Theory Of Computation.

Textbook: Michael Sipser, Introduction to Theory of computation (3rd edition) Cengage Learning.

Reference: Jeffry Ullman (Pearson) Automata Theory book.

Theory of computing machine
↓
how a machine computes.

Theory of computation
- Automata and Formal languages. (how a machine functions.)
- Computability Theory
- Complexity Theory

Automata and Formal Languages
- Finite Automata
- Regular expression
- Push-down Automata
- context-free grammar.
- Pumping lemma

Computability Theory
- Turing machines
- Church-turing hypothesis
- Decidability
- Reducibility
- Halting

Pre-requisite:

1) Set Theory
2) Relations & Functions.
3) Strings.
4) Graphs
5) Proving Techniques.

Set → homogeneous elements grouped together having same properties.

Methods of building a set → Set Builder method
→ roasted form.

Operations: union → or
Intersection → and
Cartesian Product
Complement
set difference.

each ele of a set will be combined with other.

*when sets are merged there are no repeating elements.

Eg: $A = \{x, y, z\}$
$B = \{x, y, w\}$

$A \cup B = \{x, y, w, z\}$
$A \cap B = \{x, y\}$
$A - B = \{z\}$
$B - A = \{w\}$.
$B' = \{$ all ele except $(x, y, w)\}$

Cardinality: No. of ele present in a set.

$A \times B = \{\{x,x\}, \{x,y\}, \{x,z\}, \{y,x\}, \ldots\}$

$|A| = 3 \quad |B| = 3$ → cardinality.

$|A \times B| = 9$

Power set :- Set contains all the subsets along with the set.

$P(A) = \{\{x\}, \{y\}, \{z\}, \{x,y\}, \{x,z\}, \{y,z\}, \{x,y,z\}, \phi\}$.

$|P(A)| = 2^{|A|} = 8$

Null set : (ϕ) :- a set contains no elements :

$\{\{\}\}$ or $\{\phi\}$

Mathematical Notations and Preliminaries :-

Set Theory.

Set → Group of objects as unit.

↓ known as elements or members of the set.

2 ways of writing a set :-

$S = \{11, 15, 17, 23\}$

$S_1 = \{x \mid x$ is an real number$\}$ → formulative way.

$N = \{1, 2, 3 \ldots\}$ (natural numbers)

$N_0 = \{0, 1, 2, 3, \ldots\}$

$Z = \{\ldots -2, -1, 0, 1, 2 \ldots\}$ (integers)

Q (rational no.s), R (real no.s).

Pre-requisite:

1) Set Theory
2) Relations & Functions.
3) Strings.
4) Graphs
5) Proving Techniques.

Set → homogeneous elements grouped together having same properties.

methods of building a set → Set Builder method
→ roaster form.

operations: union → or
intersection → and
Cartesian Product
Complement
set difference.

each ele of a set will be combined with other.

*when sets are merged there are no repeating elements.

Eg: $A = \{x, y, z\}$
$B = \{x, y, w\}$

$A \cup B = \{x, y, w, z\}$
$A \cap B = \{x, y\}$
$A - B = \{z\}$
$B - A = \{w\}$.
$B' = \{$all ele except $(x, y, w)\}$

Cardinality: No. of ele present in a set.

$A \times B = \{\{x,x\}, \{x,y\}, \{x,z\}, \{y,x\}, \ldots\}$

$|A| = 3 \quad |B| = 3$ → cardinality.

$|A \times B| = 9$

Power set :- Set contains all the subsets along with the set.

$P(A) = \{\{x\}, \{y\}, \{z\}, \{x,y\}, \{x,z\}, \{y,z\}, \{x,y,z\}, \phi\}$.

$|P(A)| = 2^{|A|} = 8$

Null set : (ϕ) :- a set contains no elements :

$\{\{\}\}$ or $\{\phi\}$

Mathematical Notations and Preliminaries :-

Set Theory.

Set → group of objects as unit.

↓ known as elements or members of the set.

2 ways of writing a set :-

$S = \{11, 15, 17, 23\}$

$S_1 = \{x \mid x$ is an real number$\}$ → formulating way.

$N = \{1, 2, 3 \ldots\}$ (natural numbers)

$N_0 = \{0, 1, 2, 3, \ldots\}$

$Z = \{\ldots -2, -1, 0, 1, 2 \ldots\}$ (integers)

Q (rational no.s), R (real no.s).

In a set order, sequence and ~~repetition~~ repetition doesnot matter.

Empty set: has no elements. and can be denoted as ϕ or $\{\}$.

multi set: set contains multiple no. of elements.

Eg: $\{7, 7, 10\}$ → equivalent to $\{7, 10\}$ (repetition doesn't matter)

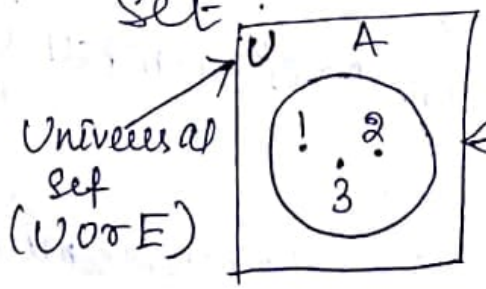
Singleton: set has a single element.

Eg: $\{a\}$.

Set operations: Union ($\cup$)
Intersection ($\cap$)
Set Difference ($-$)
Cartesian Product ($\times$)
Complement ($\bar{A}$)

Venn Diagram: Pictorial representation of a set.

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



Universal set (U or E)

← Venn Diagram.

(rupsiri) $\{\ldots -2, -1, 0, 1, 2, \ldots\} = \mathbb{Z}$

Demorgans law: (I) $(A \cup B)' = A' \cap B'$

(II) $(A \cap B)' = A' \cup B'$

Sequence & Tuple: The repeatition and order matters.

$A = \{7, 11\}$ $A_1 = \{7, 11, 11\}$, $A_2 = \{11, 7\}$

here we can say $A = A_1 = A_2$

But,

In case of sequence: The objects are in an order

$S = (7, 11)$, $S_1 = (7, 11, 11)$, $S_2 = (11, 7)$

here, $S \neq S_1$, $S \neq S_2 \neq S_1$

Sequence: keeping a group of objects in a particular order

A sequence may be finite or infinite.

But,

A finite sequence is called a **tuple**.

A sequence of k elements is called a **k-tuple**.

Eg: $S_1 = (7, 11, 11) \rightarrow$ 3-tuple

$S_2 = (7, 11) \rightarrow$ 2-tuple.

2-tuple sequences are called an **ordered pair**.

Subset: set has same or less elements in comparison to another set.

Proper subset: A set is proper subset of another set if it contains same elements & less no. of elements than that of other set.

Eg: $C = \{1,2,3\}$ $A = \{1,2,3\}$ $B = \{1,2\}$

$B \subsetneq A \rightarrow$ proper subset

$C \subseteq A \rightarrow$ subset.

Power Set: set of subsets of a set

eg: $S = \{0,1\}$

$P(S) = \{\{0\}, \{1\}, \{0,1\}, \phi\}$

↳ we cannot take proper subset else $\{0,1\}$ will be removed.

Cartesian Product

$S = \{0,1\}$ $S_1 = \{x,y\}$.

$S \times S_1 = \{(p,q) \mid p \in S \text{ and } q \in S_1\}$ → 2-tuple

$= \{(0,x), (0,y), (1,x), (1,y)\}$

(each element is an ordered pair)

$S \times S_1 \times S = \{(p,q,r) \mid (p,q) \in S \times S_1 \text{ and } r \in S\}$

$= \{(0,x,0), (0,x,1), (0,y,0), (0,y,1), (1,x,0), (1,x,1), (1,y,0), (1,y,1)\}$

↳ 3-tuples

(*) $S \times S \times S \times \ldots \times S$ (k times) $= S_k$ or S^k.
(A set multiplied with itself k no. of times).

Relations and functions

Function: is an object that ~~set~~ sets input to output relationship.
For ~~every~~ every input there exists an output.

Eg: If a is input & b is output
then we can write $f(a) = b$ (domain → a, range → b)

$f : D \to R$ (mapping from Domain to range)
D: Domain, R: range

Domain: possible set of input values.

Range: possible set of output values.

For ex:

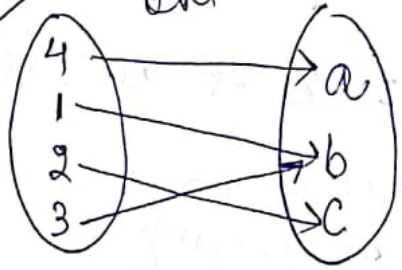
$$abs(x) = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x < 0. \end{cases}$$

~~abs : $\mathbb{Z} \to \mathbb{Z}$~~
↳ set of integers → set of positive elements along with 0

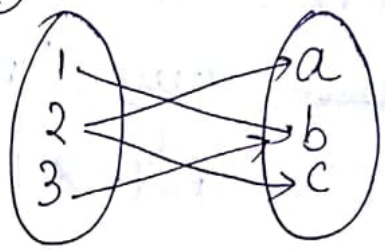
abs : $\mathbb{Z} \to$ Non-$\mathbb{Z}$ (set of integers)
↳ set of the elements along with 0

In a function, all the elements present in the domain is used but it is not necessary that all the elements present in its range will be used.

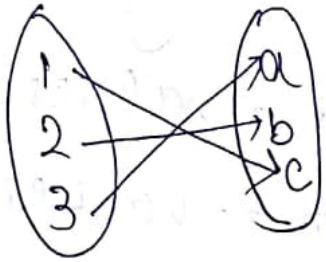
① ✓ onto.



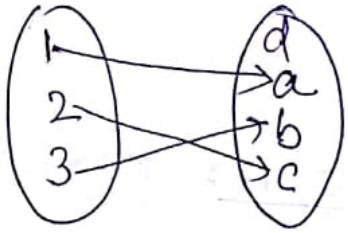
② ✗ not a function



③ ✓ onto



④ ✓



Onto function: A function that uses all elements of the range.

Function can be represented using 2 ways :-

(I) Describing the function using formulas or in language

(II) Tables with inputs and for every specified input its output.

Eg: $f(x) = (x+1)\%5$ → 1st way
$0 < x < 5$

2nd way :-

x	$f(x)$
0	1
1	2
2	3
3	4
4	0

(Domain) (Range)

When the domain of a function contains the elements which is not single / has multiple values in range → (many-one)

* When the domain of a function f is

$$A_1 \times A_2 \times \cdots \times A_k$$

where $A_1, A_2 \cdots A_k$ are same sets then input to f is a k-tuple $(a_1, a_2 \cdots, a_k)$ and 'a_i' is called argument to the function. and 'k' is called arity of the function.

Eg: $f(i,j) = (i+j)\%5$
↳ 2-ary function.
(accepts a input with 2-tuple)

Z_m (in modular arithmetic) $= \{0, 1, 2, \cdots, m-1\}$

Exp: 17 mod 5 = 2.

−17 mod 5 = −2+5 = 3 → In modular arithmetic it will have the values, no −ve values.

Let: f(i, j) = (i+j)%5, Z_5 → 2-ary/binary functions (no. of arguments is 2)

i.e Z_5 = {0, 1, 2, 3, 4}.

f(i,j)	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

(Tabular form)

Predicate / Property :-

A particular funⁿ for which the range is either True/False is called a predicate/property.

i.e,

P : D → {TRUE, FALSE}.
→ is a predicate.

even(4) = TRUE

even(7) = FALSE

Relation: - is a predicate whose domain is a k-tuple -

or

A property whose domain is a k-tuple is called a relation.

$A \times A \times A \times \ldots \times A$ is a relation or a k-ary relation

A 2-any relation is called a binary relation.

*If 'R' is a binary relation, i.e a R b.

a R b = TRUE

* A set of relation is established means the range is true for the relations.

Game of scissors, paper & stones:

Let S = {SCISSORS, PAPER, STONE}

$P_1 \times P_2$ (Two players choose arbitrary elements if both choose same element then game starts again if not then one of them wins)

9 combinations possible :-

$P_1 \times P_2$ = {(SCISSORS, SCISSORS), (SCISSORS, PAPER), (SCISSORS, STONE), (PAPER, SCISSORS), (PAPER, PAPER), (PAPER, STONE), (STONE, SCISSORS), (STONE, PAPER), (STONE, STONE)}

↳ Domain (D)

In tabular form :-

P1 \ P2	SCISSORS	PAPER	STONE
SCISSORS	FALSE	TRUE	FALSE
PAPER	FALSE	FALSE	TRUE
STONE	TRUE	FALSE	FALSE

A relation:
P_1 beats P_2.
$(P_1 \; R \; P_2)$
SCISSOR R PAPER → true relation

We can define a relation by:
$S = \{a \in D \mid P(a) = TRUE\}$ (predicate of element a.)

