

Present state	Next state i/p = 0	Next state i/p = 1	o/p
→ q_0	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

Conversion table :-

Present state	Next state i/p = 0		Next state i/p = 1	
	state	o/p	state	o/p
→ q_0	q_3 (3 o/p = 1)	0	q_2	1
q_1	q_1	1	q_2	0
q_2	q_2	0	q_3	0
q_3	q_3	0	q_0	0

Transition diag :-



REGULAR EXPRESSION

Regular Language :-

It's the set of all possible strings which are accepted by finite Automata is called as Regular language.

or

Language accepted by FA is called as Regular language.

Regular set :-

It's the set of strings taken from a Regular language.

Regular set may be taken as a subset of regular language.

Regular Expression : [RE is a mathematical expression for a given string.]

- Regular Expression is the description of a language in an algebraic fashion/description.
- → or RE are useful for representing certain set of strings in an algebraic fashion.
- → It's a set of symbols used to represent a string from a regular set.

or

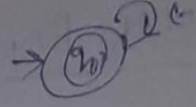
- A RE is a collection of i/p symbol & operation on strings (union, concatenation & closure)

The class of RE over Σ is defined recursively as follows ; / properties.

(1) ϕ is a RE.

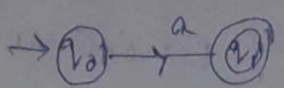
(No string)

(2) CFG

(ii) ϵ is a RE  (ϵ is a FA)

String with 0 length

(iii) If $a \in \Sigma$ then a is also a RE.

 [∵ can be a FA]

(iv) If r_1 & r_2 are both RE, then

a) $r_1 \cdot r_2$ is also a RE

b) $r_1 + r_2$ / $r_1 \cup r_2$ is also a RE

c) r_1^* is also a RE

These 3 properties are we'll prove certain properties of these 3 properties. We already know r.e. as RE from previous chapter.

Operations of RE :-

① Union :-

The union of two languages L_1 & L_2 denoted by $L_1 \cup L_2$ or $L_1 + L_2$ is the set of string that are in either L_1 or L_2 or both.

eg :- $L_1 = \{10, 111\}$ $L_2 = \{\epsilon, 001\}$,

then $L_1 \cup L_2$ / $L_1 + L_2 = \{\epsilon, 001, 10, 111\}$

② Concatenation :-

The concatenation of languages L_1 & L_2 denoted by $L_1 \cdot L_2$ (dot) can be formed by taking any string in L_1 and concatenating it with any string in L_2.

eg :- $L_1 = \{ab, c\}$, $L_2 = \{\epsilon, d, ef\}$

$L_1 \cdot L_2 = \{ab, abd, abef, c, cd, cef\}$

③ Closure :

Kleene closure
positive closure]→ These you already know

eg:- construct / Describe RE for the following langs:

(1) set of strings of any length over $\Sigma = \{0\}$
including $\epsilon \longrightarrow 0^* = \{\epsilon, 0, 00, 000 \ldots\}$
" excluding $\epsilon \longrightarrow 0^+ \{0, 00, 000 \ldots\}$

(2) set of strings of a's & b's, over $\Sigma = \{a, b\}$
of any length including $\epsilon \longrightarrow (a+b)^*$
" excluding $\epsilon \longrightarrow (a+b)^+$

(3) set of strings of a's & b's over $\Sigma = \{a, b\}$
ending with abb $\longrightarrow (a+b)^* abb$

(4) set of strings of a's & b's over $\Sigma = \{a, b\}$
starting with ab $\longrightarrow ab(a+b)^*$

(5) 0 followed by 1 over $\Sigma = \{0, 1\} \longrightarrow 01$

(6) Either 0 or 1 over $\Sigma = \{0, 1\} \longrightarrow 0 + 1$

(7) Set of strings of any no. of 0's followed by any no. of 1's followed by any no. of 2's over $\Sigma = \{0, 1, 2\} \longrightarrow 0^* 1^* 2^*$

(8) set of strings of a's & b's over $\Sigma = \{a, b\}$
of any length having a substring aa
$\longrightarrow (a+b)^* aa (a+b)^*$

(9) Set of strings containing even no. of zeroes
$\longrightarrow (00)^*$

(10) set of string containing odd no of ones
$\longrightarrow (1)^* \cdot 1$

(11) Set of strings containing even no. of a's
followed by odd no. of b's over $\Sigma = \{a, b\}$
$\longrightarrow (aa)^* [(bb)^* \cdot b]$

(12) strings with atleast two consecutive zeros over $\Sigma = \{0,1\}$ ⟶ $(0+1)^* 00 (0+1)^*$

(13) Set of strings with no consecutive a's over $\Sigma = \{a, b\}$ ⟶ $(b+ab)^* (a+\epsilon)$

(14) $L = \{w : |w| \bmod 3 = 0, w \in (a,b)^*\}$ ⟶ $[(a+b)^3]^*$

$|w| \bmod 3 = 0$ means length of string divided by 3 & remainder = 0. so length of w must be 0, 3, 6, 9 ... so there no must be multiple of 3.

Correspondence between RE & Regular set :-

Lang RE	Regular set
(1) any no. of a's a^*	$\{\epsilon, a, aa, aaa \ldots\}$
(2) either a or b $a+b$ / $a \cup b$	$\{a, b\}$
(3) string with a or b followed by c $(a+b)c$	$\{ac, bc\}$
(4) ϵ	$\{\epsilon\}$
(5) ϕ	$\{\}$ empty set
(6) string with either a or b followed by c and then either d or e $(a+b)c(d+e)$	$\{acd, ace, bcd, bce\}$
(7) abc^*de	$\{abde, abcde, abccde \ldots\}$
(8) $(abc)k^*$	$\{abc, abck, abckk \ldots\}$
(9) $(a+b)(c+d+e)v$	$\{acv, adv, aev, bcv, bdv, bev\}$

Identity of RE :-

Two RE, R_1 & R_2 are equivalent if their corresponding languages are same (i.e $L(R_1) = L(R_2)$).

Identities provide some basic equivalence relation in RE.

1. $\phi + R = R + \phi = R$ [ϕ is a LE OR R is a set]
2. $\phi \cdot R = R \cdot \phi = \phi$
3. $\epsilon \cdot R = R \cdot \epsilon = R$
4. $\epsilon^* = \epsilon$ & $\phi^* = \epsilon$
5. $R + R = R$
6. $R^* \cdot R^* = R^*$
7. $R \cdot R^* = R^* \cdot R$ (commutative)
8. $(R^*)^* = R^*$
9. $\epsilon + RR^* = \epsilon + R^*R = R^*$
10. $(PQ)^* P = P(QP)^*$ (Associative)
11. $(P+Q)^* = (P^* + Q^*)^* = (P^* Q^*)^*$
12. $(P+Q)R = PR + QR$ (Distributive)
    $P(Q+R) = PQ + PR$

Construction of RE from DFA:

For every language, there's a DFA. We can also construct RE for every DFA. So if a L is accepted by DFA, then L can also be accepted by RE.

By using Arden's thm we can construct RE from a DFA.

Arden's Theorem:

Let P & Q be two RE over Σ. If P doesn't contain ϵ, then the following equation $R = Q + RP$ has a unique solution given by

$R = QP^*$

Proof:- $R = Q + RP$ — (1)

Putting the value of R on the RHS of eqn (1)

$R = Q + [Q + RP]P$

$= Q + QP + RP^2$ — (2)

By putting value of R again & again in RHS of eqn (2),

$R = Q + QP + \dots$

$R = Q + QP^*P$

$= Q(\epsilon + PP^*)$

$= QP^*$. R [as $\because \epsilon + R^*R = R^*$]

LHS

So $R = QP^*$ is a unique solution of $R = Q + RP$.

(proved)

Prove:

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

proof: LHS:

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

$= 1+00^*1[\epsilon + (0+10^*1)^*(0+10^*1)]$

$= 1+00^*1[(0+10^*1)^*][(0+10^*1)]$ [from Iq $\because \epsilon + R^*R = R^*$]

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

$= [1(\epsilon+00^*)](0+10^*1)^*$

$= 10^*(0+10^*1)^*$ [from Iq]

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

(proved)

Steps to find RE from DFA using Arden's theorem:-

(i) The transition diagram should have ε-transition.

(ii) It must have a single initial state, say q_1.

(iii) Its vertices are $q_1 \cdots q_n$

(iv) q_i is final state

(v) w_{ij} denotes the RE representing the set of labels of edges from q_i to q_j. We can get the following set of equation in $q_1 \cdots q_n$

$$q_1 = q_1 w_{11} + q_2 w_{21} + \cdots q_n w_{n1} + \epsilon \quad \text{---(1)}$$

[since q_1 is initial state]

$$q_2 = q_1 w_{12} + q_2 w_{22} + \cdots + q_n w_{n2} \quad \text{---(2)}$$

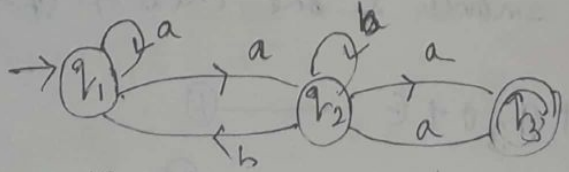
$$\vdots$$

$$q_n = q_n w_{1n} + q_n w_{2n} + \cdots + q_n w_{nn} \quad \text{---(n)}$$

NOTE:-

ε can be added to only starting state q_1 & we solve eqn to find q_i (final state) in terms of ~~DFA~~ w_{ij}.

Q1.



Consider the trans. system & find out RE.

Ans:- NO ε moves & there is only one final state.

$$q_1 = q_1 a + q_2 b + \epsilon \quad \text{---(1)}$$

$$q_2 = q_1 a + q_2 b + q_3 a \quad \text{---(2)}$$

$$q_3 = q_2 a \quad \text{---(3)}$$

putting value of q_3 from (3) to (2)

$$q_2 = q_1 a + q_2 b + (q_2 a) a$$
$$= q_1 a + q_2 b + q_2 aa$$

$q_2 = q_1 a + q_2 (b + aa)$ [$\because R = Q + RP \Rightarrow R = QP^*$]

$q_2 = q_1 a (b + aa)^*$ ——— (4)

putting equ (4) in (1)

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

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

Here dividing b to a block meaning will become

$R = \epsilon + q_1 (a + a(b + aa)^* b)$ [$\because R = Q + RP \Rightarrow R = QP^*$]

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

$q_1 = (a + a(b + aa)^* b)^*$ ——— (5)

Now find out q_2 as q_3 in terms of

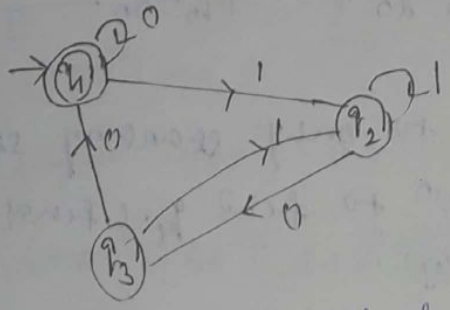
$q_3 = q_2 a$

$= q_1 a (b + aa)^* a$ [from equ (4)]

$= [(a + a(b + aa)^* b)^* a (b + aa)^* a]$ [putting equ (5)]

This is the req. ans as q_3 in terms of

2



There is no ϵ-moves & one starting state.

$q_1 = q_1 0 + q_3 0 + \epsilon$ ——— (1)

$q_2 = q_1 1 + q_2 1 + q_3 1$ ——— (2)

$q_3 = q_2 0$ ——— (3)

putting equ (3) in equ (2)

$q_2 = q_1 1 + q_2 1 + q_2 01$

$= q_1 1 + q_2 (1 + 01)$

$q_2 = q_1 1 (1 + 01)^*$ [$\because R = Q + RP \Rightarrow R = QP^*$]

(4)

$q_1 = q_1 0 + q_3 0 + \epsilon$ [eqn ① is the final state

$= q_1 0 + q_2 00 + \epsilon$ [putting eqn ③ in above eqn]

$= q_1 0 + [q_1 1 (1+01)^*] 00 + \epsilon$ [putting eqn ④ in above eqn].

$= q_1 0 + q_1 1 (1+01)^* 00 + \epsilon$

$= q_1 (0 + 1(1+01)^* 00) + \epsilon$

$q_1 = \epsilon + q_1 (0 + 1(1+01)^* 00$

$q_1 = 0 + 1(1+01)^* 00$ [$\because R = q + Rp \Rightarrow R = qp^*$]

As the final state is q_1 so we need not solve for q_3.

3

Ans: There's no ϵ-moves & one one final state.

$q_1 = q_1 0$ ——— ①

$q_2 = q_1 1 + q_2 1$ ——— ②

$q_3 = q_2 0 + q_3 (0+1)$ ——— ③.

* Here q_1 & q_2 are 2 final states so we've to find R_1 for both q_1 & q_2 i.e. $q_1 + q_2$ from eqn ① not need to solve q_3.

$q_1 = q_1 0$.

$q_1 = q_1 0 + \epsilon$. [$\because \epsilon + R = R$]

$q_1 = \epsilon + q_1 0$ [$\because R = q + Rp$].

$R = q + Rp$

$q_1 = 0^*$ ——— ④ [$\because R = q + Rp \Rightarrow R = qp^*$]

putting eqn ④ in eqn ②

$q_2 = 0^* 1 + q_2 1$

$= 0^* 11^*$ [$\because R = q + Rp \Rightarrow R = qp^*$]

$q_1 + q_2 = 0^* + 0^* 11^*$

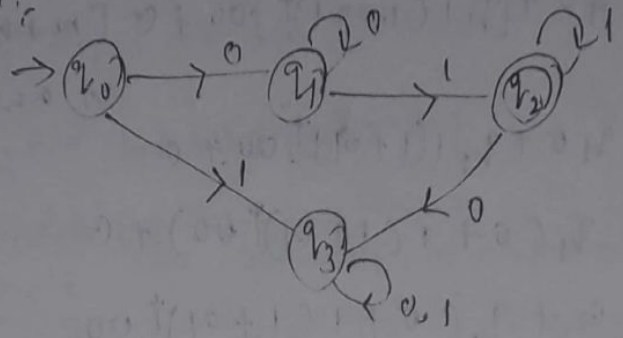
$= 0^* (\epsilon + 11^*)$

$q_1 + q_2 : 0^*1^*$ [$\because \epsilon + RR^* = R^*$]

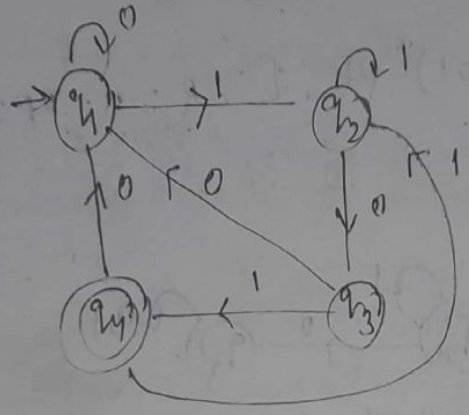
Now need to solve for q_3.

Assignment :-

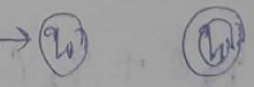
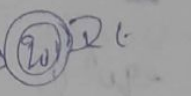
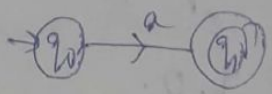
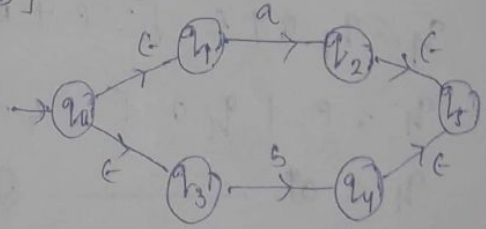
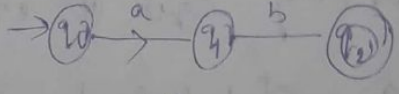
1.

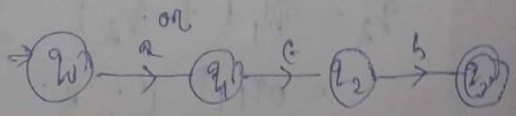


2.