$= \{(SCISSORS, PAPER), (PAPER, STONE), (STONE, SCISSORS)\}$

* A binary relation is sometimes called an equivalence relation if it obeys the following three properties:-

(i) Reflexive: A relation R is reflexive if for every x, xRx.

(ii) Symmetric: A relation R is symm. if for every x and y:
$xRy \rightarrow yRx$.

(iii) Transitive: A relation R is transitive if for every x, y & z:
xRy and $yRz \longrightarrow xRz$. (implies that)

Eg: Prove that on the set of Natural numbers the relation: $\equiv_7$, $i, j \in N$. says that $i \equiv_7 j$ i.e. $(i-j)$ is divisible by 7

is an equivalence relation.

Ans) (i) If $i \in N$ then
$i - i = 0$ and 0 is divisible by 7
So $i R i$ hence relation is reflexive

(ii) If is given: $i, j \in N$.
$i - j = 7k$ (given).
$\Rightarrow j - i = -7k = 7(-k)$.
So divisible by 7
(So symmetric (Proved).

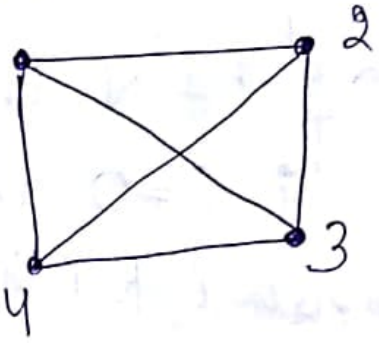
(iii) $i - j = 7m$.
$j - k = 7n$.
adding: $i - j + j - k = 7m + 7n$
$i - k = 7(m+n)$ (div by 7)

Hence $i R k$ so transitive.

Hence relation is an equivalence relation.

GRAPH

A graph can be defined as G(V, E) where V is set of vertices and E is set of edges.



V = {1, 2, 3, 4}

E = {(1,2), (2,3), (3,4), (1,3), (2,4), (1,4)}.

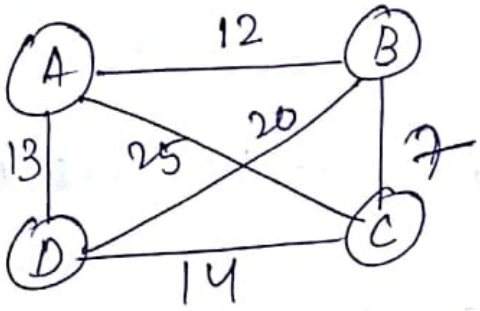
{as undirected graph so sequence doesn't matter (1,2) = (2,1)}.

Degree: The no. of edges at a particular node.

degree(1) = 3
degree(2) = 3
degree(3) = 3
degree(4) = 3

Self loop: An edge from a node to itself.

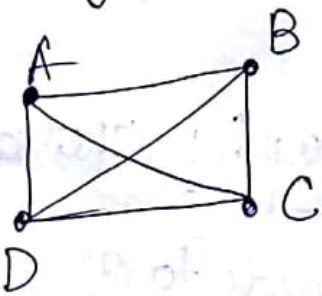
Labeled graph: If the nodes and edges of a graph are labeled with some values then graph is called labeled graph.



path: in a graph is a sequence of nodes connected by edges.

Simple path: when there is no repeatition of nodes. eg: ABC → simple.
ABDAC → not simple.

Connected Graph: If a graph is having 'n' nodes & every node has a path to every other node then it is a connected graph.
A graph is connected if every 2 nodes has a path between them.



cycle: If a path starts & ends with the same node then it is called a cycle.
Eg: ABCA

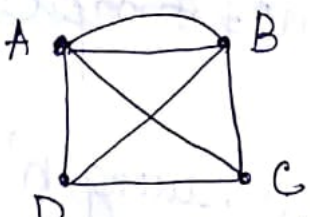
Simple cycle:

(i) On a simple cycle atleast 3 nodes are there

(ii) And only start & end node repeats no other node.

not a simple cycle.

A simple cycle.
ABCA



ACBABDA
It is a cycle but not a simple cycle.

Tree: A connected graph without any simple cycles.

Directed Graph: The edges have direction from one node to other. Each edge has a direction from a particular vertex to some other vertex.

Degree of a directed graph is of 2 types:

(a) Indegree : no. of edges coming into the vertex

(b) Outdegree : no of edges coming out of the vertex.

edges incident to the vertex.

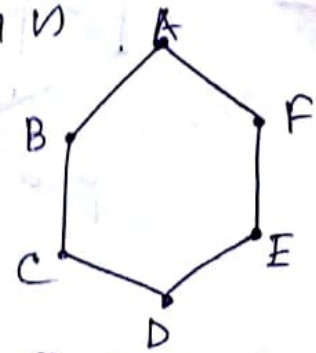
Directed path: A path in which all arrows are in same dirⁿ.

Strongly connected: If there is a directed path bet. every 2 nodes in a directed graph.

Regular graph:

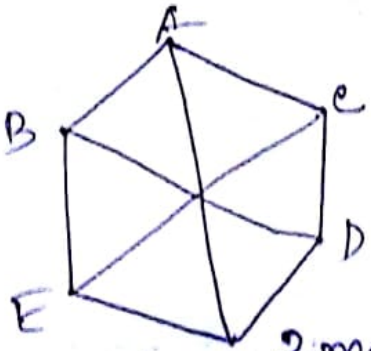
In a graph if degree of each node is same then it is a regular graph.

If degree is 'k' then graph is called k-regular graph.



degree = 2.

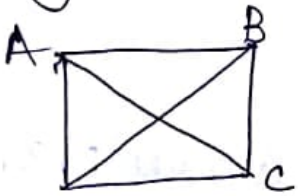
2-regular graph.



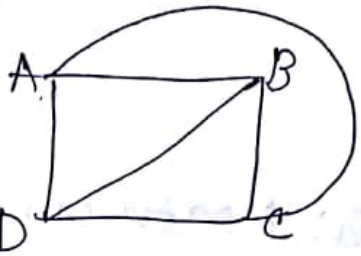
3-regular graph.

Planar graph :-

which can be drawn on a plane without any intersection of edges.

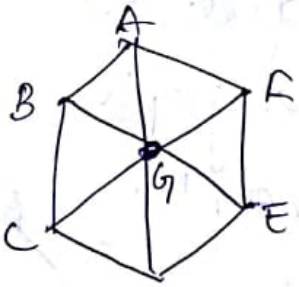


Not planar

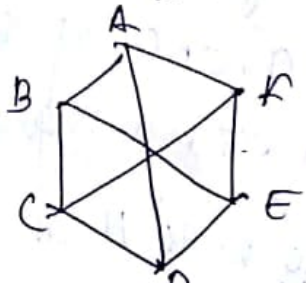


planar.

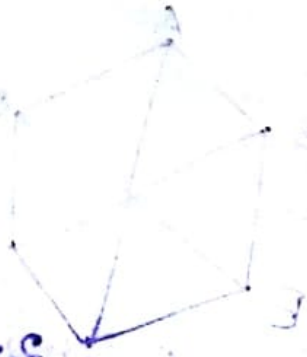
Planar graph is a graph that can be embedded in a plane in a way such that its edges only intersect at the end points or nodes (i.e no edges cross each other).



planar.



not planar



3-regular graph.

STRING :-

Alphabet : is denoted as (Σ)

↓

It is a non-empty finite set of symbols.

Some alphabet sets:

$\omega_1 = acpt$, $\omega_2 = cat$ $\Sigma_1 = \{a, b, \ldots, z, A, B, \ldots, Z\}$ → english

$\omega_3 = 0110$, $\omega_4 = 1111$ $\Sigma_2 = \{0, 1\}$ → binary System

$\omega_5 = 921$, $\omega_6 = 123$ $\Sigma_3 = \{0, 1, 2, \ldots, 9\}$ → No. system

String (ω) :- Strings over an Alphabet is a finite sequence of symbols from Alphabet set.

Empty String :- If a string contains no characters it is denoted as ϵ (epsilon). i.e string of length 0.

Length of a string :- no. of symbols or characters present in a string.

Eg : $\omega_1 = acpt$ $|\omega_2| = 3$

$|\omega_1| = 4$

Prefix: 'x' is a prefix of y where 'x' and 'y' are strings

i.e, $xz = y$.

x → prepart of string

z → next part

Eg:

Let $y = abc$

So x can be : $x = \epsilon, a, ab, abc$.

So ~~$x = abc, x = bc, x$~~

$z = abc, bc, c, \epsilon$

Proper prefix: 'x' is prefix of y equality is avoided.

i.e $xz = y$ and $x \neq y$.

Eg: Let $y = abc$

x can be : $x = \epsilon, a, ab$

(x ← prefix)

Suffix: 'x' is a suffix of y i.e

$zx = y$.

Eg: If $y = abc$ So $x = \epsilon, c, bc, abc$.

proper suffix: 'x' is a suffix of y i.e

$x = y$ & $x \neq y$

eg: if $y = abc$ so $x = \epsilon, c, bc$

substring: x is a substring of, a string w. if x appears consecutively in w.

eg: $w = abbbaab$ ← $x = bba$

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

↳ so x is substring of w

$w_1 = aababab$

↳ but x is not substring of w_1.

reverse: Considering a string 'w', its reverse is denoted as w^R and is the string found out by writing 'w' in opposite order.

eg: $w = abbb$

$w^R = bbba$

Concatenation: of two strings x and y is represented as $x.y$ or xy.

Eg: If $x = ab$, $y = abb$

$xy = ababb$.

$x.x = abab$.

$x.x.x = ababab$.

If a string w concatenated with itself 'k' times it is represented as:

w^k i.e

$x^2 = (ab)^2$ $\quad x^3 = (ab)^3$

*) Imp

Whenever we represent a set of strings we should follow lexiographic or ascending order. $\{\in, 0, 1, 00, 10, 000 \ldots\}$

Combination of string

~~Language:~~

* A language is a set of strings.

Boolean Logic

Boolean values :- True (1)
False (0)

Boolean variables :- P (0 | 1) Q (0 | 1)

P	Q	Conjuction (AND) (∧)	Disjunction (OR) (∨)	Negation (NOT) $\bar{P}$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

P	Q	Conjuction (AND) (∧)	Disjunction (OR) (∨)	Negation (NOT) $\bar{P}$	Negation (NOT) $\bar{Q}$	Implication $P \rightarrow Q$	Equality $P \leftrightarrow Q$	Exclusive OR (⊕) $P \oplus Q$
0	0	0	0	1	1	1	1	0
0	1	0	1	1	0	1	0	1
1	0	0	1	0	1	0	0	1
1	1	1	1	0	0	1	1	0

negation of equality

Equivalence Relations: (We can perform any kind of circuit by 'not' & 'and' gates)

(i) $P \vee Q \equiv \neg(\neg P \wedge \neg Q)$

(ii) $P \rightarrow Q \equiv \neg P \vee Q$

(iii) $P \leftrightarrow Q \equiv (P \rightarrow Q) \wedge (Q \rightarrow P)$

(iv) $P \oplus Q \equiv \neg(P \leftrightarrow Q)$

Distributive Laws:

(i) $P \vee (Q \wedge R)$
$= (P \vee Q) \wedge (P \vee R)$

(ii) $P \wedge (Q \vee R)$
$= (P \wedge Q) \vee (P \wedge R)$

De-morgans law:

(i) $\neg(P \vee Q) = \neg P \wedge \neg Q$

(ii) $\neg(P \wedge Q) = \neg P \vee \neg Q$.

Definitions, Theorems and Proofs

Def": means describing the objects and notations that we use.

Mathematical Statements: expresses the properties of objects.

Proof: - is a logical statement / argument used for convincing a statement is true.

Theorem: - A mathematical statement proved true.

Theorem has 2 parts :- (i) lemma
(ii) corollary.

lemma: Some statements are proved as they assist in proving some more significant statements are called lemmas.

Corollary: A theorem or proof that allows us to conclude that some other related statements are true for corollaries.

Proof Techniques

Proof by Construction :-

Many theorems state that a particular type of object exists and such theorems are proved by demonstrating how to construct the object.