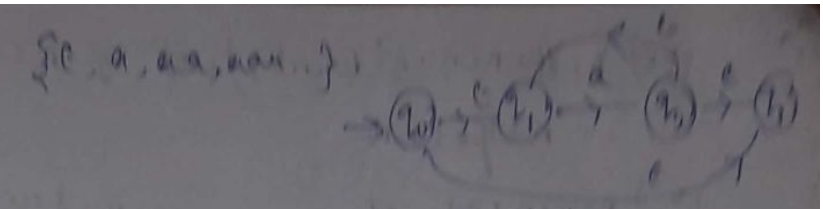


Construction of NFA from RE :-

	Reg. expression	Reg set	NFA
(1)	ϕ	$\{\}$	
(2)	ϵ	$\{\epsilon\}$	
(3)	a	$\{a\}$	
(4)	$a+b$ / $a \cup b$	$\{a, b\}$	
(5)	$a.b$	$\{ab\}$	



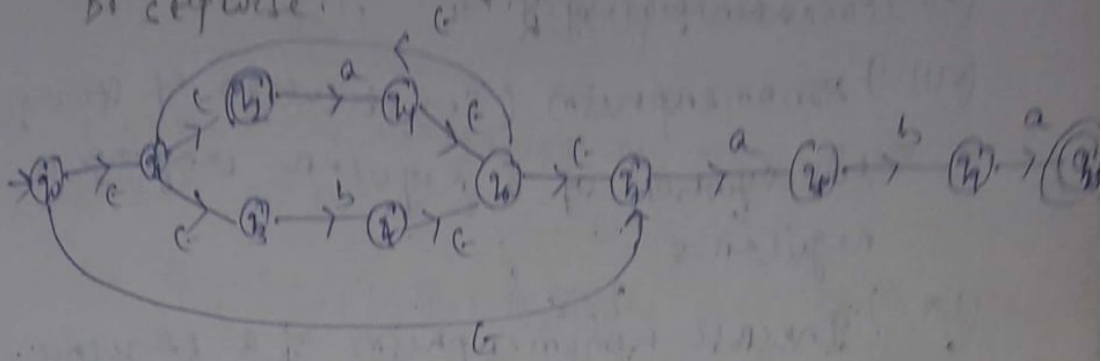
(6) a^+ {a, aa, aaa, ...}:



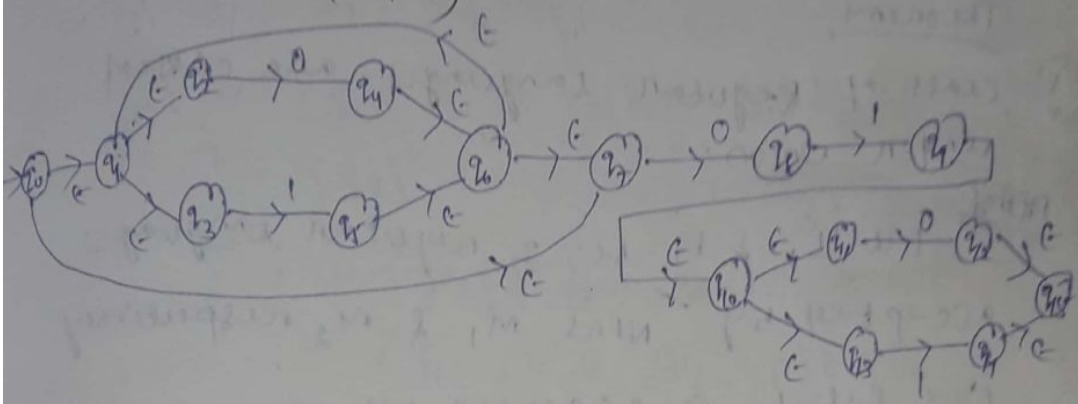
Construct ε-NFA for following RE:

1. $(a \cup b)^+ aba$

Diagram:



2. $(0+1)^+ 01(0+1)$



3. $0 + 01^+$

4. $aba^+(b \cup a)^+$

5. $(0+1)^+ 01^+(0+1)$

Closure properties of Regular sets:-

If a class of languages is closed under a particular operation, we call the fact as closure property of class of languages.

Properties:-

(i) The union of two regular languages is regular.

(ii) The intersection of 2 " " " (3) "

(iii) complement of a regular " " (3) " .

(iv) Difference ($L_1 - L_2$) of 2 regular language is regular

(v) Reversion of a regular lang is regular.

(vi) Closure (star) of a reg. " is "

(vii) Concatenation of 2 " " " "

(viii) Homomorphism (substitution of strings for symbol) of a regular language is regular.

(ix) Inverse homomorphism of a regular language is regular.

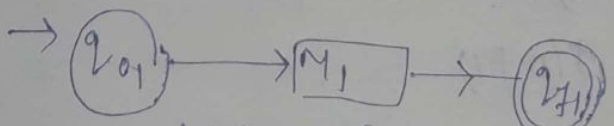
Theorems:-

(1) class of Regular languages are closed under union.

Proof:- Let L_1 & L_2 be 2 regular languages accepted by NFAs M_1 & M_2 respectively.

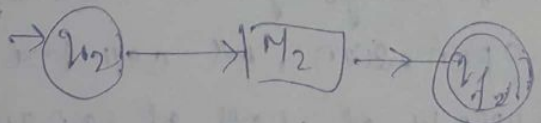
i.e Let L_1 is accepted by M_1.

or $M_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, F_1)$ recognises L_1



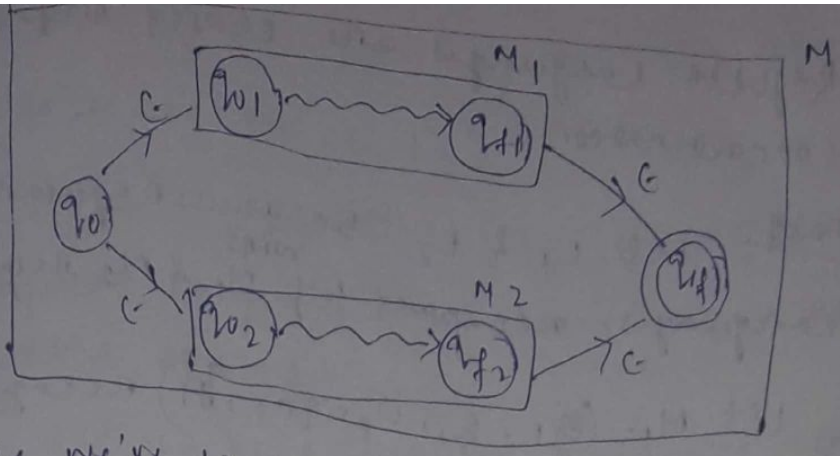
Here $\delta(q_{01}, w_1) = q_{f1}$

$M_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, F_2)$ recognises L_2



Here $\delta(q_{02}, w_2) = q_{f2}$

~~If~~ ~~L_2 be~~ Now we've to construct an NFA 'M' which accepts both the language L_1 & L_2. i.e $L_1 \cup L_2$



i.e we've to construct $M = (Q, \Sigma, \delta, q_0, F)$ which recognises $L_1 \cup L_2$

so $Q = \{q_0\} \cup Q_1 \cup Q_2 \cup \{q_f\}$ the states of M are all the states of M_1 & M_2 including q_0 & q_f

$\Sigma = \Sigma_1 \cup \Sigma_2 \cup \{\epsilon\}$

$q_0 = \{q_0\}$ q_0 is initial state of M.

$F = \{q_f\}$

δ : $\delta(q_0, \epsilon) = \{q_{01}, q_{02}\}$

$\delta(q_{f1}, \epsilon) = \delta(q_{f2}, \epsilon) = q_f$

$\delta(q_0, w_1) = \delta(q_{01}, w_1) = \delta(q_{f1}, \epsilon) = q_f$

w_1 is accepted by M

similarly

$\delta(q_0, w_2) = \delta(q_{02}, w_2) = \delta(q_{f2}, \epsilon) = q_f$

w_2 is accepted by M.

so M accepts both w_1 & w_2

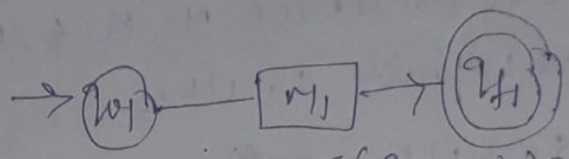
M accepts $L_1 \cup L_2$.

so $L_1 \cup L_2$ is regular.

(2) Regular languages are closed under concatenation.

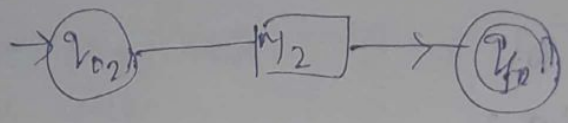
Proof: Let L_1 & L_2 be two regular languages accepted by NFAs M_1 & M_2 resp.

Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, q_{f1})$ recognises L_1.



Here $\delta(q_{01}, w_1) = q_{f1}$

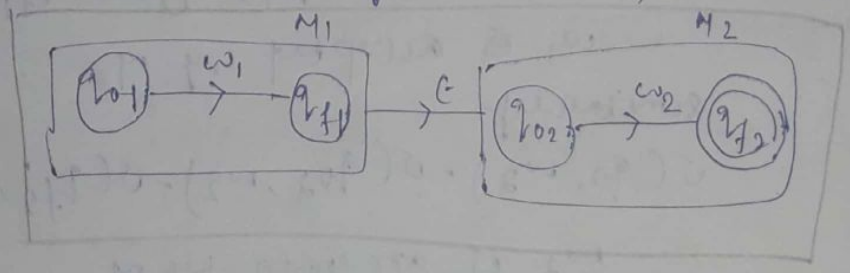
Let $M_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, q_{f2})$ recognises L_2.



Here $\delta(q_{02}, w_2) = q_{f2}$

Now we've to construct an NFA N

$N = (Q, \Sigma, \delta, q_0, q_f)$ which accepts L_1 concatenating with L_2 i.e $L_1 \cdot L_2$



So $Q: \{Q_1 \cup Q_2\}$

$\Sigma: \Sigma_1 \cup \Sigma_2 \cup \{\epsilon\}$

$q_0: q_{01}$

$F: q_{f2}$

$\delta: \delta(q_{01}, w_1) = q_{f1}$

$\delta(q_{f1}, \epsilon) = q_{02}$, $\delta(q_{02}, w_2) = q_f$

$\therefore \delta(q_{01}, w_1) = \delta(q_{f1}, \epsilon) = \delta(q_{02}, w_2) = q_f$

so $w_1 w_2$ is accepted by M.

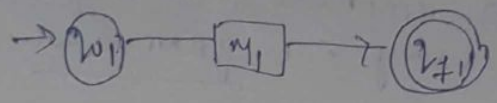
so M accepts $L_1 \cdot L_2$

$\therefore L_1 L_2$ is regular. (proved)

③ Regular languages are closed under Kleene closure operation.

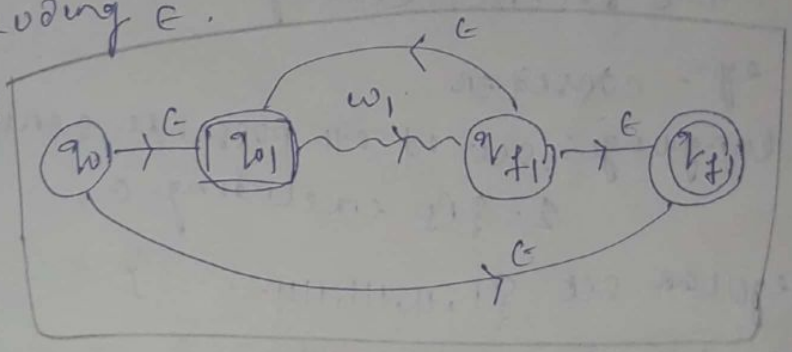
Proof: Let L_1 be a regular language accepted by NFA M_1.

i.e Let $M_1 = (Q_1, \Sigma_1, \sigma_1, q_{01}, q_{f1})$ recognise L_1



Now we've to construct an NFA M, $M = (Q, \Sigma, \sigma, q_0, F)$ that accept L_1^* i.e all the possible strings of symbols of L_1 including ϵ.

Or:



where

Q : $\{q_0\} \cup Q_1 \cup \{q_f\}$

Σ : $\Sigma_1 \cup \{\epsilon\}$

q_0 : $\{q_0\}$

F : $\{q_f\}$

σ : $\sigma(q_0, \epsilon) = \{q_{01}, q_f\}$

$\sigma(q_{01}, w_1) = q_{f1}$

$\sigma(q_{f1}, \epsilon) = \{q_{01}, q_f\}$

$\sigma(q_0, \epsilon) = \sigma(q_{01}, w_1) = \sigma(q_{f1}, \epsilon) = q_f$

$= \sigma(q_{f1}, \epsilon) = \sigma(q_{01}, w_1) = \sigma(q_{f1}, \epsilon) = q_f$

so w_1^* is accepted by M

So M accepts L_1^+.

$\therefore L_1^+$ is regular.

Regular grammar:

We've learnt three ways of characterizing regular languages: (i) Regular expressions (ii) finite automata & (iii) construction of natural language using simple operation.

Also we can express the regular language by another way i.e grammar.

→ Grammar is the set of RULES used to define language. These rules can be rewritten which are used to generate the desired string.

eg: consider

the language: set of all possible strings over $\Sigma = \{1\}$ excluding ϵ

Regular set: $\{1, 11, 111, 1111 \ldots\}$

Regular expression: 1^+

So we can also express this Reg. lang by grammar i.e Reg grammar So for grammar we need V, T, P, S to generate the strings of this lang we have one simple method:

$S \to 1$

$S \to 1S$

i.e $S \to 1 | 1S$

111 :- $S \to 1S$

$\Rightarrow 11S$ ($S \to 1S$)

$\Rightarrow 111$ ($S \to 1$)

So grammar is also an effective way to represent reg. lang.

Types of Grammar:-

① Linear & Nonlinear grammar:-

A grammar with atmost one Nonterminal at the right side of a production is a linear grammar otherwise it is a nonlinear grammar.

eg:- Linear grammar:-

1. $S \to aSb$

   $S \to \epsilon$

2. $S \to b$

   $A \to aA$

   $A \to \epsilon$

eg:- Nonlinear grammar:-

1. $S \to AS$

   $S \to \epsilon$

   $A \to aSb$

2. $S \to aA$

   $A \to \epsilon$

   $B \to aSAS$

② Left Linear & Right Linear grammar:-

Left Linear:-

A grammar G is left linear if all the production are of the form

$A \to Bx$ or $A \to x$, $A, B \in V$ and $x \in T^*$

Example:-

1. $S \to Aab$
   $A \to Aab \mid B$
   $B \to a$

2. $S \to Ab$
   $S \to Sb$
   $A \to \epsilon$

Right linear:

A grammar G is said to be right linear if all productions are of the form

$A \to xB$ or $A \to x$

where $A, B \in V$ & $x \in T^*$

Example: (1) $S \to abS$
$S \to a$

(2) $S \to A$
$A \to aA$
$A \to \epsilon$

Regular grammar:

A grammar $G = \{V, T, P, S\}$ is said to be regular, if it is either right linear or left linear.

eg: 1. $S \to abS$
$S \to a$

2. $S \to AAb$
$A \to Aab \mid B$
$B \to a$

Nonregular grammar:

A grammar $G = \{V, T, P, S\}$ is said to be nonregular, if it is neither right linear nor left linear.

eg: 1. $S \to aSA$
$S \to \epsilon$

2. $S \to Ab$
$A \to aAb$
$A \to \epsilon$

3. $S \to \epsilon$
$S \to aSb$
$S \to bSa$

Regular Grammar from DFA:

Given: DFA i.e. transition diag. of DFA.

To find out: Regular grammar.

Steps: Regular grammar is a set of production rules which is useful to make a language regular.

We know that $G = (V, T, P, S)$

So, Regular grammar G from a given DFA can be considered as

1. $G = (\{A_0, A_1, A_2 \cdots A_n\}, \Sigma, P, A_0)$

2. where $V = \{A_0, A_1, A_2 \cdots A_n\}$

3. $S = A_0$: initial state for w

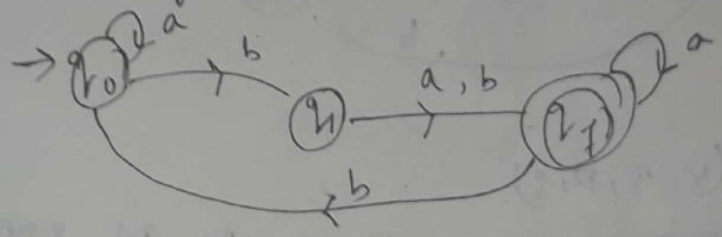
A_1 : next state q_1

4. $T = \Sigma$, ~~$S = A_0$~~

5. P : production rule & can be defined as:

1. $A_i \to aA_j$ in P if $\delta(q_i, a) = q_j \notin F$
2. $A_i \to aA_j$ & $A_i \to a$ in P if $\delta(q_j, a) = q_j \in F$

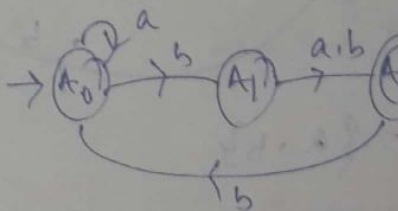
Ex 1: Construct a Reg. grammar G for the following DFA



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

Here $V = \{A_0, A_1, A_2\}$, i.e we consider states A_0, A_1, A_2 variables in place of q_0, q_1, q_2. — Now Transition Dig. will be

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



$S = A_0$

P: $A_0 \to aA_0$ ($\because \delta(A_0, a) = A_0 \notin F$)

$A_0 \to bA_1$ ($\because \delta(A_0, b) = A_1 \notin F$)

$A_1 \to aA_2$ ($\because \delta(A_1, a) = A_2 \in F$)

$A_1 \to bA_2$ ($\because \delta(A_1, b) = A_2 \in F$)

$A_1 \to a$ ($\because \delta(A_1, a) = A_2 \in F$)

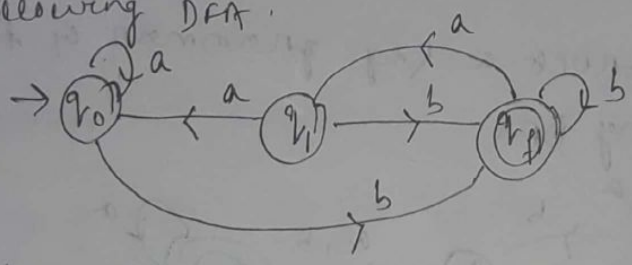
$A_1 \to b$ ($\because \delta(A_1, b) = A_2 \in F$)

$A_2 \to aA_2$ } ($\because \delta(A_2, a) = A_2 \in F$)

$A_2 \to a$

$A_2 \to bA_0$ ($\because \delta(A_2, b) = A_0 \notin F$)

2. Construct a regular grammar G for the following DFA.

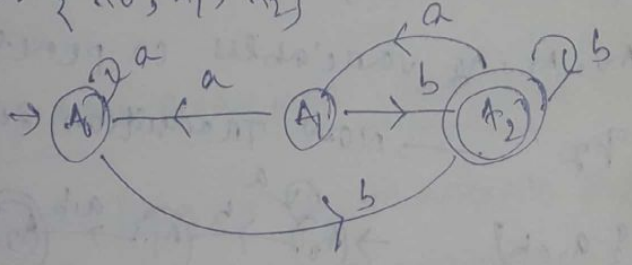


Ans:

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

Consider states A_0, A_1, A_2 as variable in place of q_0, q_1 & q_f

$V = \{A_0, A_1, A_2\}$



$T = \{a, b\}$

$S : A_0$

P: $A_0 \to aA_0$ ($\because \delta(A_0, a) = A_0 \notin F$)

$A_0 \to bA_2$ } ($\because \delta(A_0, b) = A_2 \in F$)

$A_0 \to b$

$A_1 \to bA_2$
$A_1 \to b$ } $\because \delta(A_1, b) = A_2 \in F$
$A_1 \to aA_0$ $\because \delta(A_1, a) = A_0 \notin F$
$A_2 \to bA_2$
$A_2 \to b$ } $\because \delta(A_2, b) = A_2 \in F$

$A_2 \to aA_1$ $(\because \delta(A_2, a) = A_1 \notin F)$

Regular grammar from NFA or ε-NFA) :-

Step 1 :- $G = (V, T, P, S)$

V : $A_0, A_1 \cdots A_n$

T : Σ

S : A_0

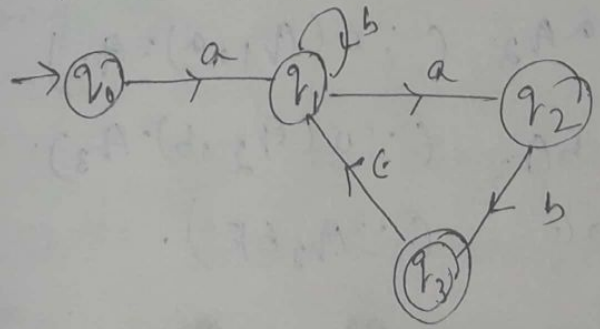
P : a) if $q_i \xrightarrow{a} q_j$

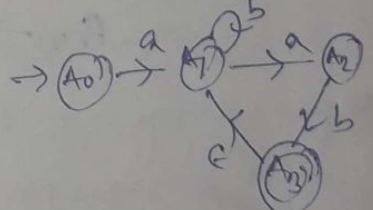
$q_i \to a q_j$ if $\delta(q_i, a) = q_j$

b) if q_f

P : $q_f \to \epsilon$

Ex1 Construct a Regular grammar from the following finite automaton.





Ans :- $G = (V, T, P, S)$

consider states A_0, A_1, A_2, A_3 as variable in place of q_0, q_1, q_2, q_3

$V = \{A_0, A_1, A_2, A_3\}$ $T = \Sigma = \{a, b\}$

Then, the rule

S: A_0

P: $A_0 \to aA_1$ ($\because \delta(A_0, a) = A_1$)

$A_1 \to bA_1$

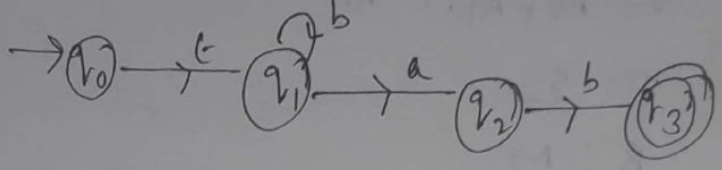
$A_1 \to aA_2$

$A_2 \to bA_3$

$A_3 \to \in A_1$ & $A_3 \to A_1$ ($\delta(A_3, \in) = A_1$)

$A_3 \to \in$ ($\because A_3 \in F$)

2.

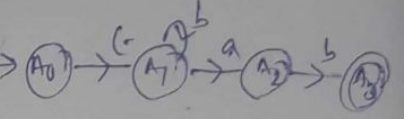


Ans:

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

Consider states A_0, A_1, A_2, A_3 as variables in place of states q_0, q_1, q_2, q_3

$V = \{A_0, A_1, A_2, A_3\}$



$T = \Sigma = \{a, b\}$, $S \to A_0$

P: $A_0 \to \in A_1$ ($\because \in A_0 \to A_1$ ($\because \delta(A_0, \in) = A_1$))

$A_1 \to bA_1$ ($\because \delta(A_1, b) = A_1$)

$A_1 \to aA_2$ ($\because \delta(A_1, a) = A_2$)

$A_2 \to bA_3$ ($\because \delta(A_2, b) = A_3$)

$A_3 \to \in$ ($\because A_3 \in F$).

Pumping Lemma for Regular Set:-

Pumping lemma for regular language is used to recognise all non-regular language. i.e

pumping lemma is a tool for proving certain languages are not regular. i.e whether a language is accepted by finite automata or not.

Pumping lemma is based on the principle of pigeon hole principle which states that if n pigeons are placed in m holes & if n>m, then atleast 1 hole must contain more than one pigeon in it.

We'll discuss 2 following things here

(1) State & proof pumping lemma

(2) Prove following grammar is not regular / Prove a given language is not regular.

NOTE:

Pumping means generating many i/p strings from a given string such that all of them (strings) are in regular language. &

Lemma: helping theorem (used to proof for nonregularity)

(1) State & Prove Pumping Lemma:

Thm:- L is a regular set accepted by finite automata M with 'n' states & with a string w in L i.e $w \in L$ written as $w = xyz$, then a) $|y| \geq 1$

b) $|xy| \leq n$

c) $xy^iz \in L \ \forall \ i \geq 0$ where

where y^i denote repeated i times & $y^0 = \epsilon$.

Proof:-

Let M be the Deterministic Finite Automata corresponding to the language 'L' & n be the number of states. Then 'n' is a defined constant.

→ Let w be a string of language L of length m i.e Let $w \in$ such that $m \geq n$.

Let $w = a_1 a_2 a_3 \ldots a_{m-1} a_m$

→ Let q_0 be the initial state for automaton M & δ be the transition function.

→ Then $\delta(q_0, a_1 a_2 a_3 \ldots a_{m-1} a_m)$ leads to the final state by passing through the various states in M.

$\delta(q_0, a_1 a_2 \ldots a_m) = q_i$ for $i = 1, 2, \ldots m$ i.e Q is the sequence of states in the path with path value $w = a_1 a_2 \ldots a_{m-1} a_m$.

→ Since $m \geq n$, all these states can't be distinct so a repetition is bound to occur. i.e Atleast two states in Q must coincide.

→ Let the first repetition occurs at $q_j = q_k$, i.e among various pairs of repeated states, we take first pair as $q_j = q_k$ such that $0 \leq j < k \leq n$ with a condition.

The string w can be decomposed into 3 substring/parts as

$w = a_1 a_2 a_3 \dots a_m$ (the tightly we need)

Dividing into 3 parts

$x = a_1 a_2 \dots a_j$

$y = a_{j+1} a_{j+2} \dots a_k$ &

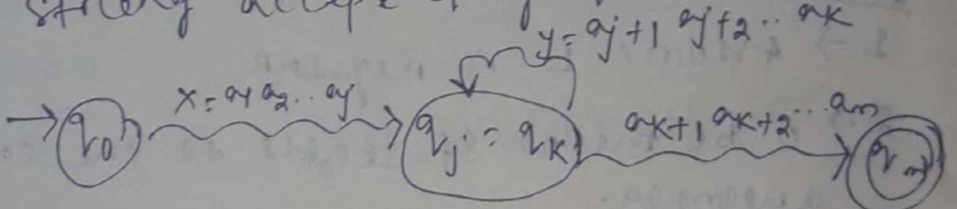
$z = a_{k+1} a_{k+2} \dots a_m$

$w = xyz$

$= \underbrace{a_1 a_2 \dots a_j}_{x} \underbrace{a_{j+1} a_{j+2} \dots a_k}_{y} \underbrace{a_{k+1} a_{k+2} \dots a_m}_{z}$

$\therefore$ As $|xy| \leq n$.

String accepted by M will be



The decomposition is valid only for strings of length greater than or equal to the number of states.

The path with path value w in the transition Dig. of M :-

→ The automaton M starts from the initial state q_0. On applying the string x, it reaches to $q_j (= q_k)$. On

→ On applying the string 'y' it comes back to $q_j (= q_k)$. So after application of y^i for each $i \geq 0$, the automaton is in same state $q_j = q_k$.

Every repetition of y keeps the acceptance path unchanged, i.e. not affecting the string acceptance. Hence $xy^0z, xy^1z, xy^2z, \dots xy^iz \in L$

Let $i = 0$, $y^0 = \epsilon$ so ~~any~~ $x \cdot x^{\overline{y}} \in z : xz$

i.e by reading x automaton reaches to $q_j : q_k$ & again on reading to z, it reaches to q_m which is final state.

So xy^0z is accepted.

→ On applying z, it reaches to q_m which is a final state.

So for all $i \geq 0$ xy^iz is accepted by M.

(proved)

To Prove given language is non regular using pumping lemma:

Steps:

1 → Assume 'L' is regular

2 → 'n' be the no. of states to the corresponding automata.

3 → Choose a string w such that $|w| \geq n$

4 → Decompose / consider $w = xyz$ such that

a) $|y| \geq 1$

b) $|xy| \leq n$

c) Show that $xy^iz \notin L$ for $i \geq 0$

5 → Find a suitable choosen 'i' such that $xy^iz \notin L$.

This contradicts our assumption that L is regular.

So L is not regular.

Q.1 Show that the language $L = \{ww^R | w \in \{a,b\}^*\}$ is non regular.

Ans: **Step 1:** Assume L is regular & n be no. of states.

Step 2: $w = a^n b^n a^n b^n$

$|w| \geq n$ [as $|w| = 4n$].

Step 3: Decompose w as xyz such that

$|y| \geq 1$ & $|xy| \leq n$

i.e. w

x y z

a^{n-1} a $b^n a^n b^n = a^n b^n a^n b^n$

Step 4: $w = xy^i z$

$= a^{n-1} a b^n a^n b^n$ [here $i = 1$]

By putting $i = 2$,

$xy^i z = a^{n-1} a^2 b^n a^n b^n$

$= a^{n+1} b^n a^n b^n \notin L$ which contradicts our assumption as here powers are different where as these powers must be same.

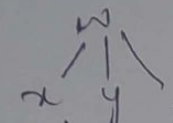
So L is not regular.

Q. Check whether the language $L = a^p$ is regular or nonregular.

Ans: Step 1: Assume L is regular & p be the no. of states.

Step 2: $w = a^p$, $|w| \geq p$ [$\because |w| = p$]

Step 3: Decompose $w = xyz$ such that $|w| \geq p$ i.e



But z can't be ε i.e z should be some string i.e a^{k+1} and but it can't be only y can be ε.

$|w| = p$ where $|y| = 1$ [$\because y = a$]

so $|w| = |x| + |y| + |z| = p$

$= |x| + 1 + |z| = p$

$\Rightarrow |x| + |z| = p - 1$ ——①

Let $w' = xy^i z$ $\quad i \geq 2$

$|w'| = |x| + i|y| + |z|$

$= p - 1 + i$ [using ①] ——②

If $w' \in L$ then it satisfies pumping lemma

$w' = a^p$

$|w'| \geq p$ where p = prime no.

As $|w'| = p - 1 + i$ [from ②].

& $w' \in L$ hence

$p - 1 + i$ must be an prime no. $\forall i$

Let $p=7$, $i=2$

$\therefore |w'| = p-1+i = 7-1+2 = 8$ where 8 is not prime no.

Hence $|w'| \notin L$ [acc. to pumping lemma]

$\therefore L$ is not regular.

* $w = xy^i z$

$= a^{p-1} a^i = a^p$ [$i=1$]

if $i=2 \Rightarrow a^{p-1} a^2 = a^{p+1}$ [$\because p$ = prime no]

if we'll put $p=3$ then $p+1 = 4$ which is not prime.

so L is not regular.

3) Show that the language $L = \{a^n b^n : n \geq 0\}$ is not regular.

Ans: Step 1: Assume L is regular & 'n' be the no. of states.

Step 2: $w = a^n b^n$

$|w| \geq n$ [$\because |w| = 2n$]

Step 3: Decomposing $w = xyz$ such that

$|y| \geq 1$, $|xy| \leq n$

w

x y z Here $|y| = |a| \geq 1$

a^{n-1} a b^n $|xy| = |a^{n-1} b^n| \leq n$

$w = xy^i z$

$= a^{n-1} a b^n$ [Here $i=1$]

By putting $i = 2$,

$w = a^{n-1} a^2 b^n = a^{n+1} b^n \notin L$

It contradicts our assumption. So L is not regular.

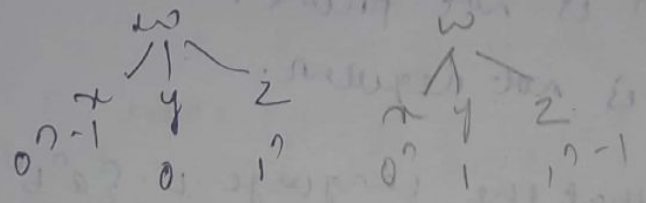
4. Prove that the language $L = \{0^i 1^i : i \geq 1\}$ is not regular.

Ans: Step 1: Assume L is regular & n be the no. of states.

Step 2: Let $w = 0^n 1^n$

$|w| \geq n \quad [\because |w| = 2n]$

Step 3: Devide $w = xyz$



such that $|y| \geq 1$ & $|xy| \leq n$

$w = xy^i z \in L$

$= 0^{n-1} 0^i 1^n \quad [\text{Here } i = 1]$

By putting $i = 2$

$= 0^{n-1} 0^2 1^n = 0^{n+1} 1^n \notin L$

∴ It contradicts our assumption.

So L is not regular.

5. Show that the language $L = \{a^{n!} : n \geq 0\}$ is not regular.

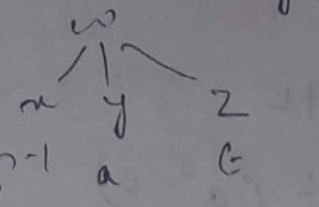
Ans: Step 1:

Assume L is regular & n be no. of states.

Step 2: let $w = a^n$

$|w| \geq n$ [$\because |w| = n$]

Step 3: Deviding $w = xyz$



* x & z can be ϵ but $y \neq \epsilon$

such that $|y| = \geq 1$ [$\because |y| = |a| = 1$]

$|xy| \leq n$ ✓

$w = xy^i z \in L$

$= a^{n-1} a^i \in L$ [$\because i = 1$]

$= a^n \in L$

putting $i = 2$

$= a^{n-1} a^2 \in L$

$= a^{n+1} \notin L$

$i = 0$

$a^{n-1} a^0 \in L$

$a^{n-1} \notin L$

$a^1 = a^1 a^0$ $\quad a^2 = a^2 a^0$

so it contradicts our assumption.

$\therefore L$ is not regular.

6. Show that the language $L = \{a^n b^{n+L} c^{n+L} : n, L \geq 0\}$ is not regular.

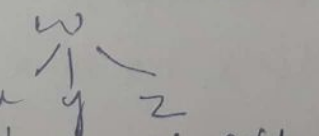
Ans: Step 1: Assume L is regular & n : no. of states

Step 2: $w = a^n b^L c^{n+L}$ s.t $|w| \geq n$

[$\because |w| = n + L + n + L$

$= 2n + 2L \geq n$]

Step 3: Deviding $w = xyz$ such that



such that $|y| \geq 1$

$|xy| \leq n$

$w = xy^i z$

$= a^{n-1} a\, b^L c^{n+L}$ [$\because i = 1$]

putting $i = 0$

$= a^{n-1} a^0 b^L c^{n+L}$

$= a^{n-1} \in b^L c^{n+L}$ ($a^0 = \in$)

$= a^{n-1} b^L c^{n+L}$

$\notin L$

So it contradicts our assumption.

$\therefore$ L is not regular.

CONTEXT FREE GRAMMAR

Grammar :-

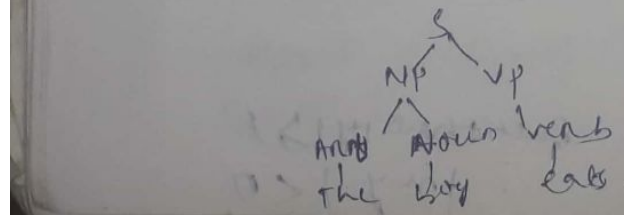
It's a set of formal rule which check the correctness of sentences by using certain rules.

CFG is type 2 grammar. The lang not accepted by types may be accepted by type 2 grammar i.e CFG.

CFGs were first used in the study of natural language like english. CFGs are also used in the design & implementation of compilers.

The original motivation of grammar was description of natural language.

eg:- "The boy eats" its grammar will be



Production Rules:-

$S \rightarrow \langle NP \rangle \langle VP \rangle$
$NP \rightarrow \langle ART \rangle \langle NOUN \rangle$
$ART \rightarrow the$
$NOUN \rightarrow boy$
$NP \rightarrow verb$
$verb \rightarrow eats$

"$\rightarrow$" rewritten as

Derivation Rules:-

$S \Rightarrow \langle NP \rangle \langle VP \rangle$
$\Rightarrow \langle ART \rangle \langle NOUN \rangle \langle VP \rangle$
$\Rightarrow the \langle NOUN \rangle \langle VP \rangle$
$\Rightarrow the\ boy \langle VP \rangle$
$\Rightarrow the\ boy\ eats$

sentential forms

$\Rightarrow$ derives

Context free language:-

Language accepted by CFG is called as context free language.