Theorem: For each even number greater than 2 there exist a 3-regular graph with 'n' nodes.

Proof: Let 'n' be the number of nodes i.e 'n' is even number greater than 2.

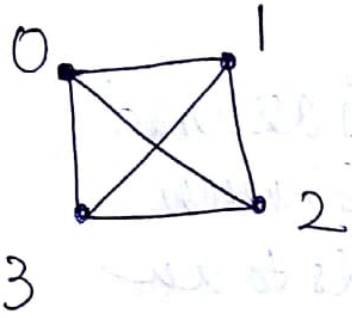
A graph G(V, E) with 'n' nodes can be constructed with the set of vertices: $V = \{0, 1, 2, \ldots n-1\}$ and the set of edges

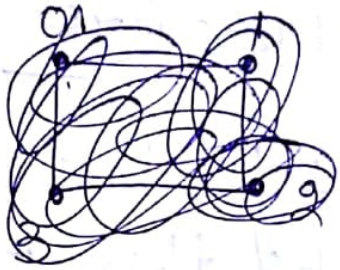
$$E = \{\{i, i+1\} \mid \text{for } 0 \le i \le n-2\} \cup \{\{n-1, 0\}\} \cup \{\{i, i+n/2\} \mid \text{for } 0 \le i \le \frac{n}{2}-1\}$$

Note: The circle concept tells that if a node is there on a circumference of circle then a 3-regular graph can be there if there is an edge between adjacent nodes & also with opp. nodes.

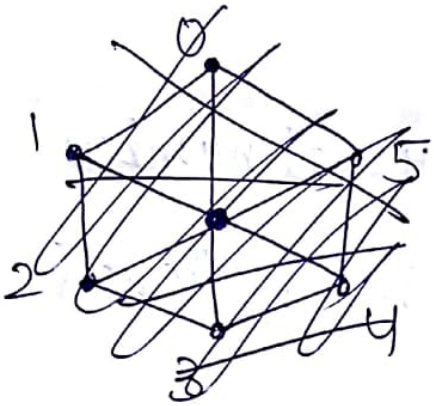
3-regular graphs using this concept are can constructed

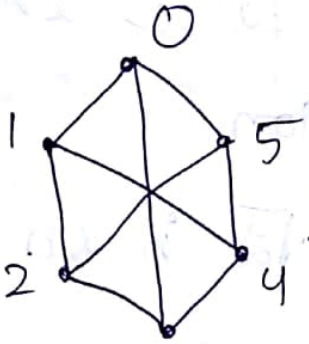
For n=4:





For n=6:





3regular.

Picturing the nodes of the graph written consecutively on the circumference of a circle. i.e edges between adjacent pairs and opp side of the circle.

The figures shows that every node in 'G' has a degree 3 and the graph is 3-regular.

Hence proved.

Proof by Contradiction.

In this technique the theorem is assumed to be false and then it is shown that the assumption leads to an obviously false consequence called a contradiction.

Theorem: $\sqrt{2}$ is an irrational number

Proof: Assuming that $\sqrt{2}$ is a rational number.

So, $\sqrt{2} = \frac{m}{n}$ where m and n are any integer ($n \neq 0$)

Dividing m and n by their gcd the function value doesnot change.

and atleast one of m and n is an odd no.

multiplying both sides by n:

$\sqrt{2}n = m$.

Squaring both sides:

$2n^2 = m^2$

Hence m^2 is 2 times of n^2, so

m^2 is even

$\Rightarrow$ m is even (square of an odd number is always odd).

As m is even

So m can be written as:

$m = 2k$ for any integer k

Substituting value of m in $2n^2 = m^2$:

$2n^2 = (2k)^2$

$\Rightarrow 2n^2 = 4k^2$

$\Rightarrow n^2 = 2k^2$

Hence n^2 is also even

So n is even (as square of odd no. is always odd).

So we conclude m & n both are even which is not possible as it violates the rational property (atleast one of m & n should be odd). Earlier we had reduced m & n so that both are not even. i.e it leads to a contradiction.

Hence our assumption is false, so $\sqrt{2}$ is irrational (Proved)

Chapter 1:

Questions :-

Formal description of sets : $\{1, 2, 3 \ldots\}$ or

$\{x \mid x \in N\}$

Informal description : The set containing all positive nos.

① (i) $\{1, 3, 5, 7 \ldots\}$

Ans) All +ve odd numbers

(ii) $\{\ldots -4, -2, 0, 2, 4 \ldots\}$

Ans) All even integers

(iii) $\{x \mid x = 2n$ for some n in N and $x = 3k$ for some k in $N\}$.

Ans) All multiples of 6 from the natural no. set N.

(iv) $\{w \mid w$ is a string of 0's and 1's and w equals reverse of $w\}$

Ans) All palindrome strings over the binary alphabet.

(v) $\{n \mid n$ is an integer and $n = n+1\}$

Ans) No number can be equal to that no. plus 1.

So ϕ set (Null set).

(2) (i) A set containing an empty string.
↳ means it is denoted by $\in$

Ans) $\{\in\}$.

(ii) A set containing nothing.

Ans) $\{\}$ or ϕ.

(iii) A set containing a string 'aba'.

Ans) $\{aba\}$.

(iv) A set containing all natural no.s less than 5

Ans) $\{1,2,3,4\}$ or $\{x \mid x \in N \text{ and } x < 5\}$.

(3) Let $A = \{x, y, x\}$, $B = \{x, y\}$.

$A \subseteq B$: F

$B \subseteq A$: T

$B \subseteq A$: T.

$A \times B = \{\underbrace{(x,x)}_{\text{↳ 2-tuple}} (x,y) (y,x) (y,y) (x,x) (x,y)\}$.

Powerset $(B) = \{\{x\}, \{y\}, \{x,y\}, \phi\}$

(4) Set A has 'a' no. of element, set B has 'b' no. of elements.

no. of elements in $A \times B$ is ab.

$|A \times B| = ab$.

If $|B| = c$ then $|P(B)| = 2^c = 2^{|B|}$

(6) Let there are 2 sets : X and Y :

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

$Y = \{6, 7, 8, 9, 10\}$

Unary function $f : X \to Y$ which can be represented as:

n	f(n)
1	6
2	7
3	6
4	7
5	6

(f(n) → range, n → domain.)

Another binary function $g : X \times Y \to Y$ and can be represented as:

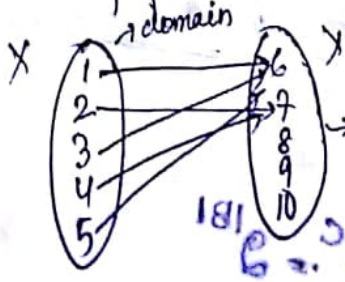
g	6	7	8	9	10
1	10	10	10	10	10
2	7	8	9	10	6
3	7	7	8	8	9
4	9	8	7	6	10
5	6	6	6	6	6

(i) $f(2) = 7$

(ii) Range and domain of f:

Range : of $f = \{6, 7, 8, 9, 10\} = Y$

domain : of $f = \{1, 2, 3, 4, 5\} = X$



(iii) $g(2,10) = 6$.

(iv) Domain of $g = X \times Y$.

$= \{(1,6)(1,7)(1,8)(1,9)(1,10)$
$(2,6)(2,7)(2,8)(2,9)(2,10)$
$(3,6)(3,7)(3,8)(3,9)(3,10)$
$(4,6)(4,7)(4,8)(4,9)(4,10)$
$(5,6)(5,7)(5,8)(5,9)(5,10)\}$

25 elements

Range of $g = Y = \{6,7,8,9,10\}$.

(v) Function g is onto, function f is not onto

(6) Give some ex of relations

(i) reflexive & symm but not transitive.

(ii) reflexive & transitive but not symm.

(iii) Symm. & transitive but not reflexive.

Ans) (i) Let $S = \{1,2,3\}$

$R = \{(1,1)(2,2)(3,3)(1,2)(2,1)(2,3)(3,2)\}$

↳ reflexive & symm.

(1,1) ∈ R

(1,2) — (2,1)

(ii) Let R is a relation of set of real no.s.

$R = \{(a,b) \mid a \leq b\}$.

xRy , $x < y$.

yRz , $y < z$.

$\therefore x < z \rightarrow xRz$ (Transitive)

xRx , $x = x$ (reflexive).

$xRy \rightarrow x < y$.

but

$yRx \rightarrow y \not< x$ (not symm.)

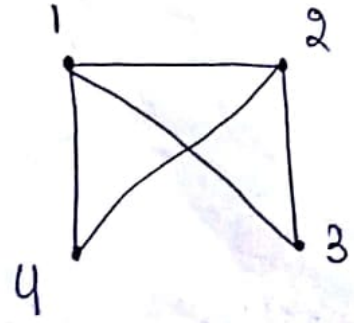
(iii) Consider a case where R is empty relation

(8) Draw a graph $G(V, E)$ with set of vertices $V = \{1,2,3,4\}$ and

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

↳ if edges are in form of sets then it is undirected

If its in form of ordered pairs then it is directed.



(1) Find out the degree of each node:

degree(1) = 3
degree(2) = 3
degree(3) = 2
degree(4) = 2.

Sum = 10 (twice the no. of edges).

(10), (11), (12), (13) → do it yourself.

(12) Prove by induction that $C(n) = (S(n))^2$

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

$$C(n) = 1^3+2^3+\dots+n^3 = \frac{1}{4}(n^2(n+1)^2)$$

Proof: For n=1: S(1) = 1
C(1) = 1
So $S(1)^2 = C(1)$.

For n=2: S(2) = 3
C(2) = 9.
$C(2) = (S(2))^2$
$\Rightarrow 9 = 3^2$.

Let us assume that for n=k:

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

$$C(k) = 1^3+2^3+\dots+k^3 = \frac{1}{4}(k^2(k+1)^2)$$

$C(k) = (S(k))^2$

for $n=k+1$:

$$C(k+1) = 1^3+2^3+3^3+\dots+k^3+(k+1)^3$$

$$=(1+2+3+\dots+k)^2+(k+1)^3$$

$$= \left(\frac{k(k+1)}{2}\right)^2+(k+1)^3$$

$$= (k+1)^2\left\{\frac{k^2}{4}+k+1\right\} = \frac{(k+1)^2\{k^2+4k+4\}}{4}$$

$$=\frac{(k+1)^2(k+2)^2}{4}$$

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

$$=(S(k+1))^2 = \text{RHS (proved)}$$

(13). Considering the eqn $a=b$.

multiplying both sides by a.

$$a\times a = b\times a$$

$$\Rightarrow a^2 = ba.$$

Subtracting b^2 from both sides:

$$a^2-b^2 = ba-b^2$$

Factorising both sides.

$$(a+b)(a-b) = b(a-b)$$

Dividing both sides by $a-b$; → not possible as $a-b=0$ we cannot divide by 0

$$a+b = b.$$

Let $a=b=1$ so $1+1=1 \Rightarrow 2=1$ ✗

Chapter 2: Theory of Computation.

An idealised computer, called computational model as real computer is used to find out a mathematical theory which is complex.

Types of Computational Model

(1) Finite Automaton (FA).

(2) Push-down Automaton (PDA).

(3) Linear Bounded Automaton (LBA).

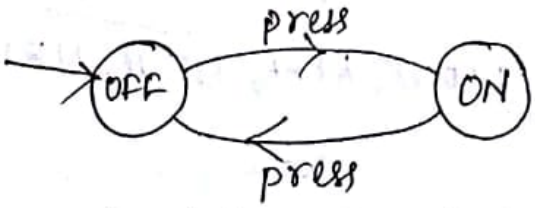
(4) Turing Machine (TM).

(5) Post Machine (PM).

Finite Automaton (FA).

It is the simplest computational model with limited memory.

*Initially we have to start from a state that exists and that state or switch is off initially.



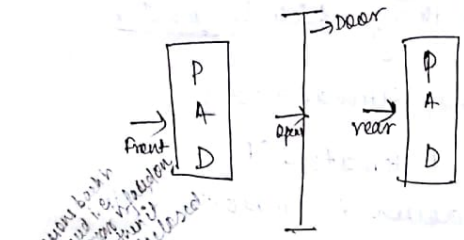
Basic Or Finite Automaton of an ON/OFF switch.