Assignment:- Differentiate between RL & CFG.

Structure of a Grammar:-

Grammars not only produce natural languages but also formal ones.

If $A \subseteq L$ i.e L is a language over $\Sigma = \{a, b\}$ then a grammar for L consists of a set of rules of the form

$X \rightarrow Y$ — production Rule

Nonterminal Terminal

where X is NT & Y is T and are disjoint sets.

NOTE:-

If 'S' is the start symbol for a grammar then there must be at least one production of the form $S \rightarrow y$ [where y: terminal]

* LHS of production must be a single NT.

eg:- $A \rightarrow 0A1$ A → start symbol.

$A \rightarrow B$ A, B → Nonterminal

$B \rightarrow \#$ 0, 1, # → Terminal

Formal Definition of CFG:-

CFG is a 4 tuple i.e

$G = V, T, P, S$

where V: set of variables or Nonterminal.

T: set of terminals.

P: set of production RULE

S: start symbol; $S \in V$

Nonterminal:-

The symbols that may be replaced by other symbol are called nonterminal or variable.

Terminal:-

The symbols that can't be replaced by other symbol are called terminal.

Production:-

Each production is of the form $A \rightarrow \alpha$ where $A \in V$ & $\alpha \in (V \cup T)^*$

NOTE:-

If there's 'n' no. of variables, then it must have atleast 'n' no. of productions.

Types of Production:-

Productions are of 4 types.

(i) Unit production
(ii) ∈ production
(iii) Direct / Recursive production
(iv) Indirect / Nonrecursive "

i) Unit production:-

Any production of the form $A \to B$ is called as unit production.

eg:- $S \to Aa \mid B$

Here $S \to B$: unit production
$S \to Aa$: Not unit "

ii) ∈ - production:-

Any production of the form $A \to \in$ is called as ∈-production.

iii) Direct / Recursive:-

A production is called recursive if its lefside occurs on its Right side i.e same nonterminal must present on both sides of $\to$

eg:- $A \to \alpha A 1$.

iv) Indirect / Nonrecursive:-

A production which is not directly recursive is called as nonrecursive production.

eg:- $S \to bA$
$A \to S$

Backus Normal form (BNF):-

The compact notation used to represent the productions rule, is called BNF.

eg:- $S \to SS$
$S \to a$
$S \to \epsilon$

BNF: $S \to SS \mid a \mid \epsilon$

CFL:-

The language generated by CFG is CFL.

→ The language generated by G is the string in $L(G)$ if

(i) The string consists of solely of terminals

(ii) The string can be derived from S.

Sentential Form:-

A string of terminals & variables (except the last string) is called sentential form.

Sentential form: $(V \cup T)^*$

It's of 2 types:

(i) Left Sentential Form:-

If the sentential form is obtained by applying the LMD, then that is called left sentential form.

(ii) Right Sentential Form:-

If the sentential form is obtained by applying an RMD, then that is called as right sentential form.

Derivation:-

The sequence of substitution of productions to obtain a string is called as derivation.

[Derivation steps when the resulting string contains only terminal symbol

eg:- consider the production rule

$S \rightarrow 0S1$

$S \rightarrow \epsilon$

Derive 0011

Derivation :- $S \Rightarrow 0S1$

$\Rightarrow 00S11$ (∵ $S \rightarrow 0S1$)

$\Rightarrow 00\epsilon 11$ (∵ $S \rightarrow \epsilon$)

$\Rightarrow 0011$

∴ so 2 types of arrows are used

$\rightarrow$ (rewrites as) used in production rule

$\Rightarrow$ (derives) used in derivation rule.

Types of Derivation :-

Derivation is of 2 types i.e we can derive a string by 2 types.

(i) Left Most Derivation (LMD)

(ii) Right Most Derivation (RMD)

LMD :-

A derivation is said to be LMD if in each step, the left most nonterminal in the sentential form is replaced.

eg:- Let the production rule:

$E \rightarrow E+E$

$E \rightarrow E*E$

$E \rightarrow id$

Derive $id+id*id$ using LMD :-

$E \Rightarrow E+E$

$\overset{LMD}{\Rightarrow} id+E$ (∵ $E \rightarrow id$)

$\overset{LMD}{\Rightarrow} id+E*E$ (∵ $E \rightarrow E*E$)

$\xrightarrow{LMD}$ id + id * E

$\xrightarrow{LMD}$ id + id * id

NOTE:-

$w_1 \Rightarrow w_n$ means

w_n is derived from w_1 in one step.

$w_1 \overset{*}{\Rightarrow} w_n$ means

w_n is derived from w_1 in more than one step.

RMD:-

A derivation is said to be rightmost if in each step, the rightmost nonterminal in the sentential form is replaced.

eg: considering the above example

Derive id + id * id using RMD.

Ans:- $E \Rightarrow E + E$

$\xrightarrow{RMD}$ E + E * E (∵ E → E * E)

$\xrightarrow{RMD}$ E + E * id (∵ E → id)

$\xrightarrow{RMD}$ E + id * id (∵ E → id)

$\xrightarrow{RMD}$ id + id * id

Derivation Tree (parse Tree / production Tree / Generation Tree)

The graphical representation of derivation as a tree known as derivation tree.

Properties of derivation Tree:-

1) Root of the tree is labelled with start symbol of the grammar.

2. All leaf nodes are labelled with terminals of the grammar.

3. All interior nodes are labelled with nonterminal

4. The derivation is read from left to right.

5. If an interior node has label A & it has n childs like $x_1 x_2 \cdots x_n$ from left to right, then the production rule $A \to x_1 x_2 x_3 \cdots x_n$ must exist in the grammar.

eg:- For the grammar G with production rule

$E \to E + E$

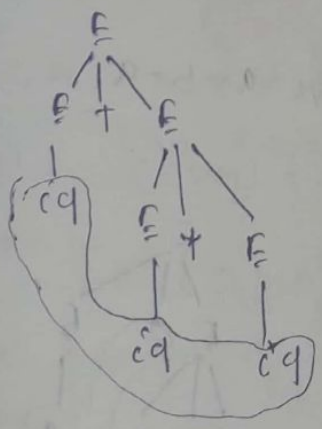
$E \to E * E$

$E \to id$

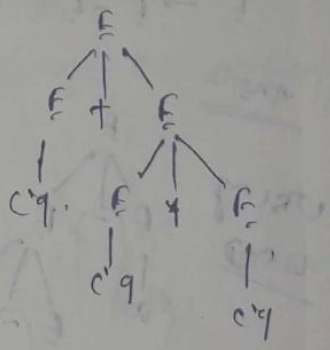
construct derivation tree for id + id * id

Ans:-

using LMD:-



using RMD:-



Ambiguity:-

A grammar is said to be ambiguous, if more than one derivation tree can be constructed either using LMD or RMD for a given string.

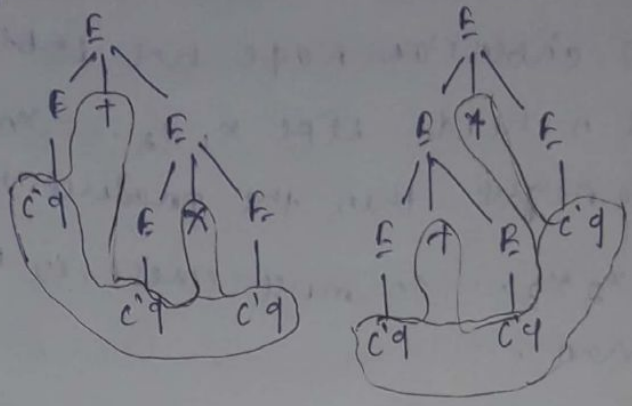
Q. Show that the given grammar is ambiguous or not:

$E \to E+E$

$E \to E*E$

$E \to id$ derive id+id*id

Ans:- Using LMD :-



So here as more than one derivation tree is possible using LMD for the string id+id*id, so the grammar is ambiguous

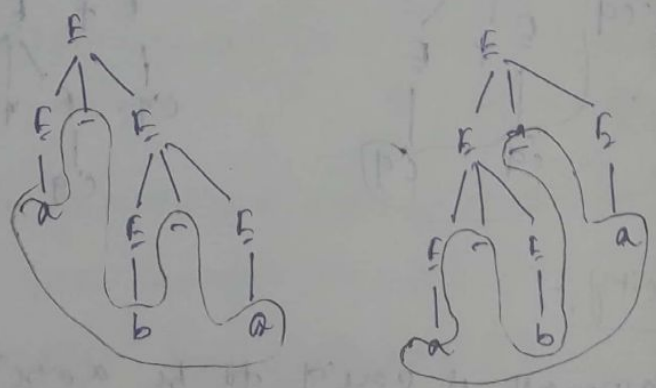
Q. Show that the given grammar is ambiguous or not. Also derive the string using LMD & RMD.

$E \to a/b$

$E \to E-E$, Derive a-b-a

Ans:-

Using LMD



So it's ambiguous.

3. $S \to aAS \mid a$
$A \to SbA \mid SS \mid ba$
Derive aabbaa

$S \to 0S1 \mid 01$, $w = 000111$

4. $S \to +SS \mid *SS \mid a$, $w = +*aaa$

5. $S \to S(S)S \mid \in$, $w = (()())$

4. $S \to S+S \mid SS \mid (S) \mid S* \mid a$

5. $w = (a+a)*a$

6. $S \to (L) \mid a$
$L \to L,S \mid S$
$w = ((a,a),a,(a))$

7. $S \to aSbS \mid bSaS \mid \in$
$w = aabbab$.

Language of CFG:

eg

eg 1: For the given strings sets
$\{\in, a, aa, aaa \ldots a^n\}$ find out.

i) $L(G) = L(CFG)$ i.e language accepted by CFG: $\{a^n \mid n \geq 0\}$

(ii) what grammar is required to derive a string?
Ans: $S \to \in$
$S \to aS$

(iii) what is the regular expression?
Ans: $RE \to a^*$

eg: $\{\epsilon, ab, aabb, aaabbb \ldots a^n b^n\}$

① $L(G) = \{a^n b^n \mid n \geq 0\}$

② grammar :-

$S \to \epsilon$

$S \to aSb$

③ RE :- $(ab)^*$

DESIGN OF A CFG :-

Q construct a CFG to generate a set of palindromes over $\Sigma = \{a, b\}$

step 1 : understand the language specified by listing the example of strings in the language.

$L(G) = \{\epsilon, a, b, (aabbaa, bwb\}$ are palindromes.

~~i.e ϵ is a pa~~

step 2 : Determine the base case productions of CFG by listing the shortest / smallest strings in the language.

i.e ϵ is a palindrome.

a, b are palindromes.

If w is a palindrome then the strings awa, bwb are palindromes.

Let the CFG, $G = (V, T, P, S)$ where

$V = \{S\}$

$T = \{a, b\}$

S : start symbol.

identify the rules in the language

smallest sentence

$P = \{ S \to \epsilon \mid a \mid b \mid c$
$S \to a S a$
$S \to b S b \}$

4) Test the CFG obtained by enumerating examples.

Test for abbba. [Since it is a palindrome]

i.e $S \Rightarrow a S a$
$\Rightarrow a b S b a$ [$S \to b S b$]
$\Rightarrow a b b b a$ [$\because S \to b$]

2. Construct a CFG to generate $L(G) = \{a^n b^n \mid n \geq 0\}$

Ans: $L(G) = \{ \epsilon, ab, aabb \ldots \}$

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

$V = \{S\}$

$T = \{a, b\}$

S : starting symbol

$P = \{ S \to \epsilon$
$S \to a S b \}$.

Test for aabb :-

$S \Rightarrow a S b$ [$\because S \to a S b$]
$\Rightarrow a a S b b$ [$\because S \to a S b$]
$\Rightarrow a a \epsilon b b$ [$\because S \to \epsilon$]
$\Rightarrow a a b b$

3. Construct a CFG to generate $L(G) = \{a^{2n} b \mid n \geq 1\}$.

Ans: $L(G) = \{ abb, aabbb \ldots \}$.

$P = \{ S \to abb \mid S \to a S bb \}$.

4. Construct a CFG to generate the lang. $L(G) = \{ww^R \mid w \in \{a,b\}^*\}$ where w^R is the reverse of w.

$L(G) = \{\epsilon, aa, bb, abba \ldots\}$

ϵ belongs to $\{a,b\}^*$.

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

$V = \{S\}$

$T = \{a, b\}$

S : start symbol.

P : $\{S \to \epsilon$

$S \to aSa$

$S \to bSb\}$

Test for abba :-

$S \Rightarrow aSa$

$\Rightarrow abSba$ [$\because S \Rightarrow bSb$]

$\Rightarrow ab\epsilon ba$ [$\because S \to \epsilon$]

$\Rightarrow abba$

5. Construct a CFG to generate the language $L(G) = \{wcw^R \mid w \in \{a,b\}^*\}$.

$L(G) = \{c, aca, bcb, abcba \ldots\}$

P : $S \to c$

$S \to aSa$

$S \to bSb$

Test for abcba :-

$S \Rightarrow aSa$

$\Rightarrow abSba$ [$\because S \to bSb$]

$\Rightarrow abcba$ [$\because S \to c$]

Q. Consider a CFG over $\Sigma = \{a, b, c, d, e, +, -, \times, /, (,)\}$ to generate $(a+b) \times c / (d \times e)$.

Ans:-

$S \to a \mid b \mid c \mid d \mid e$

$S \to S+S$

$S \to S \times S$

$S \to S/S$

$S \to (S)$

To generate $(a+b) \times c / (d \times e)$:-

$S \Rightarrow S/S$

$\Rightarrow S \times S / S \quad [\because S \to S \times S]$

$\Rightarrow (S) \times S / S \quad [\because S \to (S)]$

$\Rightarrow (S+S) \times S / S \quad [\because S \to S+S]$

$\Rightarrow (a+S) \times S / S$

$\Rightarrow (a+b) \times S / S$

$\Rightarrow (a+b) \times c / S \Rightarrow (a+b) \times c / (S) \quad [\because S \to (S)]$

$\Rightarrow (a+b) \times c / (S \times S)$

$\Rightarrow (a+b) \times c / (d \times S) \Rightarrow (a+b) \times c / (d \times e) \in L$

Simplification of CFG :-

In a CFG, it's not necessary to use all the symbols in V or all the productions in P, for deriving sentences.

So, we can reduce the complexity of the grammar without reducing the generating power of CFG.

This can be done by the following procedures.

(i) By eliminating the useless symbols

(ii) By eliminating ~~ε-rule~~ unit productions

(iii) By eliminating ε productions, if ε is not included in the sentences

Elimination of useless symbol :-

Useful & useless symbol :-

Useful →

A grammar symbol X in CFG G useful, if & only if

a) it derives a string of terminals
b) it is used in derivation of atleast one w in L(CFG)

Useless →

A grammar symbol X in CFG G useless, if and only if

a) it doesn't derive a string of terminals
b) it doesn't occur in a derivation sequence of any w in L(CFG).

How to find out useful & useless symbol in a grammar :-

(1) Identify the non-terminals from which the terminal strings are derived.

(2) Identify variables which are reachable from start state.

Q1:- Eliminate useless symbol from the given grammar.

$S \rightarrow a \mid AB$

$A \rightarrow a$

Ans:

(i) Find variables from which terminal strings are derived.

$S \to a$.

$A \to a$

Here B does'nt derive the terminal string. So B = useless symbol.

(ii) Find variables which are reachable from start state that derives terminal symbols.

i.e $S \to a$

or

$S \Rightarrow AB$

$\Rightarrow aB$ $[A \to a]$

└ But it's not string of terminals.

Though $A \to a$ but it's not used in derivation of a string (of terminals) from starting symbol.

So A & B are useless symbol

P: $S \to a$ [After removing useless symbol]

2. Eliminate useless symbol from the grammar

$S \to aS \mid A \mid C$

$A \to a$

$B \to aa$

$C \to aCb$

Ans:- (i) find variables from which terminal strings are derived.

$A \to a$

$B \to aa$

(ii) C is eliminated because it's not cause of any string. ~~from start~~ Because No double

$S \to C$

$\Rightarrow acb$

$\Rightarrow aacbb$

as C doesn't derive any terminal it's a NT. so it's as useless symbol.

After removing C

(iii) $S \to aS \mid A$

$A \to a$

$B \to aa$

B is also eliminated because it's not cause of any string from starting symbol.

i.e $B \to aa$ [.

so B is useless symbol.

But S & A are useful because

$S \to aS$ $\quad S \to A$

$\Rightarrow aA$ $\quad S \to a$

$\Rightarrow aa$

∴ B & C are useless symbol.

After removing useless symbol,

P: $S \to aS \mid A$

$A \to a$

Eliminate useless symbol from the grammar

$S \to aAa$

$A \to bBB$

$B \to ab$

$C \to aB$

Ans: ① C is useless symbol as it's not cause of a string from starting symbol.

After removing 'C',

$P: S \to aAa$

$A \to bBB$

$B \to ab$.

② finding useful symbol.

$S \to aAa$

$\Rightarrow abBBa$ [$A \to bBB$]

$\Rightarrow abababa$ [$B \to ab$].