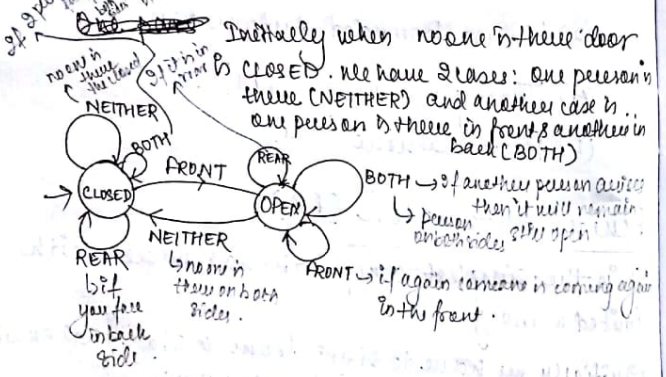
If we provide inputes in form of 'press' continuously then the switch goes on, off, on - - -

Two things are required: Input and State.

Inputs : press press press
State : OFF ON OFF ON.

Automated Door :





Initially when no one in the door is CLOSED. We have 2 cases: one person in there (NEITHER) and another case is.. one person is there in front & another in back (BOTH)

BOTH → if another person arrives then it will remain open

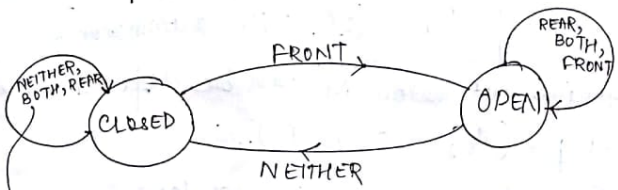
FRONT → if again someone is coming again in the front.

States are :- {OPEN, CLOSED}.

Inputs are : Σ = {FRONT, REAR, BOTH, NEITHER}

State change :

	FRONT	REAR	BOTH	NEITHER
CLOSED	OPEN	CLOSED	CLOSED	CLOSED.
OPEN	OPEN	OPEN	OPEN	CLOSED.

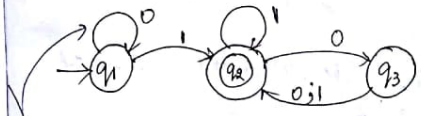


Simplified diagram

both sides there are persons with their back faced to the door.

Mathematical concept

Description of Mathematical Theory of Automaton



① State diagram
3 states q_1, q_2 and q_3.

Finite Automaton M,
each arrow is labelled by an input symbol

⇒ input alphabets :- 0, 1.
⇒ each state is represented by a circle.
⇒ Arrows are representing the transitions.

→ start state / initial state (q_1)
↳ (indicated by an arrow from no where)

→ accept state / final state → q_2 (represented by a double circle) ◎

Formal definition:- (of finite automaton)

A finite automaton 'M' can be defined as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

Q = A finite set called states.

Σ = A finite set called alphabet. (input)

$\delta : Q \times \Sigma \longrightarrow Q$ (state with an input going for another state)

$q_0 \in Q$ = q_0 is an initial state / start state

$F \subseteq Q$ is the set of accept / final states.

δ is a transition state.

Note:- (i) For a finite automaton the number of initial states is '1'.

(ii) The no. of final states :
0 or more no. of accept states are possible :-

$$0 \leq |F| \leq |Q|$$

(iii) If a finite automaton contains 'n' no. of states i.e $|Q| = n$

and 'm' no. of input symbols
i.e $|\Sigma| = m$

The total number of transitions will be :

$$|Q| \times |\Sigma| = nm$$

Eg:

Finite automaton M_1 → 2 transitions

here
$|Q| = 3$
$|\Sigma| = 2$
so no. of transitions is $|Q| \times |\Sigma| = 6$

(iv) The transition function (δ) specifies exactly one next state for each possible combination of a state and input symbol.

In above ex: $Q = \{q_1, q_2, q_3\}$

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

δ : is represented by a transition table.

q_1 is the initial state.

$F = \{q_2\}$.

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

* Finite automaton is nothing but a finite state machine or computational machine.

eg Considering a string input to the automaton M, ∑ :

as w = 1101, the string is processed and an output is produced.

Ans output is either accept or reject

Initially in state q_1

w = 1101

Read 1 follow the transition from q_1 to q_2

Read 1 follow the transition from q_2 to q_2

Read 0 follow the transition from q_2 to q_3.

Read 1 follow the transition from q_3 to q_2

Finally we reach on q_2 which is an accept state. acc^n to diagram & w is accepted.

These strings are accepted which after processing reach on the accept state.

Mathematically,

$\delta(q_1, 1) \rightarrow q_2$
$\delta(q_2, 1) \rightarrow q_2$
$\delta(q_2, 0) \rightarrow q_3$
$\delta(q_3, 1) \rightarrow q_2$

} In form of transition function.

Transition fun^n can be represented in 2 ways:
(1) General way
(2) Transition Table

In a single line:

$\delta(q_1, 1101) \rightarrow \delta(q_2, 101) \rightarrow \delta(q_2, 01) \rightarrow \delta(q_3, 1) \rightarrow \delta(q_2, \in)$

$\in$ → no input or empty.

$\delta(q_1, 1101\$) \rightarrow \delta(q_2, 101\$) \rightarrow \delta(q_2, 01\$) \rightarrow \delta(q_3, 1\$) \rightarrow \delta(q_2, \$)$

$\$$ → to mark end of string

Q) Find strings accepted in the automaton.

→ language.

{1, 01, 100, 101, 01100, ...}

Note:
If A is the set of all strings that machine M accepts, then we say that A is the language of machine M.

$L(M) = A$.

For the automaton above :

A_1 = {w | w contains atleast one 1 and an even no. of 0's follow the last 1}

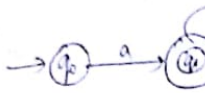
⇓

or

{w | w contains atleast one 1 and it is finished with either a 1 or an even no of 0's}.

$L(M) = A_1$

1)



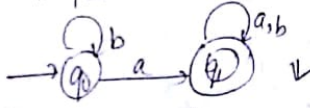
✗ not a Deterministic finite automaton as no. of transitions here is 2 It should be 4.

	a	b
q_0	q_1	✗
q_1	✗	q_1

not poss.

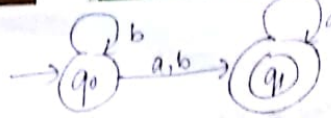
as $\Sigma = \{a, b\}$ $Q = \{q_0, q_1\}$

$|\delta| = |\Sigma| \times |Q| = 4$



✓ A DFM

	a	b
q_0	q_1	q_0
q_1	q_1	q_1

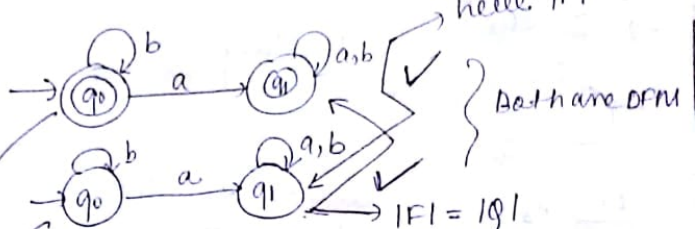


✗ not a DFM

here q with b input q_0 is having transition to q_0 as well as q_1 which is not poss.

{For a DFM, more than one transition for a same input is not possible}

$\delta(q_0, b) = q_0$
$\delta(q_0, b) = q_1$ } ✗



here $|F| = 0$.

✓

Both are DFM

$|F| = |Q|$

here all strings of {a,b} will be accepted as both state are final.

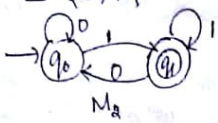
here no strings will be accepted.

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

A DFM with one state. (accepts all strings over {a,b}) an empty string. ∈

whenever a state is both final & initial so if we give no input i.e it is also accepted

⇒ $\Sigma = \{0, 1\}$



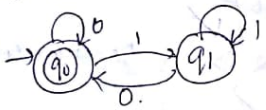
M_2

Find language of M_2

Ans) $L(M_2) = \{1, 101, 001, 11, 011, \ldots\}$

$A_2 = \{w \mid w \text{ ends with } 1\} = L(M_2)$

⇒ $\Sigma = \{0, 1\}$

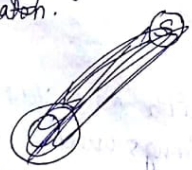


Find language.

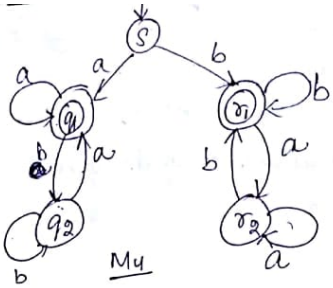
Ans) $A_3 = \{w \mid w$ ends with 0 or w is the empty string $\in\}$

$L(M_3) = \{\in, 0, 0110 \ldots\}$

⇒ Find language of: the following finite automaton.



P.T.O



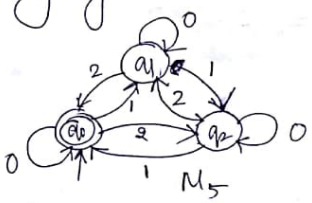
M_4

Ans) $L(M_4) = \{a, aaba, b, bb, baab \ldots\}$

$A_4 = \{w \mid w$ starts with 'a' and ends with 'a' or starts with 'b' ends with a 'b'$\}$

$= L(M_4)$

Q) Find language:



M_5

Ans)

$L(M_5) = \{\in, 0, 012, 0111, 0100100, 102, 21$

$= \{\in, 012, 12, 102$ (Starting with 0 & ends with 2)

21, 201, ... (starting with 2 & ends with 1)

012, 021 .. (starts with 0, if

$\{000 \ldots, 12, 1212, 121212,$

all no. of 12 skipped $21, 2121, 212121, 111, 111111 \ldots,$

222, 222 222 ——— , }

9+7+7 = 6 multiple of 3.

A_{75} = { w | w running count of the sum of numerical input symbol is multiple of 3 }

q) → (q0) M_6.

Ans) $L(M_6) = \phi$.

p) → (q0) M_7

Ans) $L(M_7) = \{\epsilon, \ldots\}$.

Formal def^n of computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ is a finite automaton

Let $w = w_1 w_2 w_3 \ldots w_n$ be a string where w_i is the element of an alphabet i.e $w_i \in \Sigma$.

Then M accepts w if the sequence of states $r_0 r_1 \ldots r_n$ in Q with the following conditions :-

(i) $r_0 = q_0$. (There must be an initial state)

(ii) $\delta(r_i, w_{i+1}) = r_{i+1}$ (Transition from r_i with input w_{i+1} to r_{i+1}) for $i = 0, 1, \ldots, n-1$

(iii) $r_n \in F$ (The last state should be final state)

* M recognizes A if $A = \{w \mid M \text{ accepts } w\}$.

Note:- A language is called a regular language if some finite state machine recognizes it.

Designing Finite Automaton

Power of an alphabet

Let $\Sigma = \{a, b\}$ is an alphabet

Σ^0 = set of all strings of length 0.
$= \{\epsilon\}$.

Σ^1 = set of all strings of length 1.
$= \{a, b\}$.

Σ^2 = set of all strings of length 2
$= \{aa, ab, ba, bb\}$.

Σ^n = set of all strings of length n.
= { } (n times occurrence of any symbol)

No. of elements or strings present in Σ^n.
$= |\Sigma|^n$

can be filled in 2 ways (a or b).

$\frac{P_1}{2} \times \frac{P_2}{2} \times \cdots \times \frac{P_n}{n} \rightarrow 2^n$ times → Total no. of strings

Σ^* - set of all strings that can be formed over Σ (strings of any length)

↓ Kleen closure

$= \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cdots \cup \Sigma^n \cdots$

$= \{\epsilon\} \cup \{a, b\} \cup \cdots \cup \Sigma^n$

$= \{\epsilon, a, b, ab, ba, aa, bb \cdots\}$

= set of all strings of any length.

* If $\Sigma = \{a, \cdots x\}$
then Σ^* is the set of all words possible in the english alphabet.

Σ^+ - set of all non-empty strings.

$= \Sigma^1 \cup \Sigma^2 \cdots \cup \Sigma^n$

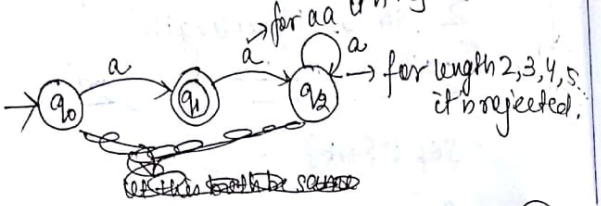
$= \{a, b \cdots\}$

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

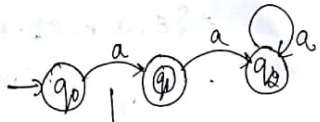
Q) Design a deterministic FA over the alphabet set: $\Sigma = \{a\}$ that accepts.