So S, A, B variables are useful because these are cause of deriv' of string of terminals from starting symbol

Elimination of ε-production:-

If a production is of the form $A \to \varepsilon$, then it's called as ε-production. A : Nullable nonterminal. We've to eliminate it.

1. Eliminate ε-production from the given grammar.

$S \to aSa$

$S \to bSb$

$S \to \epsilon$

Ans:- $P: \{ S \to aSa$

$S \to bSb$

$S \to \epsilon \}$

Step 1: Replace $S \to \epsilon$ in place of 'S' & create a set of production rule i.e P'.

$P': \{ S \to aSa, S \to a\epsilon a, S \to bSb,$

$S \to b\epsilon b \}$ i.e

$P': \{ S \to aSa, S \to aa, S \to bSb, S \to bb \}$

∴ Production rule :-

$S \to aSa \mid bSb \mid aa \mid bb$.

2. Eliminate ε-production from the following grammar.

$S \to aS \mid bA$

$A \to aA \mid \epsilon$

Ans:- $P = \{ S \to aS$

$S \to bA$

$A \to aA$

$A \to \epsilon \}$

Step 1: Removal $A \to \epsilon$

$P' = \{ S \to aS, S \to bA, A \to aA, S \to b, A \to a \}$

3. Eliminate ϵ-production & then useless symbol.

$S \to AaB \mid aaB$

$A \to \epsilon$

Ans: $P = \{ S \to AaB, S \to aaB, A \to \epsilon \}$

Step 1: removing $A \to \epsilon$

$P' = \{ S \to AaB, S \to aaB, S \to \epsilon aB \}$

$P' = \{ S \to AaB, S \to aaB, S \to aB \}$

4. Eliminate ϵ-production & then eliminate useless symbol from the given grammar.

$S \to AaB \mid aaB$

$A \to \epsilon$

$B \to bbA \mid \epsilon$

Ans: Eliminating ϵ-production.

$P' = \{ S \to AaB, \overset{S \to}{\epsilon} aB, \overset{S \to}{\epsilon} a\epsilon, \overset{S \to}{a} a\epsilon, B \to bbA$

$S \to aaB, B \to bb\epsilon, \cancel{B \to bb\epsilon}$

$S \to Aa\epsilon \}$

$P' = \{ S \to AaB, S \to aB, S \to Aa, S \to a, S \to aa,$

$B \to bbA, S \to aaB, B \to bb \}$

In this above production rule, variable A is not producing any string from terminal symbol. So A is a useless symbol. After eliminating Production rule will be

$P'' = \{ S \to aaB, S \to aB, S \to a, S \to aa, B \to bb \}$

useless production rule:-

$\{ S \to AaB, S \to Aa, B \to bbA \}$

Elimination of unit production:

Step 1:- Identify unit production in the grammar.

Step 2:- construction of P'

a) Identify all nonunit productions & add it to P'.

b) If there exists a unit production $A \to B$ in the grammar, then apply the following steps until there are no unit production left.

↳ Select a unit production $A \to B$ such that there exist atleast one nonunit production $B \to \alpha$.

↳ Now, for every nonunit production $B \to \alpha$, add production $A \to \alpha$ (i.e $A \to B$ & $B \to \alpha$ so $A \to \alpha$) to the grammar & eliminate $A \to B$ from the grammar.

Eliminate unit production from the given grammar.

$P = \{ S \to 0A \mid 1B \mid C$
$A \to 0S \mid 00$
$B \to 1 \mid A$
$C \to 01 \}$

Sol: **Step 1:** Identify all unit production

$S \to C$
$B \to A$

Step 2: Construction of P'

$P' = \{ S \to 0A \mid 1B$
$A \to 0S \mid 00$
$B \to 1$
$C \to 01 \}$

a) To eliminate $S \to C$, substitute alternative of C i.e what C produces to $S \to C$

Here $C \to 01$ so putting these in $S \to C$

$S \to C$

$\boxed{S \to 01}$ → non unit production.

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

b) To eliminate $B \to A$, substitute alternative of A to $B \to A$

Here $A \to 0S \mid 00$ so putting these in $B \to A$

$\therefore B \to A$

$\boxed{B \to 0S \mid 00}$ → Non unit production

∴ Final non unit production Rule P': { S → 0A | 1B | 0
A → 0S | 00
B → 1 | 0S | 00
C → 0 }

2. Eliminate unit production from the grammar given below:

S → Aa | B
B → A | bb
A → a | bc | B

Ans: Step 1: Identify unit production i.e

S → B
B → A
A → B

Step 2: construction of P'

P': { S → Aa
B → bb
A → a | bc }

a) To eliminate S → B, substitute alternative of B i.e B → α to S →

S → B
S → bb [∵ B → bb]
└ non unit production.

S → Aa | bb — (1)

b) To eliminate B → A, substitute alternative of A i.e A → α to B → A

B → A
B → a | bc [∵ A → a | bc]
└ non unit production.

$B \to bb \mid a \mid bc$ — (2)

c) To eliminate $A \to B$, substitute the alternative of B to $A \to B$

So $A \to B$

$A \to bb$ [$\because B \to bb$ from p']

non unit production.

$\therefore A \to a \mid bc \mid bb$ — (3)

$\therefore$ final non unit production Rule p' =

$\{ S' \to A\,a \mid bb$

$B \to a \mid bc \mid bb$

$A \to a \mid bc \mid bb\}$

3) Eliminate unit production from the grammar given below.

$S \to AB$

$A \to a$

$B \to C \mid b$

$C \to D$

$D \to E$

$E \to a$

Ans: Step 1:- generate all unit production

$B \to C$

$C \to D$

$D \to E$

Step 2:- construction of p'

$p' = \{ S \to AB$

$A \to a$

$B \to b$

$E \to a\}$

a) To eliminate $B \to C$, substitute alternative of C i.e $C \to \alpha$ to $B \to C$

$B \to C|b$

$b \to D|b$ [$\because C \to D$]

$B \to E|b$ [$\because D \to E$]

$B \to a|b$ [$\because E \to a$] — ①

└ non unit production.

b) To eliminate $C \to D$, substitute alternative of D i.e $D \to \alpha$ to $C \to D$

$C \to D$

$C \to E$ [$\because D \to E$]

$C \to a$ [$\because E \to a$] — ②

└ non unit production

c) To eliminate $D \to E$, substitute alternative of E to $D \to E$

$D \to E$

$D \to a$ [$\because E \to a$] — ③

└ non unit production.

Final non unit production rule'll be

P': { $S \to AB$

$A \to a$

$B \to a|b$

$C \to a$

$D \to a$

$E \to a$ }

Normal Form:-

After removing useless symbols, ε-productions & unit productions, sometimes also there's a need for productions rules of CFG to satisfy some specific forms.

So there are 2 types of Normal form in CFG

(i) Chomsky Normal Form (CNF)

(ii) Greibach Normal Form (GNF)

CNF:-

Formally a CFG is said to be in CNF if all the productions are in the form

$A \to BC$ or $A \to a$

where $A, B, C \to$ Non-terminal

$a \to$ Terminal

i.e CNF puts restrictions on number of symbols on RHS of the production.

In otherwords strings on the RHS of the production should not consist more than 2 variables or one single terminal (i.e more than 2 symbols). eg: $S \to AA$ (NOT in CNF)

GNF :-

Formally a CFG is said to be in GNF, if all productions are in the form of

$A \to aX$

where a is a terminal i.e $a \in T$

or

$\forall X \in V^*$

$A \to a$ [$\because X \in V^*$].

i.e GNF doesn't put restriction on the length of RHS of a production, but on the position in which terminals & variables appear.

eg: $S \to aA$

$B \to b \in$

Above grammar is in GNF.

Thm: For every CFG there is an equivalent Grammar G in CNF. / Reduction from CFG to CNF.

Proof: Let $G_1 = (V, T, P, S)$ be the CFG generating a language not containing $\in$. Construct $G_2 = (V'', T, P'', S)$ which will be in CNF as follows.

Step 1: Eliminate $\in$-production & unit production.

Step 2: Eliminate terminals on RHS of productions as follows:

1. All the productions in P of the form $A \to Bc$ & $A \to a$ are included in P' & all variable of such production are included in V'.

2. Then consider a production of the form

$A \to w_1 w_2 \cdots w_n$, $n \neq 2$ i.e $n \geq 3$

If w_i is a terminal 'a', introduce a new variable T_a & Replace the terminal w_i by T_a.

These new productions are added to P' & variable to V'.

Step 3:
Restricting the no. of variables on RH as follows:

↳ All productions that are already in acceptable form of P' are added to P'' & all corresponding variables are included in V''.

↳ Consider a production $A \to A_1 A_2 \cdots A_m$, $m \geq 3$, then we introduce new productions

$A \to A_1 K_1$ (NT NT)

$K_1 \to A_2 K_2$

$K_2 \to A_3 K_3$

⋮

$K_{n-1} \to A_n A_n$

$K_{n-2} \to A_{n-1} K_{n-1}$

Then we get CNF.

(proved)

Q1 Find the equivalent grammar in CNF for the following CFG.

$S \to aAD$

$A \to aB \mid bAB$

$B \to b$

$D \to d$

Ans: As here there is no ε-production & unit production. So no need to eliminate.

Step 2: In CNF, on RHS there must be 2 nonterminals & one terminal; so we've to eliminate multiple terminals of a production.

$P' = \{B \to b, D \to d\}$

$V' = \{B, D\}$

considering

$S \to aAD$

$S \to T_a AD$ (NOT CNF)

$T_a \to a$ (CNF)

$S \to T_a K_1$ (CNF)

$K_1 \to AD$ (CNF)

considering $A \to aB$

$A \to T_a B$ ($\because T_a \to a$) → CNF

considering $A \to bAB$

$A \to T_b AB$ (NOT CNF)

$T_b \to b$ (CNF)

~~$A \to T_b T_a A$ ($\because T_a \to a$)~~

$A \to T_b K_2$ [CNF]

$K_2 \to AB$ (CNF)

Step 3:-

$P'' = \{B \to b, D \to d, T_a \to a, S \to T_a K_1, K_1 \to AD, A \to T_a B, T_b \to b, A \to T_b K_2, K_2 \to AB\}$

$V'' = \{B, D, T_a, S, K_1, A, T_b, K_2\}$

convert the following grammar to CNF

$S \to a \mid b \mid aSS$

Ans: Step 1: As here there's no $\in$ & unit production. so no need to eliminate it.

Step 2: $P' = \{S \to a, S \to b\}$

$V' = \{S\}$

considering

$S \to aSS$

$S \to T_a SS$ (Not in CNF)

$T_a \to a$ (in CNF)

$S \to T_a K_1$ (CNF)

$K_1 \to SS$ (CNF)

Step 3: $P'' = \{S \to a, S \to b, T_a \to a, S \to T_a K_1, K_1 \to SS\}$

$V'' = \{S, T_a, K_1\}$

GNF:- If a production is in the form $A \to aX$ & $X \in V^*$ or $A \to a$

eg: $S \to AB$

$A \to aA \mid bB \mid b$

$B \to b$

is not in GNF.

eg: $S \to aAB \mid bBB \mid bB$

$A \to aA \mid bB \mid b$

$b \to b$

is in GNF.

All the productions should satisfy the condition of unique pairs, then only GNF.

How to get GNF :

To convert a given grammar into GNF, following 2 lemma's are considered.

Lemma 1 :

Let $G = V, T, P, S$ be a given CFG. If there's a production $A \to B\alpha$ &

$$B \to \beta_1 | \beta_2 | \cdots \beta_n$$

Then $A \to \beta_1\alpha | \beta_2\alpha | \cdots \beta_n\alpha$

Lemma 2 :-

Let $G = (V, T, P, S)$ be a given CFG. If there's a production

$$A \to A\alpha_1 | A\alpha_2 | \cdots A\alpha_n | \beta_1 | \beta_2 | \cdots \beta_n$$

such that β_i don't start with A, then GNF will be

$$A \to \beta_1 | \beta_2 | \cdots \beta_n$$
$$A \to \beta_1 Z | \beta_2 Z | \cdots \beta_n Z$$
$$Z \to \alpha_1 | \alpha_2 | \cdots \alpha_n$$
$$Z \to \alpha_1 Z | \alpha_2 Z | \cdots | \alpha_n Z$$

eg :- Consider

$$A \to A1 | 0B | 2$$

convert it into GNF.

Ans :- $A \to A1 | 0B | 2$ $\quad [A \to A\alpha_1 | \beta_1 | \beta_2]$

$\therefore \alpha_1 = 1$, $\beta_1 = 0B$ & $\beta_2 = 2$ then

$A \to \beta_1 | \beta_2 \Rightarrow 0B | 2$

$A \to \beta_1 Z | \beta_2 Z \Rightarrow 0BZ | 2Z$

$Z \to \alpha_1 \Rightarrow Z \to 1$

$Z \to \alpha_1 Z \Rightarrow Z \to 1Z$

Reduction of CFG to GNF :-

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

Step 1:
Remove useless variable.

- eliminate ϵ-production
- Remove unit production.
- convert grammar to CNF.
- Then we convert this grammar into an equivalent grammar in GNF applying Lemma 1 / Lemma 2

eg: Convert the following grammar to GNF

$S \to CA$

$A \to a$

$C \to aB \mid b$

Ans: Here in this grammar A & C are in GNF as $A \to a$,

$C \to aB \mid b$

But S is not in GNF as $S \to CA$

so applying Lemma 1, $S \to C A$ ($A \to B\alpha$)

$A \to a$

$S \to aBA \mid bA$

($B \to B_1 \mid B_2 \Rightarrow A \to \beta_1 \alpha \mid \beta_2 \alpha$)

$\therefore$ GNF will be

$S \to aBA \mid bA$

$A \to a$

$C \to aB \mid b$

2. Convert the following CFG to CNF

$S \to AB$	$S \to aBA$
$A \to aA \mid bB \mid b$	$A \to aA \mid \epsilon$
$B \to b$	$B \to bB \mid \epsilon$

Pushdown Automata :-

PDA is an abstract device that recognize CFL.

PDA accepts CFL i.e nonregular lang. When a language is not regular if its infinite.
eg: $a^n b^n / n \geq 0$, a^p : PU any prime no. else that

In FA we don't have memory to store ones values. So that they are nonregular lang. Though we've memory is nearly & more m/c in FA with memory i.e memory is used in this m/c but there no concept of final state. So that a string can't be accepted.

Basic Reason :- During state transition, o/p is produced & that can't be used in future. Bcz o/p is called in each step. Because no component (memory) to store the o/p.

So after that PDA came into feature with same features of FA but with additional memory i.e stack which provides an unlimited amount of memory

If $L = a^n b^n / n \geq 0$ can't be accepted by FA. Because it has to remember no. of a's & b's in the string. For this a FA will required an infinite no. of states. To overcome this problem, an addition of auxiliary memory in the form of stack is included/introduced to store the values.

→ In stack the stored elements are processed in LIFO fashion.

To accept strings of the form $a^n b^n$ given by any L, a's are pushed onto stack (push operation) & when b's encounter, then pop operation (deletion) is being carried out i.e topmost 'a's from the stack will be deleted. Thus the matching of no. of a's & b's is accomplished.

This type of additional arrangement in FA is called pushdown Automata.

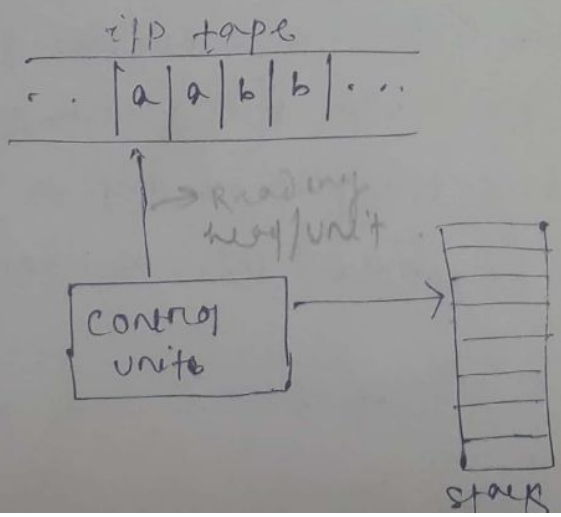
Def^n:- a) Pushdown Automata is modified form of FA with memory (stack). Unlike FA, PDA changes from state to state, reading i/p symbol & transitions update the stack either by pushing the symbol or popping them.

b) PDA is a way to represent the language called CFL.

PDA = FA + 1 stack

Components of PDA:-

as PDA: FA + 1 stack so dig will be



PDA includes 4 components.

(i) Input Tape
(ii) Read Unit
(iii) Control Unit
(iv) Stack

Input Tape :-

It's an infinitely long tape on which i/ps are written.

→ The tape is divided into sequence of cells. Each cell begins from the left end & extends to the right without an end.

→ Each cell of the tape holds one i/p letter or blank or ε.

Read Unit :-

Read unit of PDA reads words from the cells of the i/p tape, beginning with the first letter in the leftmost cell & then moves to the right. But it can't go to back.

Control Unit :-

It governs / performs the operation of PDA by performing a sequence of transition between internal states available to it.

Stack :-

A PDA has an infinitely tall pushdown stack which has LIFO discipline.

- Stack always starts with stack empty.

It has 3 operations.

a) push - Add i/p alphabet to stack.
b) pop - Remove topmost i/p alphabet from the stack.

c) nop - does nothing to stack.

Formal Definition of PDA :-

A PDA is a 7-tuple m/c.

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

where Q : Finite set of states

Σ : " set of i/p alphabets

Γ : finite set of stack symbols.