(i) all strings of length 1.
(ii) all strings of length 2
(iii) all strings of length 3.

Ans) (i) $L = \{a\}$.



Final design:



(ii) $L = \{aa\}$

(iii) $L = \{aaa\}$

When the length is a final value i.e 'l':

then no. of states $(n) = l+2$

Q) Construct a Deterministic FA over the alphabet set: $\Sigma = \{a, b\}$ where.

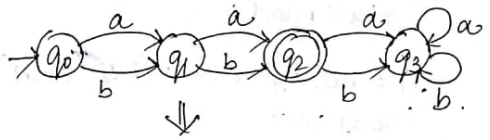
(i) The length of all strings is exactly 2.

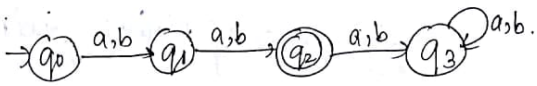
(ii) The length " " is atleast 2

(iii) " " " is atmost 2.

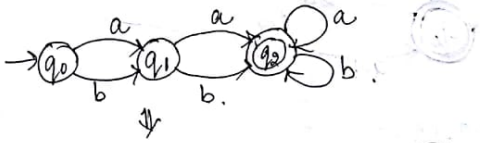
Ans)

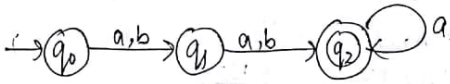
(i) $L(M) = \{aa, bb, ab, ba\}$





(ii) $L(M) = \{aa, bb, ab, ba, aaa - - - - \}$





(iii) $L(M) = \{\in, a, b, ab, ba, aa, bb\}$.

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

(iv) The length of string is even. ($|w| \bmod 2 = 0$)

(v) Length of the string is divisible by 3 ($|w| \bmod 3 = 0$)

Ans) (iv): 2 cases:

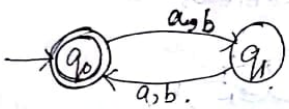
$|w| \bmod 2 = 0$

$|w| \bmod 2 = 1$

n = no. of remainders (here)
i.e. no. of states.

→ in case of divisibility and multiple

So the design is: when no string $|w| = 0$, $0 \bmod 2 = 0$



~~(v) 3 cases:~~

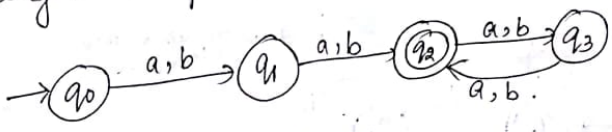
~~$|w| \bmod 3 = 0$~~

~~$|w| \bmod 3 = 1$~~

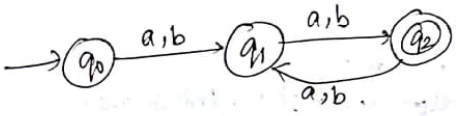
~~$|w| \bmod 3 = 2$~~

$n = 3$

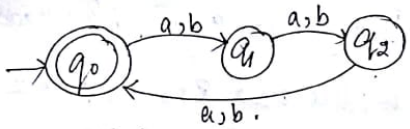
Design a DFA such that strings of even length is accepted except $\in$:-



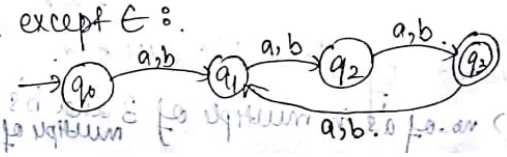
OR.



(v) 3 cases: $|w| \bmod 3 = 0$, $|w| \bmod 3 = 1$, $|w| \bmod 3 = 2$. } $n = 3$.



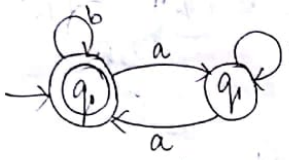
If except $\in$:



Q) Construct a DFA that accepts string where $\Sigma = \{a, b\}$ such that string contains even number of a's :-

Ans) $\{\in, aa, aab, aba, aaaab \ldots\}$

b can be there any no. of times.



Q) Construct a DFA : that accepts a set of all strings over an alphabet set :

$\Sigma = \{a, b\}$ s.t :

i) No. of 'a's are even and no. of 'b's are even.

ii) no. of a's are even & no of b's are odd

iii) no. of a's are odd & no. of b's are even

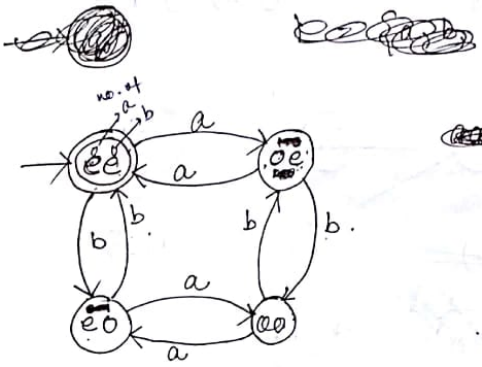
(iv) no. of a's & b's are odd.

(v) no. of a's is multiple of 3 and b's are multiple of 2.

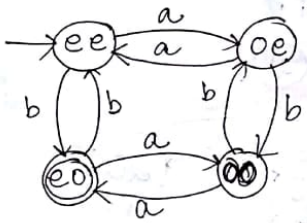
Ans)

(i) $\Sigma = \{a, b\}$

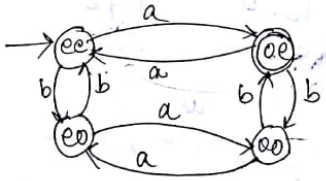
ee → even (a) even (b)



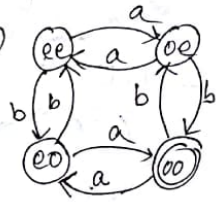
(ii)



(iii)

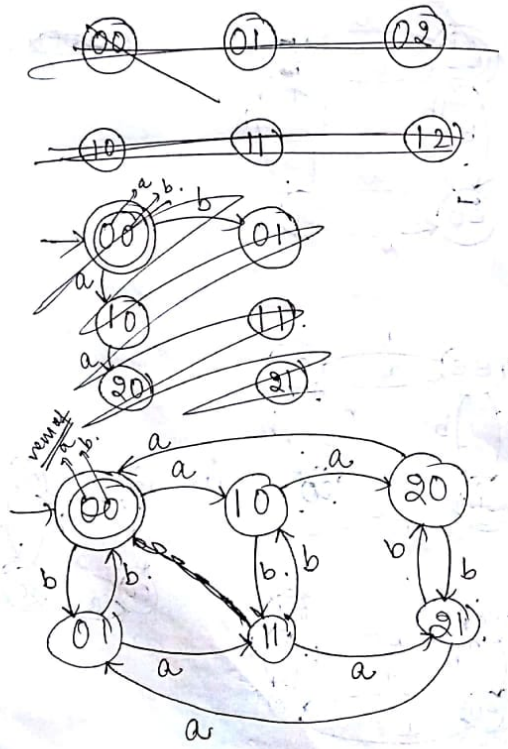


(iv)



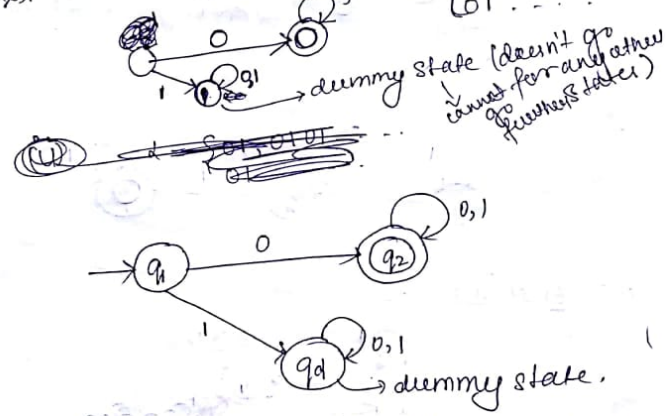
(v) a b div by 3 and b is div by 2.

remainder a → 0 1 2
" b → 0 1.



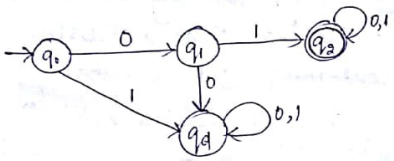
Q Construct a DFA over an alphabet set :-
$\Sigma = \{0, 1\}$ such that
(i) All strings starting with 0
(ii) all strings starting with 01
(iii) containing 0 as a substring.
(iv) Containing 01 as substring.
(v) all string ending with 0.
(vi) all " " " 01
(vii) " " " 011

Ans) (i).

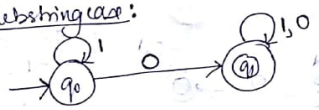


When we deal of starting cases then go for dummy for the not accepted strings.

(ii) $L = \{01, 0101 \ldots, 01 \ldots\}$

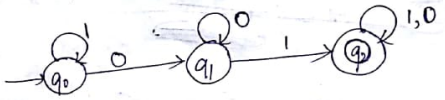


(iii) Substring case:

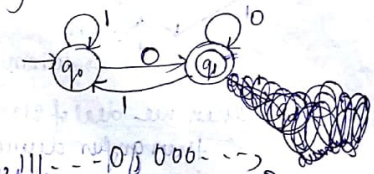


$L = \{0, 10, 01 \ldots, 011 \ldots, 000 \ldots\}$

(iv)

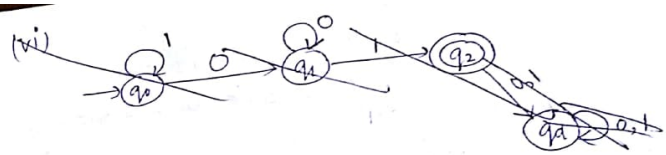


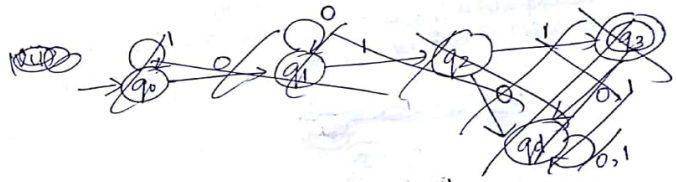
(v) ending case:



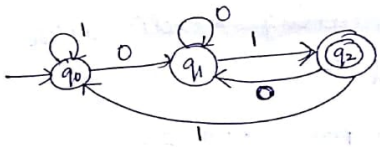
$L = \{0, 111 \ldots 0, 000 \ldots 01 \ldots 0\}$

(vi)

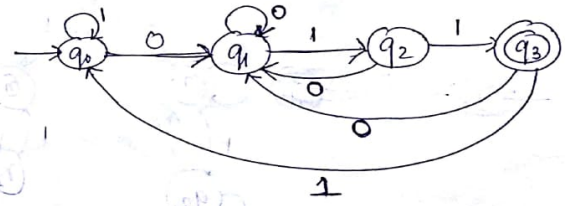




(vi) $L = \{01, 1111 \ldots 01, 000 \ldots 1, 010\}$

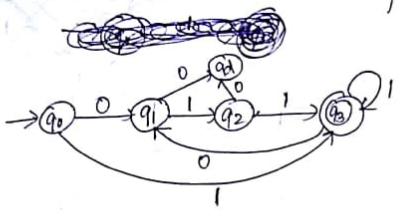


(vii)



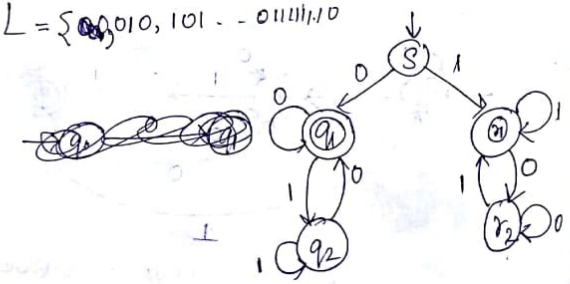
Q) Construct a DFA :-
over the alphabet set $\Sigma = \{0,1\}$ s.t all strings containing 0's and 1's in which every "0" is followed by "11".

Ans) $L = \{011, 0110011, 011111,$
$1, 1111, 1111\ldots\ldots, 1111011111,$
ending with 11, a string with '0' finisher it will start with 11, & can go with any no. of 1's.



Q) All strings starting & ending with same symbol.