δ : Transition function.

$$Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to Q \times \Gamma^*$$

q_0 : Initial state

Z_0 : Top of stack

F : set of final state.

NOTE :-

→ A PDA may make transition without reading any i/p symbol. But it can make a transition when the stack is empty.

→ A FA is in some state & on reading an i/p symbol goes to a newstate.

whereas a PDA is in some state & on reading both an i/p symbol & the topmost symbol of the stack moves to a newstate.

Description of PDA :-

There are different ways to describe a pushdown automata.

(I) Transition diagram

(II) Transition Table

Transition Diagram:

It's a directed graph where states are represented by vertices & the labelling of edges can be represented in different forms.

form 1:

(i/p symbol, top input symbol of stack)
operation on stack.

(i/p symbol, top i/p symbol of stack
operation on stack)

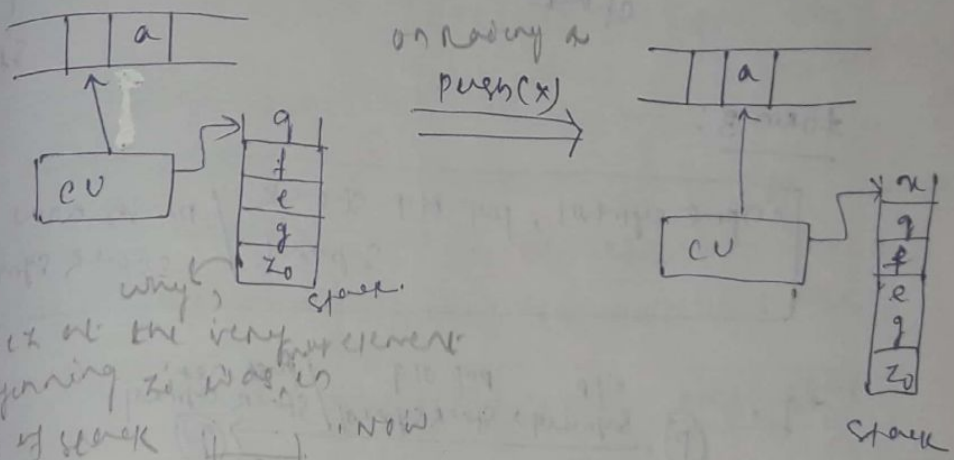
(p) → (q)

a, q
push(x)

(p) → (q)

current

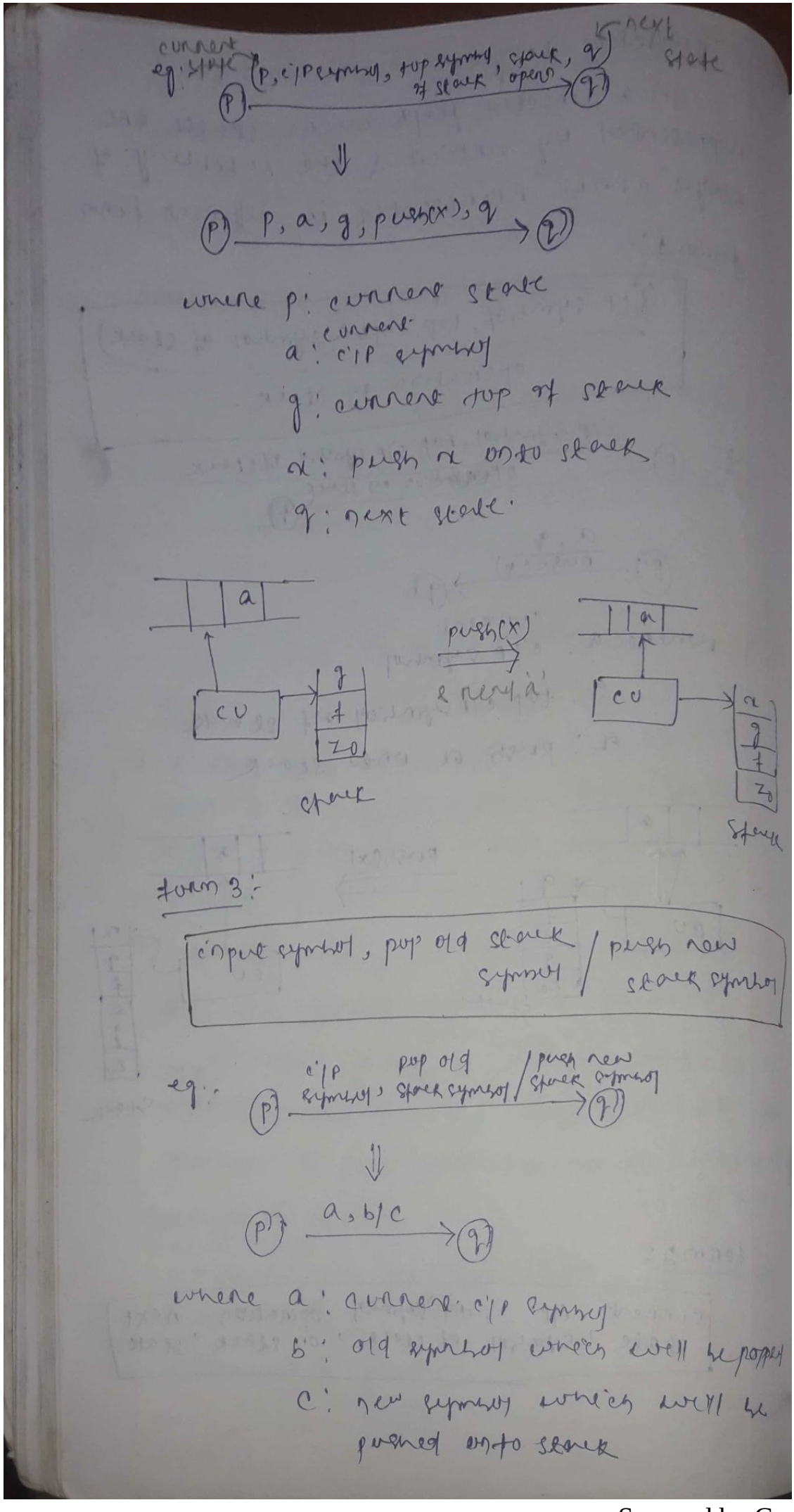
where a: i/p symbol
q: top i/p symbol of stack
x: push a onto stack

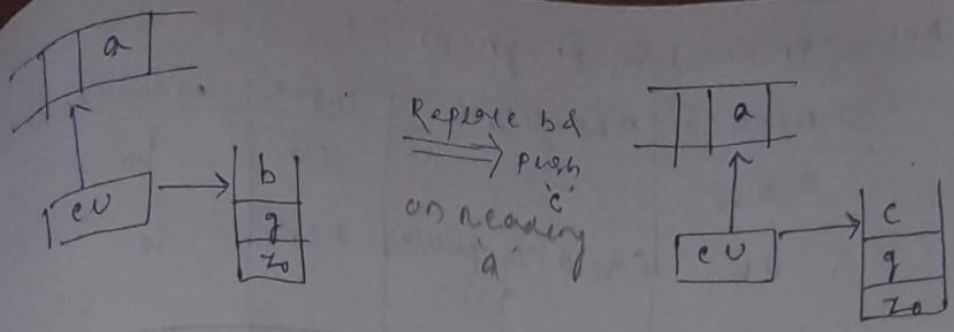


why,
Bcz at the very beginning Z_0 was in top of stack
NOW
Z_0
current top element of stack is q.

form 2:

current i/p, top symbol operation, next state, symbol of stack, on stack, state





Better use format 1 in all cases.

The transition diagram of PDA incorporates the following operations:

If a PDA is in state p, the i/p symbol is any letter of w (i/p symbol can be ∈ also) & 's' be the top element of the stack, then PDA

a) executes the stack operation (push/pop/nop)
b) moves to state q & next state
c) of the i/p symbol ≠ ∈, then it goes to right (i.e the next cell of the tape)

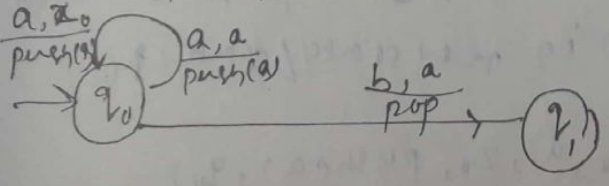
Transition Table:-

Description of operation of a PDA for a given i/p string can be represented in a tabular format called transition table.

Unread i/p	Transition	Stack	New state

[∴ Transition Table format]

Q1 Construct a transition table for the following PDA with i/p aab.



Ans:

Transition [redacted]:

Unread i/p	Transition	Stack (initially top of stack Z_0)	New state
aab	—	Z_0	q_0 (initial state)
ab	$(q_0, a, Z_0, push(a), q_0)$	$aZ_0 \to \frac{a}{Z_0}$	q_0
b	$(q_0, a, a, push(a), q_0)$	$aaZ_0 \to \frac{a}{a}\frac{a}{Z_0}$	q_0
ε	(q_0, b, a, pop, q_1)	$aZ_0 \to \frac{\not{a}}{a}\frac{a}{Z_0}$	q_1

Read upto ε.

Transition function of PDA:

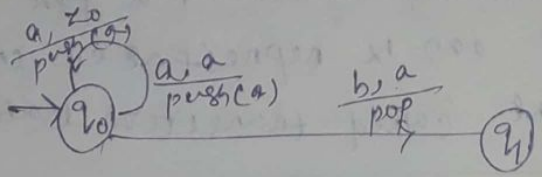
Transition function of PDA can be represented in different forms.

Form 1:

$$\delta(\text{current state}, \text{i/p symbol}, \text{top of stack}) = (\text{new state}, \text{new top of stack})$$

Form 2:

$$(\text{current state}, \text{current i/p symbol}, \text{current top of stack}, \text{operation on stack}, \text{new state})$$



Transition function:

Form 1: $\delta(q_0, a, Z_0) = (q_0, a)$

Here means for current state q_0, current i/p symbol 'a', & the current top of stack is Z_0, then push a onto stack & remain in new state / state q_0

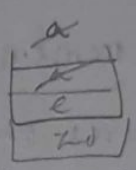
Form 2: $(q_0, a, Z_0, push(a), q_0)$

This means for the current state q_0, current i/p symbol 'a', & the current top of stack is z_0, then push 'a' onto stack & remains in state q_0.

form 2: $(q_0, a, a, push(a), q_0)$

form 3: $\delta(q_0, a, a) = q_0, a$

form 1: $\delta(q_1, b, a) = q_1, \epsilon)$



ε: It's used to explicitly pop 'a' (current top of stack)

form 4: (q_0, b, a, pop, q_1)

Language Accepted by PDA :-

There are 2 ways to define the language acceptance by PDA.

(i) Accepted by final state.

$M: Q, \Sigma, \Gamma, \delta, q_0, Z_0, F$

(ii) Accepted by empty stack → final state will not be there

$M: Q, \Sigma, \Gamma, \delta, q_0, Z_0$

Language Accepted by final state :-

Even though stack is not empty, after scanning the i/p string, if the state reaches to the final state, then it's accepted otherwise not.

$\{w \mid (q_0, w, Z_0) \vdash^* (p, \epsilon, \gamma)$ for some p in F and $\gamma \in \Gamma^*\}$

11) Language accepted by empty stack:

After scanning the entire string, if the stack is empty then i/p is accepted otherwise not.

$\{w \mid (q_0, w, z_0) \vdash^* (p, \epsilon, \epsilon)$ for some $p \in Q$

Instantaneous Description (ID) of PDA:

(Represented with $\vdash$)

ID remember state & stack content at

or

ID gives the current situation of PDA.

ID of a PDA is defined as a triple (q, w, s) where

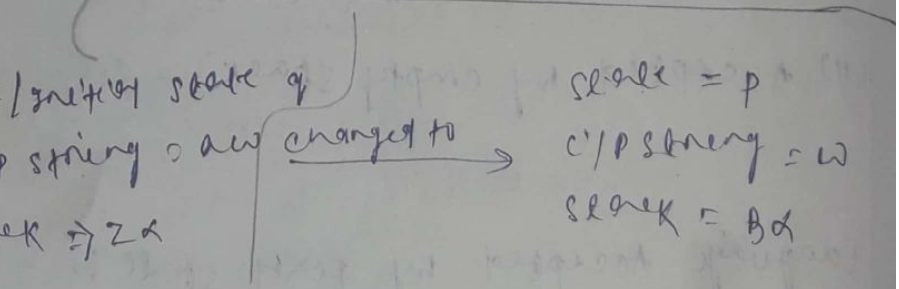
q = current state

w = remaining i/ps / string.

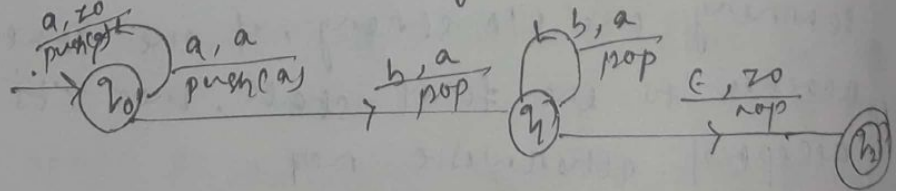
s = current stack contents

Formally let $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ be a PDA then ID of a PDA is

$\delta(q, aw, z\alpha) \vdash_M (p, w, \beta\alpha)$



Q. Consider the following transition diagram of PDA



Consider the i/p string $w = aabb$ & z_0 current stack top. Find out transition table, transition function & ID. Also show whether given string is accepted or not.

Transition Table:-

Unread i/p	Transition	stack	New state
aabb	q_0, —	Z_0	q_0
abb	$(q_0, a, Z_0, \text{push}(a), q_0)$	aZ_0	q_0
bb	$(q_0, a, a, \text{push}(a), q_0)$	aaZ_0	q_0
b	$(q_0, b, a, \text{pop}, q_1)$	aZ_0	q_1
ϵ	$(q_1, b, a, \text{pop}, q_1)$	Z_0	q_1
—	$(q_1, \epsilon, Z_0, \text{nop}, q_2)$	Z_0	q_2

(Acceptance is stack q_2 Z_0)

Since after scanning the i/p symbol/ string, it reaches to q_2 which is a final state & stack is empty, (i/p is ϵ). so the given language is accepted by PDA.

Transition Function:-

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

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

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

$\delta(q_1, b, a) = (q_1, Z_0)$

$\delta(q_1, \epsilon, Z_0) = (q_2, Z_0)$

ID:-

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

$\vdash (q_0, bb, aaZ_0)$

$\vdash (q_1, b, aZ_0)$

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

$\vdash (q_2, z_0)$

$\therefore (q_0, aabb, z_0) \overset{*}{\underset{M}{\vdash}} (q_2, z_0)$

where $q_2 \in F$.

Design of PDA:-

There are certain strategies to design PDA:

a) → Understand the language properties for which PDA has to be designed.

b) → Determine the stack & alphabet set required.

c) Identify the initial, accepting & dead state of PDA.

d) Decide on the stack symbols required.

e) Determine the initial stack symbol from the stack symbol set.

f) For each state, decide on the transition to be made for each character of the i/p string.

g) For each state transition, decide on the stack operation to be performed.

h) Obtain the transition diagram & table for PDA.

i) Test the PDA obtained by taking a sample string.

Ex: construct a PDA to accept the lang $L=\{a^n b^n / n \geq 0\}$

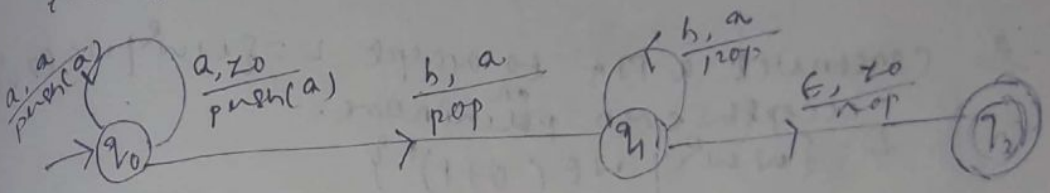
Sol:- $L: \{a^n b^n / n \geq 0\}$

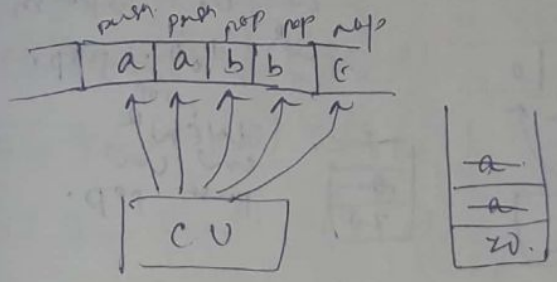
a^n b^n ϵ
push pop nop

Let initially top of stack is z_0.
with a^n, push operation will be performed & will be in same state. Let i.e in q_0

on b^n, state will be changed from q_0 to q_1 & pop operation will be carried out. But q_1 is not final state

Atlast with ϵ, again state will be changed i.e q_1 to q_2 (which is final state) with nop.