$L = \{0, 010, 101 \ldots 0111110\}$



Q) Construct a DFA for the language given below over the alphabet $\Sigma = \{0,1\}$

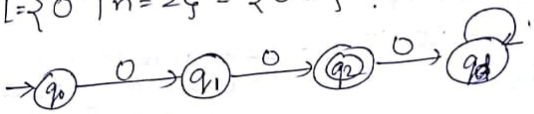
(i) $L = \{0^m 1^n \mid m, n \geq 1\}$.

(ii) $L = \{0^m 1^n \mid m, n \geq 0\}$

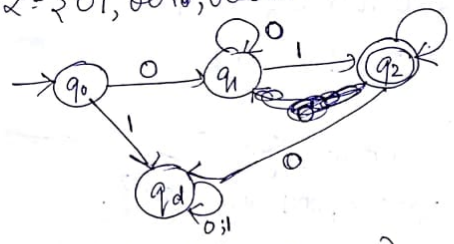
(iii) Over $\Sigma = \{0\}$,
$L = \{0^n \mid n = 2\}$.

Ans)

(iii) $L = \{0^n \mid n = 2\} = \{00\}$.

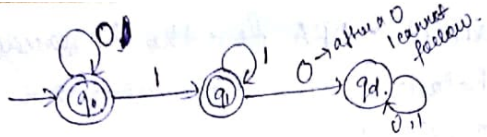


(i) $L = \{01, 0010, 00001$



(ii) $L = \{0^0 1^{10}, 0^{10} 1^0, , \in \ldots\}$

P.T.O



Regular Operations

⇒ operations we make with regular language are Regular Operations.

⇒ Language accepted by a finite state machine are regular language.

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

$L_1 = \{110, 101\}$
$L_2 = \{00, 11\}$ → Both are regular language as we can design a machine for them.

(1) Union:

$L_1 \cup L_2 = \{110, 101\} \cup \{00, 11\}$

$= \{110, 101, 00, 11\}$.

(2) Concatenation:

$L_1 \cdot L_2 = \{110, 101\} \cdot \{00, 11\}$

$= \{11000, 11011, 10100, 10111\}$.

(3) Closure / kleen closure / kleen star:

Σ^* means strings with length 1

$L_1^0 = \{\in\}$

$L_1^1 = \{110, 101\}$

$L_1^2 = L_1^1 \cdot L_1^1 = \{110110, 110101, 101110, 101101\}$

$L_1^3 = L_1^2 \cdot L_1^1 = \{110110, 110101, 101110, 101101\}, \{110, 101\}$

$L_1^4 = \{\ldots\}$ (kleen star)

So $L_1^* = L_1^0 \cup L_1^1 \cup \ldots\ldots$

$L_1^+ = L_1^1 \cup L_1^2 \cup \ldots\ldots$

$= L_1^* - \{\in\}$

(4) Complement:

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

Σ^* = all strings possible out of alphabet set Σ.

Closure property:-

The class of regular languages are closed under :-

(i) Union
(ii) Concatenation
(iii) Complement
(iv) Set difference.
(v) Intersection.
(vi) Kleen star.

Proof

Theorem:- The class of regular language is closed under union. In other words, If two language A_1 and A_2 are regular then $A_1 \cup A_2$ is regular.

Proof: (By method of construction)

Let A_1 is a regular language i.e A_1 is recognised by M_1. Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$

A_2 is a regular language i.e A_2 is recognised by M_2.

Let $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$

Let the DFA be M that recognises $A_1 \cup A_2$ can be constructed as follows:-

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

1. $Q = Q_1 \times Q_2$
   $= \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$
   no. of state in Q in which state in Q_1 multiplied by no. of states in Q_2
2. $\Sigma = \Sigma_1 \cup \Sigma_2$.
3. $\delta = \delta((p, q), a) = (\delta(p, a), \delta(q, a))$
   where $a \in \Sigma$, $p \in Q_1$, $q \in Q_2$
   input
4. $q_0 = (q_1, q_2)$. (Initial state of new machine by combining both).
5. (i) Union.

M recognizes $A_1 \cup A_2$.

So $F = \{(p, q) \mid p \in F_1 \text{ or } q \in F_2\}$.

$F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$.

> if any one is final state then it is final.

(ii) Proof of Intersection (all above 1. → 4 are same)

M recognizes $A_1 \cap A_2$

where $F = \{(p, q) \mid p \in F_1 \text{ and } q \in F_2\}$.

(iii) Proof of set difference:

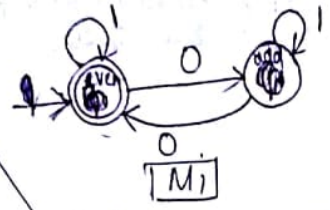
M recognises $A_1 - A_2$.

where $F = \{(p, q) \mid p \in Q_1 \text{ and } q \notin Q_2\}$

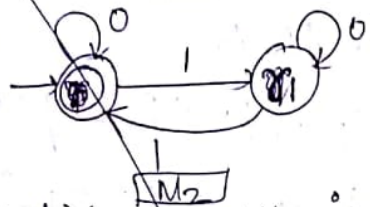
Q) Design a FA that contains (i) even no. of 0's (ii) even no. of 1's.

Combine (i) & (ii).

Ans) (i)



(ii)

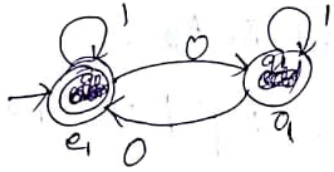


Now combining M_1 and M_2 :-

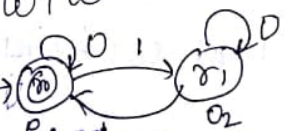
1. $Q = Q_1 \times Q_2$
   $= \{q_0, q_1\} \times \{r_0, r_1\}$
   $= \{(q_0,r_0)(q_0,r_1)(q_1,r_0)(q_1,r_1)\}$
2. $\Sigma = \{0,1\}$

Ans) (i) $A_1 = \{w \mid w$ has even no. of 0's$\}$

M_1:



(ii) $A_2 = \{w \mid w$ has even 1's$\}$



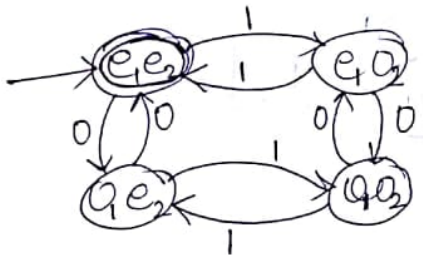
Combining:

① $Q = Q_1 \times Q_2$
$= \{e_1, o_1\} \times \{e_2, o_2\}$
$= \{(e_1e_2)(e_1o_2)(o_1e_2)(o_1o_2)\}$

② $\Sigma = \{0,1\}$

③ $\delta((p,q),a) = (\delta(p,a)\ \delta(q,a))$

④ $q_0 = (ee)$ {initial state}



$\delta(e_1o_2, 0)$
$= o_1o_2$.

$\delta(e_1o_2, 1)$
$= e_1e_2$.

$\delta(o_1o_2, 0)$
$= e_1o_2$

$\delta(o_1o_2, 1)$
$= o_1e_2$

$\delta(o_1e_2, 0)$
$= e_1e_2$

$\delta(o_1e_2, 1)$
$= o_1o_2$

See from M_1, M_2

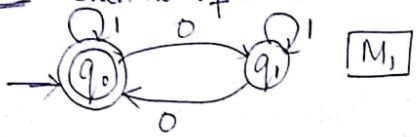
$\delta(e_1e_2, 0)$
$= \{\delta(e_1,0), \delta(e_2,0)\}$
$= \{o_1, e_2\}$

$\delta(e_1e_2, 1)$
$= \{\delta(e_1,1)\ \delta(e_2,1)\}$
$= \{e_1o_2\}$.

⑤ even no. of 0's and even no. of 1's so intersection:
$F = \{(p,q) \mid p \in F_1, q \in F_2\}$
so $F = \{e_1e_2\}$.

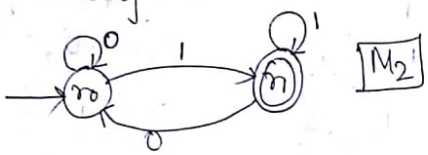
Q) Construct a DFA over the alphabet set $\Sigma(0,1)$ such that all strings containing even no. of 0's or ending with 1.

Ans) even no. of 0's:



M_1

ending with 1:



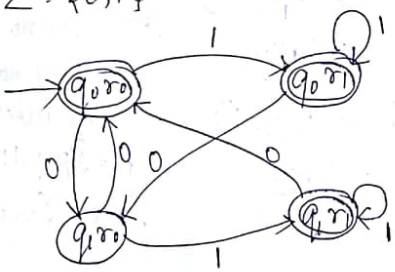
M_2

Combining:

(i) $Q = Q_1 \times Q_2$

$= \{(q_0 r_0)(q_0 r_1)(q_1 r_0)(q_1 r_1)\}$

(ii) $\Sigma = \{0,1\}$



(iii) $\delta(q_0 r_0, 0) = q_1 r_0$ $\quad \delta(q_0 r_0, 1) = q_0 r_1$ $\quad \delta(q_0 r_1, 0) = q_1 r_0$ $\quad \delta(q_0 r_1, 1) = q_0 r_1$

$\delta(q_1 r_1, 0) = q_0 r_0$ $\quad \delta(q_1 r_1, 1) = q_1 r_1$ $\quad \delta(q_1 r_0, 0) = q_0 r_0$

$\delta(q_1 r_0, 1) = q_1 r_1$

(iv) $q_0 = (q_0 r_0)$.

(v) F (union so) $= \{q_0 r_0, q_0 r_1, q_1 r_1\}$.

we can use transition table.

	0	1
q_0	q_1	q_0
q_1	q_0	q_1

	0	1
r_0	r_0	r_1
r_1	r_0	r_1

Q) Construct a DFA for the language $L = \{a^n \mid n \geq 0, \text{ and } n \neq 2\}$. where $\Sigma = \{a\}$

Ans) M_1: $L_1 = \{a^n \mid n \geq 0\}$

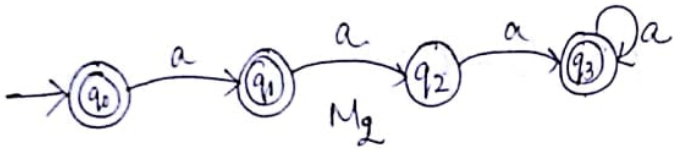
$L_1 = \{\epsilon, a, aa, aaa \ldots\}$

M_1:

M_2: $L_2 = \{a^n \mid n \neq 2\}$. $L_2 = \{\epsilon, a, aaa \ldots\}$.

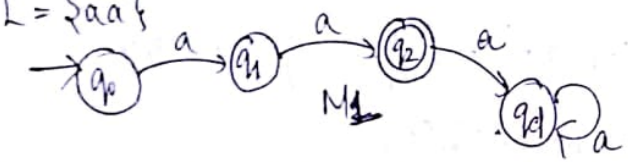
Complement case:

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



DFA for:

$\bar{L} = \{aa\}$

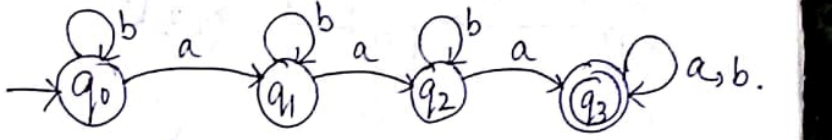


Considering A_1 is a regular language recognised by a machine M_1, its complement $\bar{A}$ is recognised by a machine M_2 where if $M_1 = (Q, \Sigma, \delta, q_0, F)$ then M_2 can be represented as :-

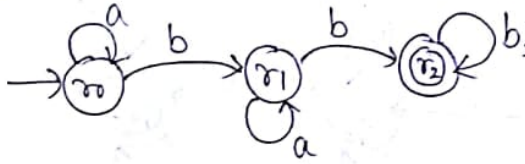
$M_2 = (Q, \Sigma, \delta, q_0, Q - F)$.

Q) Construct a DFA over the alphabet set: $\Sigma = \{a, b\}$ such that:
all strings contain atleast 3 a's and atleast 2 b's.

Ans) M_1: atleast 3 a's.



M_2: atleast 2 b's.



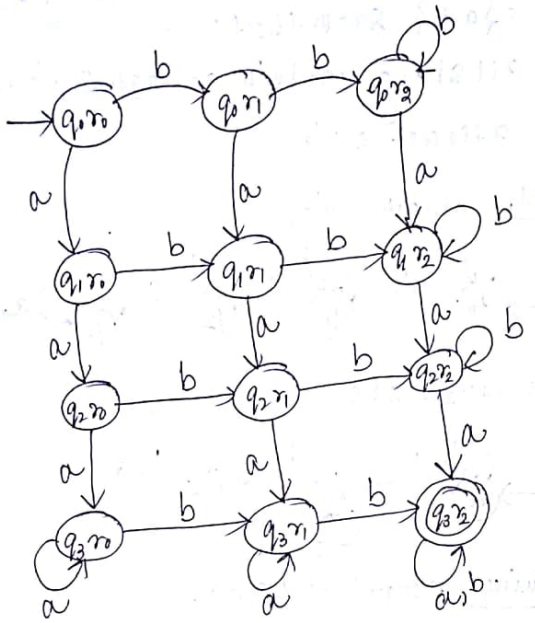
Combining: atleast 3 a's & 2 b's.

(i) $\Sigma = \{a, b\}$ (ii) $Q = \{(q_0, q_1, q_2, q_3) \times (r_0, r_1, r_2)\}$

$= \{(q_0 r_0), (q_0 r_1)(q_0 r_2), (q_1 r_0)(q_1 r_1)(q_1 r_2)$
$(q_2 r_0)(q_2 r_1) q_2 r_2)(q_3 r_0)(q_3 r_1)(q_3 r_2)\}$

(iii) $F = \{ q_3 r_2 \}$.

P. T. O.



	a	b
q_0	q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_2
q_3	q_3	q_3

	a	b
r_0	r_0	r_2
r_1	r_1	r_2
r_2	r_2	r_2

Q) Construct a DFA for the alphabet set $\Sigma = \{a, b\}$ for the language:

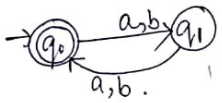
(a) $L_1 = \{w \mid w$ has even length or odd no. of a's$\}$

(b) $L_2 = \{w \mid w$ has even no of a's and each a is followed by atleast one b$\}$

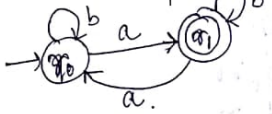
(c) $L_3 = \{w \mid (n_a(w) + 2n_b(w)) \bmod 3 = 0\}$

Ans) (a) $L_1 = \{w \mid w$ has even length or odd no. of a's$\}$

M_1: even length.



M_2: odd no. of a's.



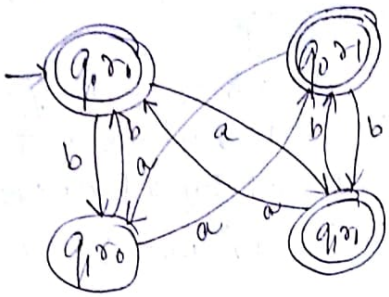
Combining: (i) $\Sigma = \{a, b\}$

(ii) $Q = \{q_0r_0, q_0r_1, q_1r_0, q_1r_1\}$

(iii) $q_0 = q_0r_0$

(iv) $F = \{q_0r_0, q_0r_1, q_1r_1\}$

	a	b
q_0r_0	q_1r_1	q_1r_0
q_1r_1	q_0r_0	q_0r_1
q_1r_0	q_0r_1	q_0r_0



$\delta((q_0r_0), a) = q_1r_1$

$\delta(q_0r_0, b) = q_1r_0$.

$\delta(q_0r_1, a) = q_1r_0$.

$\delta(q_0r_1, b) = q_1r_1$

$\delta(q_1r_0, a) = q_0r_1$

$\delta(q_1r_0, b) = q_0r_0$.

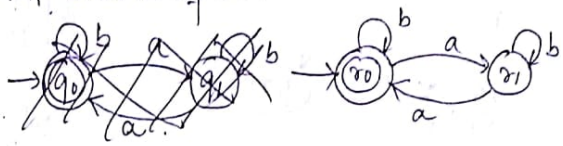
$\delta(q_1r_1, a) = q_0r_0$.

$\delta(q_1r_1, b) = q_0r_1$

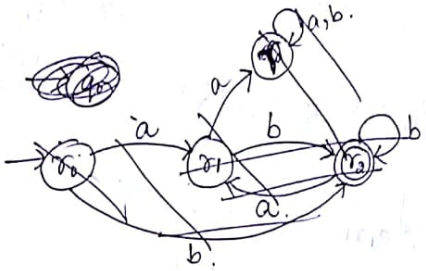
(b) $\alpha_2 = \{w \mid w$ has even no. of a's and each a is followed by atleast one b$\}$.

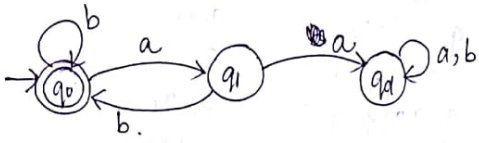
Ans)

M_1: even no. of a's:



M_2: each a is followed by atleast one b.



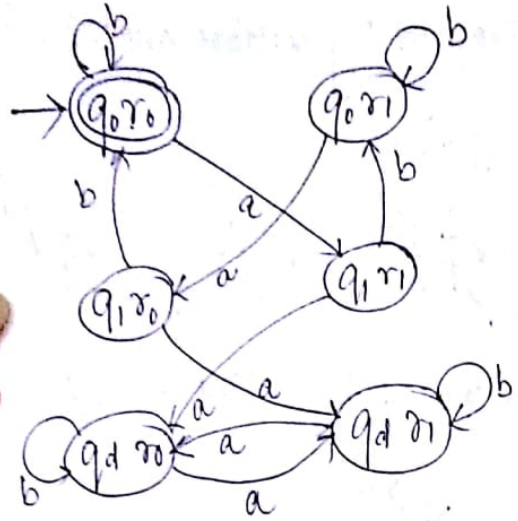


Combining:

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

$q_0 = q_0r_0$.

$Q = \{q_0r_0, q_0r_1, q_1r_0, q_1r_1, q_2r_0, q_2r_1\}$



	a	b
→ q_0r_0*	q_1r_1	q_0r_0
q_0r_1	q_1r_0	q_0r_1
q_1r_0	q_dr_1	q_0r_0
q_1r_1	q_dr_0	q_0r_1
q_dr_0	q_dr_1	q_dr_0
q_dr_1	q_dr_0	q_dr_1

	a	b
→ q_0*	q_1	q_0
q_1	q_d	q_0
q_d	q_d	q_d

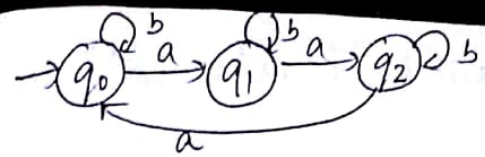
	a	b
→ r_0*	r_1	r_0
r_1	r_0	r_1

In table '*' is final state.
'→' is initial state.

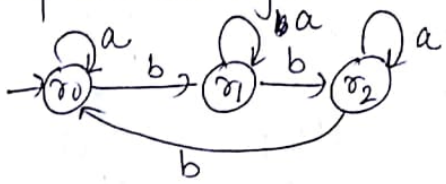
(c) $L = \{w \mid (n_a(w) + 2n_b(w)) \bmod 3 = 0\}$
↳ can be anything

● no. of a's div by 3:

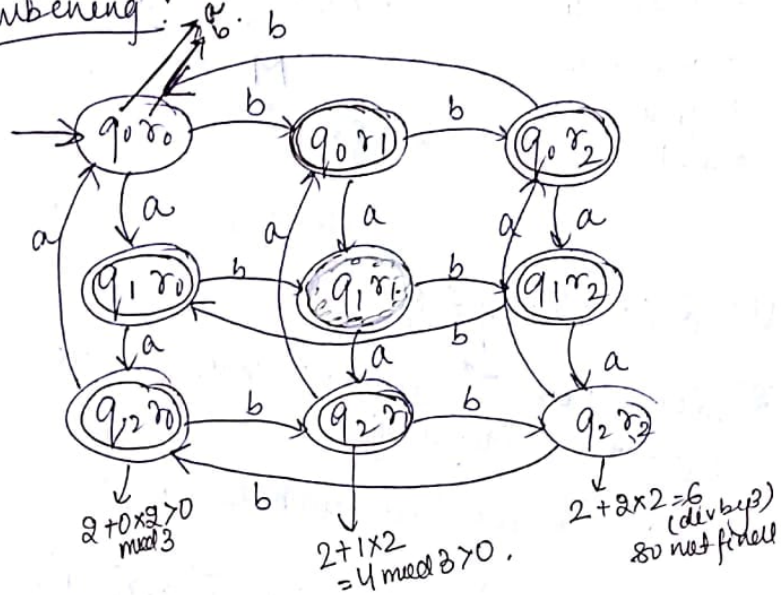
(not considering final state now)



no. of b's div by 3:



Combining:



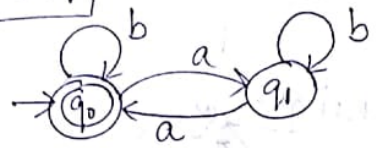
If ques is: $n_a(w) + n_b(w) \bmod 3 = 0$
then final states will be
q_0r_0, q_1r_2, q_2r_1.

If ques is $n_a(w) + n_b(w) \bmod 3 = 0$ and $n_a \neq n_b$
then final states will be
q_1r_2, q_2r_1.

Extended Transition function ($\hat{\delta}$) on String.

Let M be a machine $(Q, \Sigma, \delta, q_0, F)$

$\delta: Q \times \Sigma \to Q$



M

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

$\hat{\delta}: Q \times \Sigma^* \to Q$, $\hat{\delta} = (\delta^*)$.

$q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_0$.

(1) $\hat{\delta}(q, \epsilon) = q \in Q$.

(2) Let $w \in \Sigma^*$ and $w = xa$ where $x \in \Sigma^*$ (Any string). and 'a' is the last symbol, $a \in \Sigma$.

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

$\hat{\delta}(q, b) = \hat{\delta}(q, \epsilon b) = \delta(\hat{\delta}(q, \epsilon), b)$

$= \delta(q, b)$

Ex: $\hat{\delta}(q_0, baab)$

$= \delta(\hat{\delta}(q_0, \underbrace{baa}_{x}), b)$

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

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

$= \delta(\delta(\delta(\delta(\hat{\delta}(q_0, \epsilon), b), a), a), b)$.

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

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

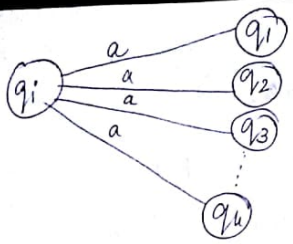
$= \delta(\delta(q_1, a), b)$

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

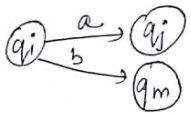
Non Deterministic Finite Automata

* One state with one input may transit to more than one states (NFA)
One state with one input will transit to one state (DFA)
* Mayer mayner give all transitions (NFA)

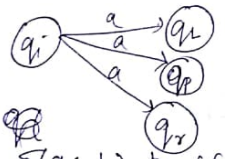
(Deterministic)



(Non deterministic)



(Deterministic)



(Non deterministic)

$\delta(q_i, b) = \phi$ (for transition with input 'b' there is no state or it moves to ϕ state).

* In a Non-Deterministic finite Automate (NDFA/NFA), the transition function is a state and input symbol producing a set of states.

$$\delta : Q \times \Sigma \to P(Q)$$

Formal Def^n:

A NFA is a machine represented with 5 tuples:

$M = (Q, \Sigma, \delta, q_0, f)$. where Q is the set of finite states, Σ is a finite set of alphabets.

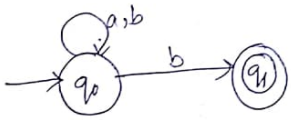
$\delta : Q \times \Sigma \longrightarrow P(Q)$

$q_0 \to$ initial state

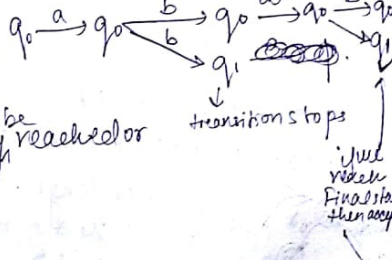
$f \to$ set of final state.

Q) Construct a NFA that ends in 'b' where $\Sigma = \{a, b\}$.