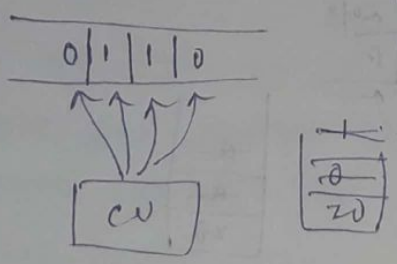




Transition Table (i/p: aaabbb)

3. Construct a PDA to accept $L = \{ww^R \mid w \in (0+1)\}$ or accepts even palindrome.

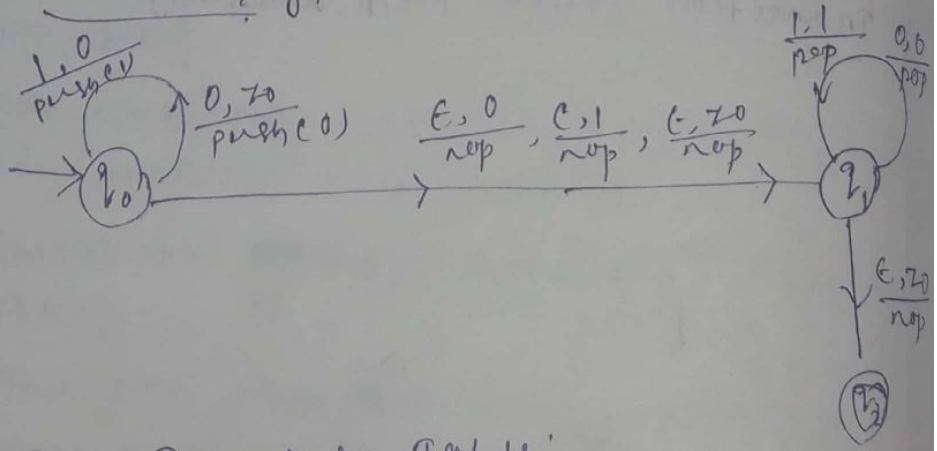
$L = \{ww^R \mid w \in (0+1)^*\}$



$w = 0, 1 \Rightarrow$ push
$w^R = 1, 0 \Rightarrow$ pop.

$\lambda \in w w^R$ (nop)
CU CU
push pop.

(2) Transition Dig:-



(3) Transition Table:-

Consider the i/p string $w = 0110$

Unread i/p	Transition	Stack	Next state
0110	-	Z0	q_0
110	$(q_0, 0, Z_0, push(0), q_0)$	0Z0	q_0

10	$(q_0, 1, 0, \text{push}(1), q_0)$	$10Z_0$	q_0
1	$q_0, \epsilon, 1, \text{nop}, q_1$	$10Z_0$	q_1
0	$q_1, 1, 1, \text{pop}, q_1$	$0Z_0$	q_1
ϵ	$q_1, 0, 0, \text{pop}, q_1$	Z_0	q_1
-	$q_1, \epsilon, Z_0, \text{nop}, q_2$	Z_0	q_2

ID :- $(q_0, 0110, Z_0) \vdash (q_0, 110, 0Z_0)$

(1) $\vdash (q_0, 10, 10Z_0)$

$\vdash (q_0, 0, 0Z_0)$

$\vdash (q_1, \epsilon, Z_0)$

$\vdash (q_2, Z_0)$

(1) Transition function :-

$\delta(q_0, 0, Z_0) = (q_0, 0)$

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

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

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

$\delta(q_1, 0, 0) = (q_1, Z_0)$

$\delta(q_1, \epsilon, Z_0) = (q_2, Z_0)$.

$\because$ PDA : $M = \{Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\} = \{\{q_0, q_1, q_2\}, \{0, 1, \epsilon\}, \{0, Z_0\}, \delta, q_0, Z_0, q_2\}$

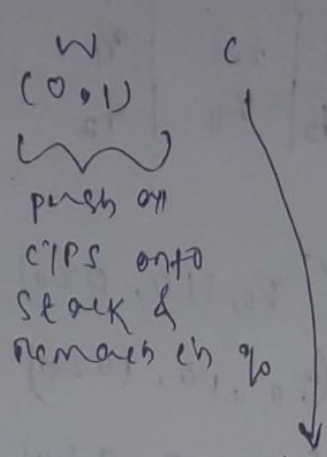
$\therefore$ By scanning the entire string we reach to q_2 which is a final state. So the string 0110 is accepted by this PDA.

3. Construct a PDA to accept $L = \{wcw^R \mid w \in (0+1)^*\}$

or

odd palindrome.

$L = \{wcw^R \mid w \in (0+1)^*\}$



once 'c' is encountered, change from q_0 to q_1 with nop

possible

(1) Transition function :-

i) Read each symbol of the i/p string & push onto stack before c is encountered.

a) $q_0, \in, Z_0, nop, q_0$

b) $q_0, 0, Z_0, push(0), q_0$

c) $q_0, 1, Z_0, push(1), q_0$

d) $q_0, 0, (0), q_0$ (push)

e) $q_0, 0, 1, push(0), q_0$

f) $q_0, 1, 0, push(0), q_0$

g) $q_0, 1, 1, push(1), q_0$

ii) Once c is encountered change to q_1 with nop

h) $q_0, c, 0, nop, q_1$

i) $q_0, c, 1, nop, q_1$

j) q_0, c, Z_0, nop, q_1

iii) Read the next i/p symbol from second word & pop from the stack, if there's a match.

k) $q_1, 0, 0, pop, q_1$

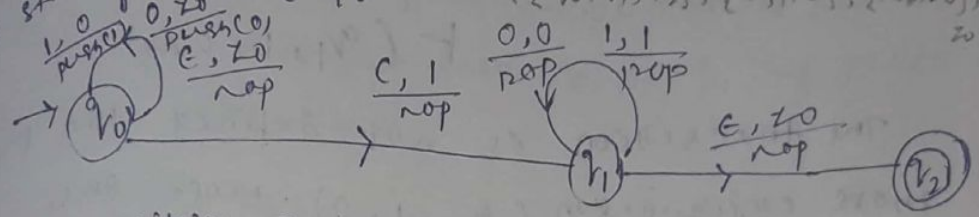
l) $q_1, 1, 1, pop, q_1$

iv) Once the stack is empty or i/p is ∈, change to q_2

m) $q_1, \epsilon, z_0, nop, q_2$

PDA: $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F) = \{\{q_0, q_1, q_2\}, \{0, 1, \epsilon\}, \{0, 1, z_0\}, \delta, q_0, z_0, q_2\}$

string: 01 C 10



Transition table:-

Unread i/p	Transition	stack	Newstate
01C10	q_0 . —	z_0	q_0
1C10	$(q_0, 0, z_0, push(0), q_0)$	$0z_0$	q_0
C10	$(q_0, 1, 0, push(1), q_0)$	$10z_0$	q_0
10	$q_0, C, 1, nop, q_1$	$0z_0$	q_1
0	$q_1, 1, 1, pop, q_1$	z_0	q_1
ϵ	$q_1, 0, 0, pop, q_1$	z_0	q_1
-	$q_1, \epsilon, z_0, nop, q_2$	z_0	q_2

ID is:

$(q_0, 01C10, z_0) \vdash (q_0, 1C10, 0z_0)$

$\vdash (q_0, C10, 10z_0)$

$\vdash (q_1, 10, 10z_0)$

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

$\vdash (q_1, \epsilon, Z_0)$

$\vdash (q_2, Z_0)$

& stack empty is ∴ Hence w is

Since q_2 is final state ∴ Hence w is accepted by PDA.

To show 01c11

$(q_0, 01c11, Z_0) \vdash (q_0, 1c11, 0Z_0)$

$\vdash (q_0, c11, 10Z_0)$

$\vdash (q_1, 11, 10Z_0)$

$\vdash (q_1, 1, 0Z_0)$

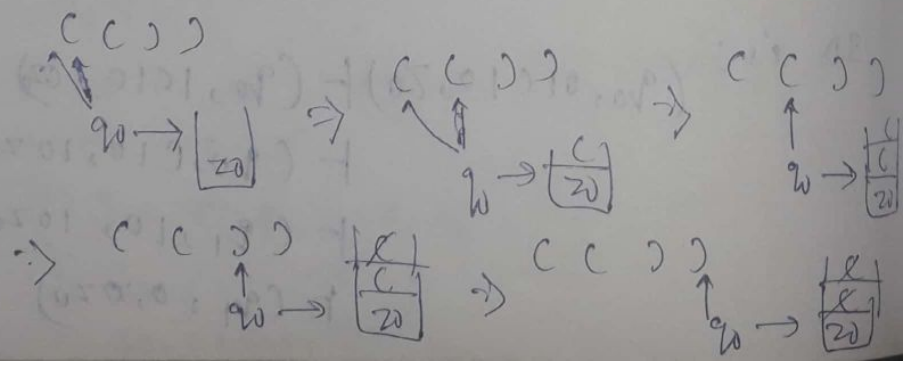
The transition is not defined for the above configuration $(q_1, 1, 0)$. Hence the string 'w' is rejected by PDA.

4. Construct a PDA to accept the language of nested balanced parantheses.

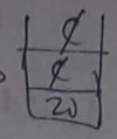
Ans:- eg:- ((()))

- push all symbols of type '(' into the stack.
- pop all symbols of type ')' when symbols of type '(' is encountered.

Remove all symbols from the stack until the stack becomes empty.



(( )) ϵ



Transition function :-

i) To read each symbol of the i/p string of type '(', push (onto stack until ')' is encountered.

a. $(q_0, (, Z_0, push(), q_0)$

b. $(q_0, (, (, push(), q_0)$

ii) Once symbol ')' is encountered, state will be changed from q_0 to q_1 by performing pop operation.

c. $(q_0,), (, pop, q_1)$

d. $(q_1,), (, pop, q_1)$

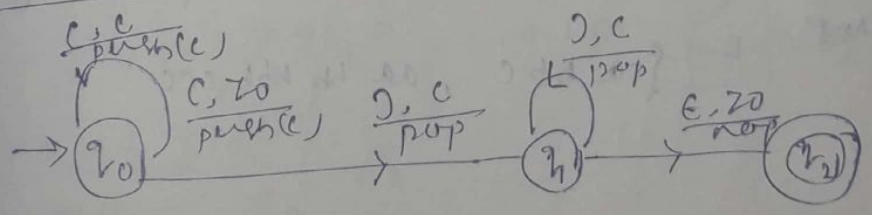
iii) If the i/p is ϵ, then change state q_1 to final state q_2

e. $(q_1, \epsilon, Z_0, nop, q_2)$

PDA = $\{Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\}$

$= \{\{q_0, q_1, q_2\}, \{(,)\}, \{(, Z_0\}, \delta, q_0, Z_0, q_2\}$

Transition diagram :-



Transition table :- w : (( ))

Unread i/p	Transition	stack	new state
(( ))	-	z_0	q_0
( ))	$(q_0, (, z_0, \text{push}(()), q_0)$	(z_0	q_0
))	$(q_0, (, (, \text{push}(()), q_0)$	((z_0	q_0
)	$q_0,), (, \text{pop}, q_1$	(z_0	q_1
ϵ	$q_1,), (, \text{pop}, q_1$	z_0	q_1
—	$q_1, \epsilon, z_0, \text{nop}, q_2$	z_0	q_2

To show w : (( )) accepted by given PDA,

ID :- $(q_0, (( )), z_0) \vdash (q_0, ( )), (z_0)$

$\vdash (q_0,)), ((z_0)$

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

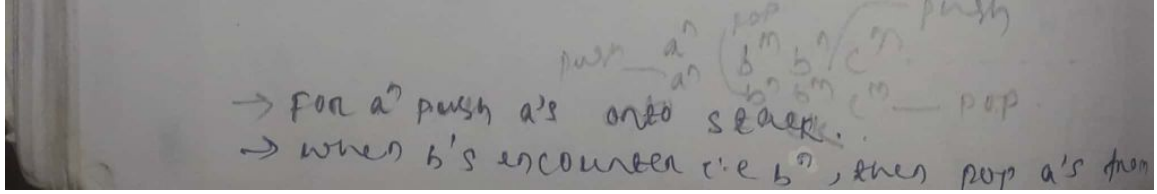
$\vdash (q_1, \epsilon, z_0)$

$\vdash (q_2, z_0)$

Since the final state is q_2 & the stack is empty i.e it is reach to z_0. So, w is accepted by PDA.

⇒ Construct a PDA to accept $L = \{a^n b^{m+n} c^m \mid m, n$

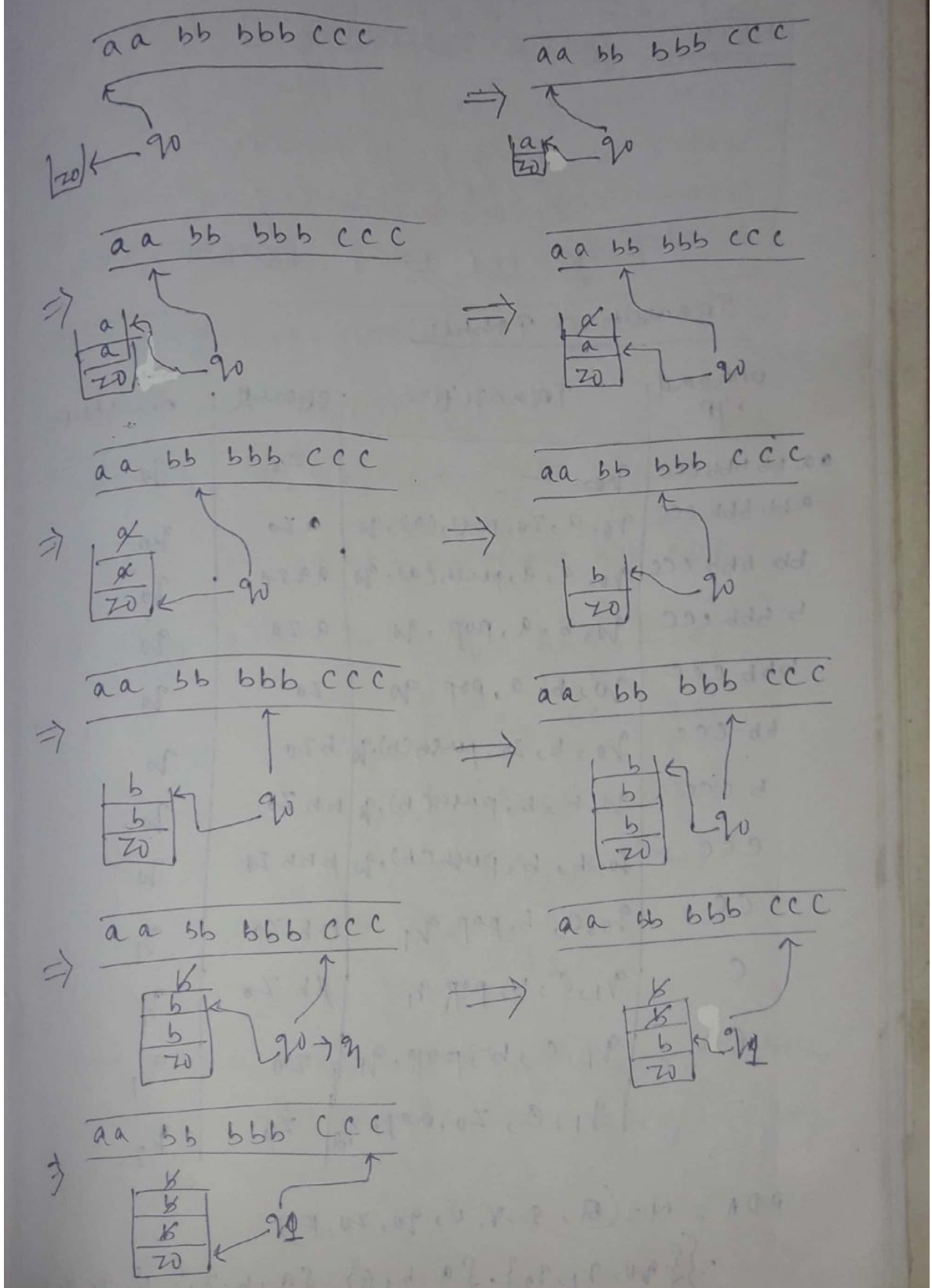
Ans :- $L = \{ abbc , aabbbccc \}$

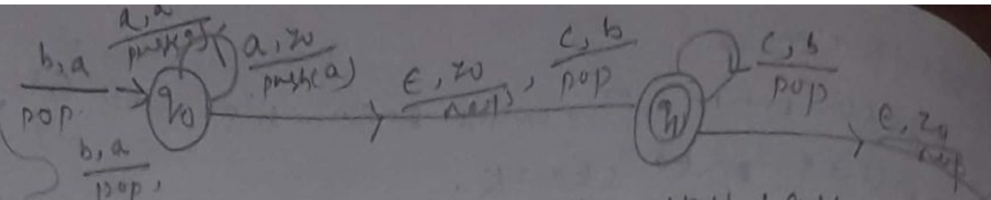


→ For a^n push a's onto stack.
→ When b's encounter i.e b^n, then pop a's from

stack upto z_0.

- Again when b^m comes then b's will be pushed onto stack.
- After that when c's will be encountered i.e c^m, then pop operation will be carried out i.e b's (ab^m) will be popped.





why here $\frac{b, Z_0}{push(b)}$, $\frac{b,b}{push(b)}$, $\frac{b,b}{push(b)}$. comb^n of both push & pop! bcz required b when comes pop will be done. But after b there's only one symbol Z0 in stack so we've to push b's.

In this case c can't read at last bcz $L = a^n b^{m+n} c^m$ i.e. aabbc... last symbol is 'c' so c can't ...

NOTE: It's not compulsory that in a single state push oper^n will be done & on popping state'll be changed. It depends upon the situation. So transition diag is done by analysing the lang. properly.

Transition Table :-

unread i/p	Transition	Stack	new state
aa bb bbb ccc	q_0 —	Z_0	q_0
a bb bbb ccc	$q_0, a, Z_0, push(a), q_0$	aZ_0	q_0
bb bbb ccc	$q_0, a, a, push(a), q_0$	aaZ_0	q_0
b bbb ccc	q_0, b, a, pop, q_0	aZ_0	q_0
bbb ccc	q_0, b, a, pop, q_0	Z_0	q_0
bb ccc	$q_0, b, Z_0, push(b), q_0$	bZ_0	q_0
b ccc	$q_0, b, b, push(b), q_0$	bbZ_0	q_0
ccc	$q_0, b, b, push(b), q_0$	$bbbZ_0$	q_0
cc	q_0, c, b, pop, q_1	bbZ_0	q_1
c	q_1, c, b, pop, q_1	bZ_0	q_1
ε	q_1, c, b, pop, q_1	Z_0	q_1
—	$q_1, ε, Z_0, nop, q_2$	Z_0	q_2

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

$= \{\{q_0, q_1, q_2\}, \{a, b, c\}, \{a, b, Z_0\}, \delta, q_0, Z_0, q_2\}$

Transition Functions:- $(q_0, \epsilon, z_0, \text{nop}, q_1)$

i) $\delta(q_0, a, z_0, \text{push}(a), q_0)$

ii) $(q_0, a, a, \text{push}(a), q_0)$

iii) $(q_0, b, a, \text{pop}, q_0)$

iv) $(q_0, b, a, \text{pop}, q_0)$

v) $(q_0, b, z_0, \text{push}(b), q_0)$

vi) $(q_0, b, b, \text{push}(b), q_0)$

vii) $(q_0, b, b, \text{push}(b), q_0)$

viii) $(q_0, \epsilon, b, \text{pop}, q_1)$

ix) $(q_1, c, b, \text{pop}, q_1)$

x) $(q_1, c, b, \text{pop}, q_1)$

xi) $(q_1, \epsilon, z_0, \text{nop}, q_2)$

To show w = abbc accepted by PDA.

ID:- $(q_0, abbc, z_0) \vdash (q_0, bbc, a z_0)$

$\vdash (q_0, bc, z_0)$

$\vdash (q_0, c, b z_0)$

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

$\vdash (q_2, z_0)$

Since the final state is q_2 & stack is empty with z_0. So w is accepted by given PDA.

6) Construct a PDA to accept $L = \{ w \in \{a,b\}^* \mid n_a : n_b \}$ (no. of a : no. of b)

Ans:- $L = \{ w \in \{a,b\}^* \mid n_a : n_b \}$.

$\therefore L = \{ \epsilon, ab, aabb, abab, a \}$ with ϵ

Transition Functions:-

i) $q_0, \epsilon, z_0, nop, q_1$ with ϵ state will be changed & reading all symbols of i/p str by performing push & pop op

ii) Read each symbol of i/p string & push onto stack — we remarked same : Bcoz input ϵ not to be read & it'll remain to final state.

a) $(q_0, a, z_0, push(a), q_0)$

b) $(q_0, b, z_0, push(b), q_0)$

iii) push onto stack the i/p symbol, if it same as top of the stack.

c) $(q_0, a, a, push(a), q_0)$

d. $(q_0, b, b, push(b), q_0)$

iv) If the i/p symbol is not same as that of top of stack, then

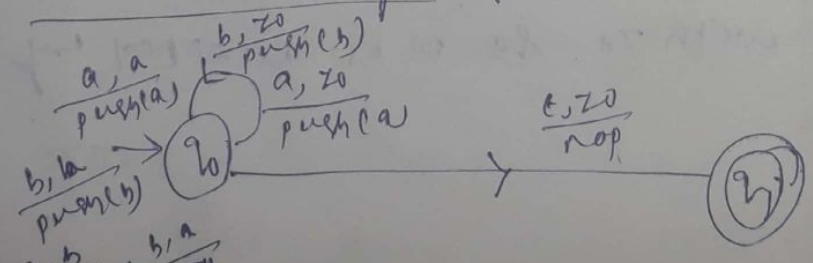
e) (q_0, a, b, pop, q_0)

f) (q_0, b, a, pop, q_0)

$\therefore PDA = \{M = (Q, \Sigma, \Psi, \delta, q_0, z_0, F)$

$= \{\{q_0, q_1\}, \{a, b, \epsilon\}, \{a, b, z_0\}, \delta, q_0, z_0$

Transition diagram:-



Transition Table:- w : ababba

Unread i/p	Transition	stack	new state
ababba	–	z_0	q_0
babba	$(q_0, a, z_0, push(a), q_0)$	aq_0	q_0
abba	q_0, b, a, pop, q_0	z_0	q_0
bba	$q_0, a, z_0, push(a)$ q_0	az_0	q_0
ba	q_0, b, a, pop, q_0	z_0	q_0
a	$q_0, b, z_0, push(b)$, q_0	bz_0	q_0
e	q_0, a, b, pop, q_0	z_0	q_0
–	q_0, e, z_0, nop, q_1	z_0	q_1

To show ababba accepted by PDA :-

ID :- $(q_0, ababba, z_0) \vdash (q_0, babba, az_0)$

$\vdash (q_0, abba, z_0)$

$\vdash (q_0, bba, az_0)$

$\vdash (q_0, ba, z_0)$

$\vdash (q_0, a, bz_0)$

$\vdash (q_0, e, z_0)$

$\vdash (q_1, z_0)$

∴ As q_1 is final state & stack is empty with z_0, so w is accepted by PDA.

Ex: Construct a PDA to accept $L = \{0^{2n} \mid n \geq 0\}$

$L = \{\epsilon, \underbrace{00}_{0^{2(n=1)}}, \underbrace{0000}_{0^{4(n=2)}} \ldots\}$

i) For first 0^n push 0's onto stack.

ii) When 2nd 0^n is encountered, then pop 0's (of the 0^n) from stack. & All this push & pop operations will be done in same state.

iii) When ϵ will be encountered, then state (q_0) will be changed to final state (q_1)

Transition functions:

i) $q_0, \epsilon, z_0, nop, q_1$

ii) a) $q_0, 0, z_0, push(0), q_0$

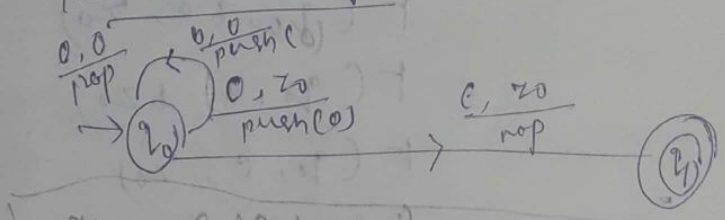
b) $q_0, 0, 0, push(0), q_0$

iii) c) $q_0, 0, 0, pop, q_0$

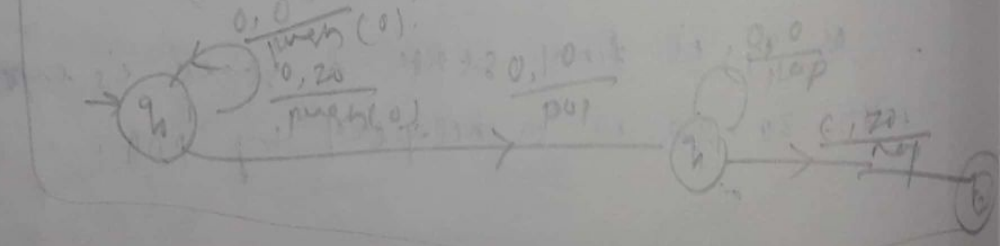
PDA: $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

$= (\{q_0, q_1\}, \{0, \epsilon\}, \{0, z_0\}, \delta, q_0, z_0, \{q_1\})$

Transition Diagram:



q1 $L = \{0^{2n} \mid n > 0\}$



Transition table :- w = 0000

unread i/p	Transition	stack	newstate
0000	-	Z_0	q_0
000	$q_0, 0, Z_0, push(0), q_0$	$0Z_0$	q_0
00	$q_0, 0, 0, push(0), q_0$	$00Z_0$	q_0
0	$q_0, 0, 0, pop, q_0$	$0Z_0$	q_0
ϵ	$q_0, 0, 0, pop, q_0$	Z_0	q_0
-	$q_0, \epsilon, Z_0, nop, q_1$	Z_0	q_1

To show 0000 accepted by PDA :-

ID :- $(q_0, 0000, Z_0) \vdash (q_0, 000, 0Z_0)$
$\vdash (q_0, 00, 00Z_0)$
$\vdash (q_0, 0, 0Z_0)$
$\vdash (q_0, \epsilon, Z_0)$
$\vdash (q_1, Z_0)$

As q_1 is final state & stack is empty with Z_0. So w is accepted by PDA.

Types of PDA :-

PDA is of 2 types

(i) Deterministic PDA (DPDA)

(ii) Nondeterministic PDA (NPDA)

DPDA :-

A PDA is deterministic, if each i/p string can only be processed by the machine in only one way i.e for the same i/p symbol & same stack symbol, there must be only one choice.

Formally, a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is deterministic if

i) $\delta(q, a, Z)$ has only one element.

ii) $\delta(q, \epsilon, Z)$ is not empty, then $\delta(q, a, Z)$ should be empty.

If conditions i) & ii) are satisfied, then PDA is deterministic otherwise nondeterministic.

NPDA:-

A PDA is nondeterministic, if there's some string that can be processed by it in more than one way.

There are 2 types of NPDA (two types of non

a) 1st kind of nondeterminism occurs,

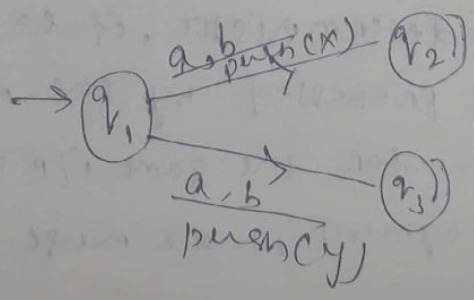
when a state emits 2 or more edges labelled with the same i/p symbol & same stack symbol.

i.e for same i/p symbol & same stack symbol, there exist number of choices. which is represented as

$\delta(q, a, Z) = \{(P_1, \gamma_1), (P_2, \gamma_2) \cdots, (P_n, \gamma_n)\}$ where

P_i & q are states, $a \in \Sigma$, Z: stack symbol

$\gamma_i \in \Gamma^*$



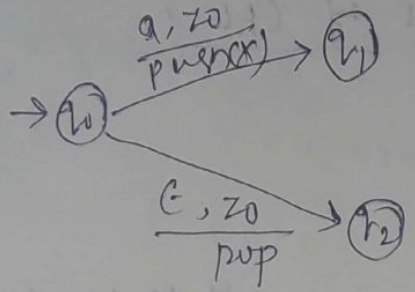
$\delta(q_1, a, b) = (q_2, x) \Leftrightarrow (q_1, a, b, push(x), q_2)$

$\delta(q_1, a, b) = (q_2, y) \Leftrightarrow (q_1, a, b, push(y), q_2)$

b) 2nd kind of nondeterminism occurs, when a state emits two edges labelled with the same stack symbol, where one i/p symbol is ϵ & the other i/p symbol is not. which is represented as formally

$\delta(q_1, \epsilon, z) = \{(p_1, \alpha_1), (p_2, \alpha_2) \ldots (p_n, \alpha_n)\}$

& $\delta(q_1, a, z) = \{(p_1, \alpha_1), (p_2, \alpha_2) \ldots (p_n, \alpha_n)\}$



→ $(q_0, a, z_0, push(x), q_1)$

→ $(q_0, \epsilon, z_0, nop, q_2)$

NOTE:-

DFA = NFA

DPDA ≠ NPDA ⇒ DPDA < NPDA

i.e → For FA, nondeterminism doesn't increase the power of machines.

→ For PDA, nondeterminism does increase the power of machine.

Closure properties of CFL:-

1. CFL are closed under union
2. CFL " " concatenation
3. CFL " " Kleene closure
4. " " " homomorphism
5. " " " substitution
6. CFL " " " inverse homomorphism

Not closed:-

1. CFL are not closed under intersection.
2. CFL " " " complementation

Decision Properties of CFL:-

1. Emptiness
2. Finiteness

Emptiness:-

A language L is said to be empty if S (starting symbol) is an useless symbol or $L = \phi$

eg:- $S \rightarrow AB$

$A \rightarrow BC$

$C \rightarrow \wedge$

$\therefore L = \phi$, so empty language.

Finiteness:-

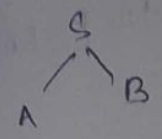
A language L is finite if graph can be constructed using variables of productions that contains no cycle.

eg:-

verify the grammar is finite or infinite.

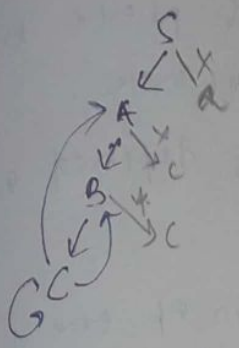
e.g. $S \to AB$
$A \to a$
$B \to b$

(Terminals not taken)



As here no cycle is present. So it's finite.

2. $S \to Aa$
$A \to Bc \mid a$
$B \to Cc \mid b$
$C \to AB \mid C$



BCZ terminals are not considered here

∴ Hence L is infinite due to the presence of cycles.

Equivalence of PDA & CFG :-

The CFG (G) & PDA (P) are said to be equivalent iff $L(G) = L(P)$

Otherwords equivalence of PDA & CFG means the class of languages accepted by CFG is exactly same as the class of languages accepted by PDA i.e

It's possible to convert any CFG to PDA such that $L(G) = L(P)$ & vice versa ($L(P) = L(G)$)

Conversion from CFG to PDA :-

Let $G = (V, T, P, S)$ be the CFG & $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ be the PDA. Then the conversion from G to P proceeds in following steps.

Step 1:- check the given grammar is in GNF or not.

→ If the given grammar is not in GNF, then convert it into GNF.

Step 2:- Initially consuming any i/p, change the state from q_0 to q_1 & place the start symbol of G onto stack.

i.e the transition is defined as

$$\delta(q_0, \epsilon, z_0) = (q_1, Sz_0)$$

Step 3:- If there's a production of the form $A \to a\alpha$, then the corresponding transition is $\delta(q_1, a, A) = (q_1, \alpha)$

Step 4:- To state q_1, on encountering the end of the i/p, if z_0 is present at the top of stack, then change the state to q_2 & done after the content of the stack.

The transition is given as

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

2) Construct a PDA for the following grammar.

$S \to aAA$

$A \to aS \mid bS \mid a$

Ans:

Step 1:-

$S \to aAA$

$A \to aS \mid bS \mid a$

given grammar is in GNF.

Step 2:- $\delta(q_0, \epsilon, Z_0) = (q_1, SZ_0)$

Step 3:-

Grammar	PDA
$S \to aAA$	$\delta(q_1, a, S) = (q_1, AA)$
$A \to aS$	$\delta(q_1, a, A) = (q_1, S)$
$A \to bS$	$\delta(q_1, b, A) = (q_1, S)$
$A \to a$	$\delta(q_1, a, A) = (q_1, \epsilon)$

Step 4:

$\delta(q_1, \epsilon, Z_0) = (q_2, Z_0)$

So the equivalent PDA, $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

$Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{S, A, Z_0\}$

$q_0 = q_0$, $Z_0 = Z_0$, $F = \{q_2\}$

3) Construct a PDA for the following grammar.

$S \to aA$

$A \to aABD \mid bB \mid a$

$B \to b$

$D \to d$.

Ans:-

Step 1:-

$S \to aA$

$A \to aABD \mid bB \mid a$

$B \to b$

$D \to a$

The given grammar is in GNF.

Step 2:- $\delta(q_0, \epsilon, Z_0) = (q_1, SZ_0)$

Step 3:-

Grammar	PDA
$S \to aA$	$\delta(q_1, a, S) = (q_1, A)$
$A \to aABD$	$\delta(q_1, a, A) = (q_1, ABD)$
$A \to bB$	$\delta(q_1, b, A) = (q_1, B)$
$A \to a$	$\delta(q_1, a, A) = (q_1, \epsilon)$
$B \to b$	$\delta(q_1, b, B) = (q_1, \epsilon)$
$D \to a$	$\delta(q_1, a, D) = (q_1, \epsilon)$

Step 4:- $\delta(q_1, \epsilon, Z_0) = (q_2, Z_0)$

So, the equivalent PDA, $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

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

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

$\Gamma = \{A, B, S, Z_0, D\}$

$q_0 = \{q_0\}$

$Z_0 = Z_0$

$F = \{q_2\}$.

$S \rightarrow asbb \mid a$

Ans:

Step 1: $S \rightarrow asBb \mid a$

The above grammar is not in GNF.

So we've to convert the grammar into GNF.

$S \rightarrow asA \mid a$ (as(bB), A, b)

$A \rightarrow bB$

$B \rightarrow b$

Now the grammar is in GNF.

Step 2: $\delta(q_0, \epsilon, Z_0) = (q_1, SZ_0)$

Step 3:

Grammar	PDA
$S \rightarrow aSA$	$\delta(q_1, a, S) = (q_1, SA)$
$S \rightarrow a$	$\delta(q_1, a, S) = (q_1, \epsilon)$
$A \rightarrow bB$	$\delta(q_1, b, A) = (q_1, B)$
$B \rightarrow b$	$\delta(q_1, b, B) = q_1, \epsilon)$

Step 4: $\delta(q_2, \epsilon, Z_0) = (q_2, Z_0)$

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

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

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

$\Gamma = \{S, B\}$

$q_0 = \{q_0\}$

$Z_0 = Z_0$, $F = \{q_2\}$