Ans) $\Sigma^* b$



Process a string : $w = abab$

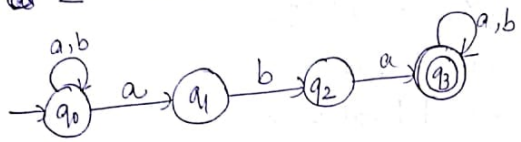


The string may or may not be accepted. So the destination may be reached or may not be reach.

Scanned by CamScanner

Q) Construct a NFA for $\Sigma = \{a, b\}$ such that all strings containing 'aba' as a substring.

Ans) $\Sigma^* aba \Sigma^*$



Write the above in a transition table:

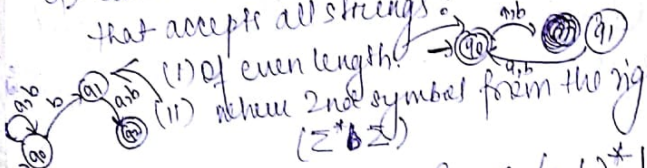
	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	q_0
q_1	ϕ	q_2
q_2	q_3	ϕ
q_3^*	q_3	q_3

Q) Construct a NFA over $\Sigma = \{a, b\}$ that accepts all strings:

(I) Of even length

(II) where 2nd symbol from the right is 'b' ($\Sigma^* b \Sigma$)

(III) Language is: $L = \{ w \in (a,b)^* \mid$ 2nd symbol from righthandside is b$\}$



Equivalence of NFA to DFA

If NFA & DFA can accept the same language then we can say both NFA & DFA machines are equivalent.

No. of states in NFA : k

No. of states in DFA : 2^k (max)

No. of states in DFA >= k and <= 2^k.

Proof: There exist an equivalent DFA for NFA. (To prove)

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA that recognises a language A.

We can construct a DFA, $M = (Q', \Sigma', \delta', q_0', F')$ for recognising the same language.

① $Q' = P(Q) \rightarrow$ The no. of states in DFA is equal to powerset of NFA

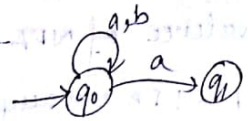
$= 2^Q$.

② $\Sigma' = \Sigma$

③ δ' : for $\delta'(R, a)$ where $R \in Q'$ and $a \in \Sigma$

$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$

Ex: If $\underbrace{q_0 q_1}_{R}$ is a state



$\delta'(\overbrace{(q_0, q_1)}^{R}, a)$

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

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

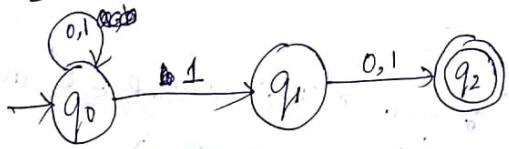
$= \{q_0, q_1\}$

④ $q_0' = q_0$

⑤ $F' = \{R \in Q' \mid R$ contains an accept state of the NFA$\}$

Q) Construct a NFA which accepts all strings that contains 1. Find an equivalent DFA for the NFA with the set $\Sigma = \{0,1\}$ such that 2nd last symbol as 1. Remove all unreachable states & construct the DFA again.

Ans) $\frac{\text{NFA}}{L = \Sigma^* 1 \Sigma}$



DFA: There will be 8 states (2^3)

For NFA:

	0	1
→$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	$\{q_2\}$	$\{q_2\}$
*$\{q_2\}$	ϕ	ϕ

For DFA: (next transition DFA)

	0	1
ϕ	ϕ	ϕ
→$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	$\{q_2\}$	$\{q_2\}$
*$\{q_2\}$	ϕ	ϕ
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
*$\{q_0, q_2\}$	$\{q_0\}$	$\{q_0, q_1\}$
*$\{q_1, q_2\}$	$\{q_2\}$	$\{q_2\}$
*$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$

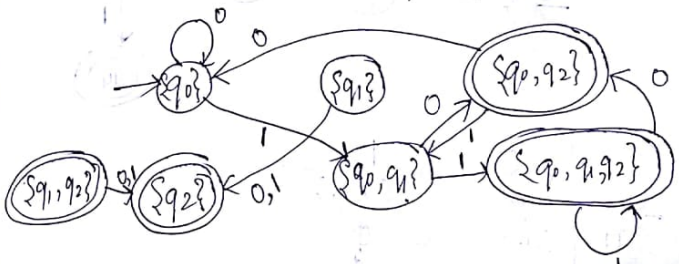
final states

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

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

$= \{q_0\ q_2\}$

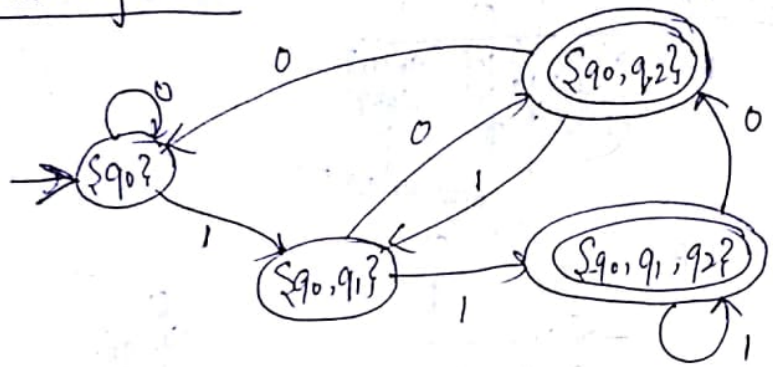
$q_0' = q_0$



The unreachable states are:

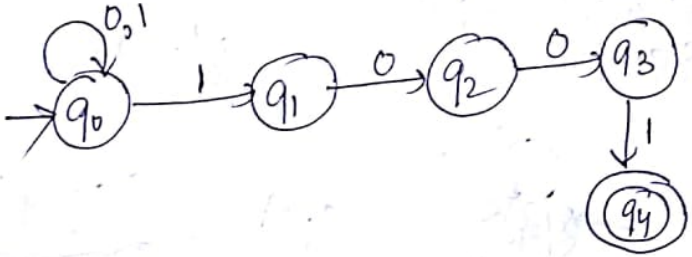
$\{q_1\}, \{q_2\}, \{q_1, q_2\}$

Removing them:



Q) (i) Construct an NFA over $\Sigma = \{0,1\}$ such that all strings ending with '100'.

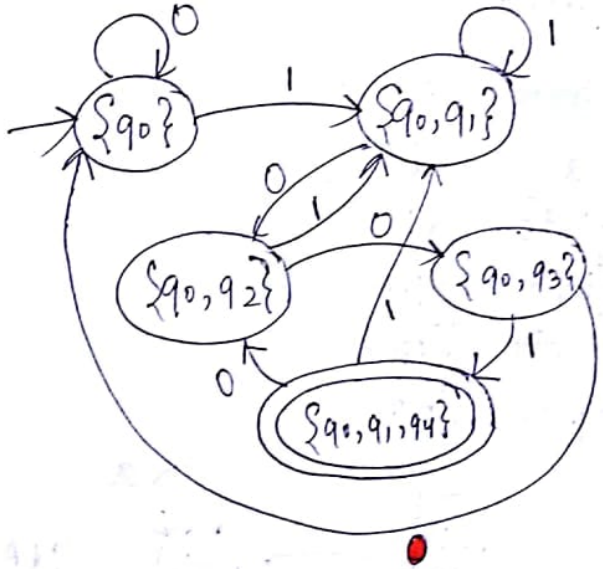
Ans)



	0	1
→ q0	{q0}	{q0,q1}
q1	{q2}	ϕ
q2	{q3}	ϕ
q3	ϕ	{q4}
(q4)	ϕ	ϕ

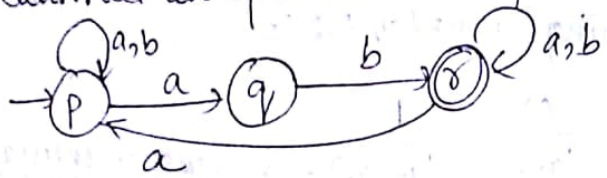
(ii) Construct an equivalent DFA:
(as here we will have 32 states which will be lengthy so we use another method)
we use state that are reachable using Transition table.

	0	1
→ {q0}	{q0}	{q0,q1} → when encountered a new state write it on left.
{q0,q1}	{q0,q2}	{q0,q1}
{q0,q2}	{q0,q3}	{q0,q1}
{q0,q3}	{q0}	{q0,q1,q4}
{q0,q1,q4}	{q0,q2}	{q0,q1} → no new states so don't proceed.



NFA with ε moves

Q1) Construct an equivalent DFA for the following NFA.

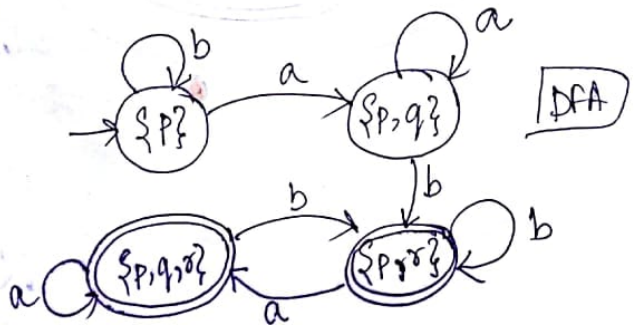


Ans)

	a	b
→ p	{p, q}	{p}
q	ϕ	{r}
(r)	{p, r}	{r}

For DFA transition table:

	a	b
→{p}	{p, q}	{p}
{p, q}	{p, q}	{p, r}
{p, r}	{p, q, r}	{p, r}
{p, q, r}	{p, q, r}	{p, r}



DFA

Q2)

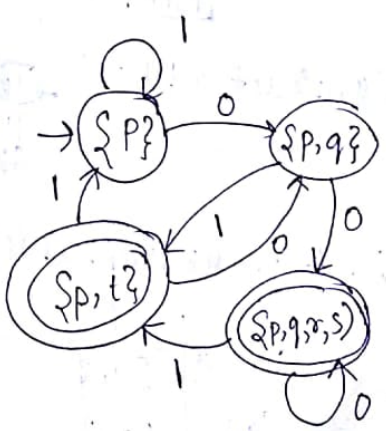
	0	1
→{p}	{p, q}	{p}
{q}	{r, s}	{t}
{r}	{p, r}	{t}
(s)	ϕ	ϕ
(t)	ϕ	ϕ

Construct equivalent DFA.

Ans) DFA transition table:

	0	1
→{p}	{p, q}	{p}
{p, q}	{p, q, r, s}	{p, t}
{p, q, r, s}	{p, q, r, s}	{p, t}
{p, t}	{p, q}	{p}

DFA:

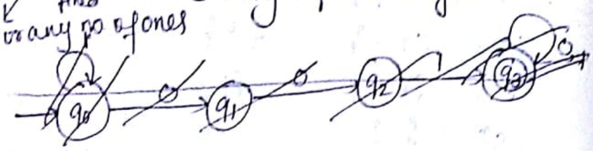


ϕ | t' sps

Q3) (i) $1^*(001^+)^*$ → repeated as * in there.

↓ fixed → no ε string is positive length.

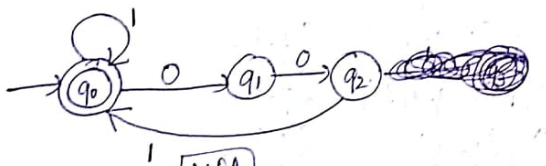
Ans) ε or any no of ones



~~α = {ε, 1, 11, 111 (00(1, 11, 111 ...))}~~

1^*

α = {ε, 111 ... 001111, 111 .... (001 ..)(001 ..)}



NFA

when there is '+' that means a loop.

$r.r^* = r^+$

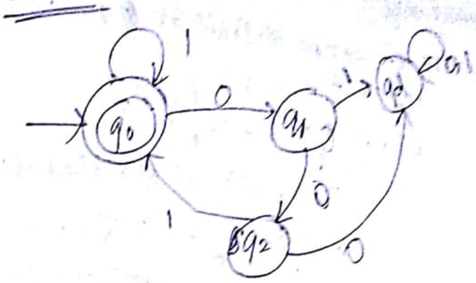
(ii) Construct a DFA:

	0	1
→q0	q1	q0
q1	q2	φ
q2	φ	q0

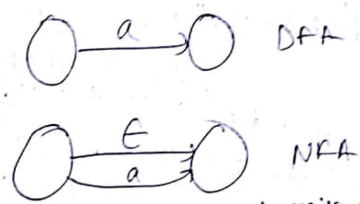
DFA transition table:

	0	1
→{q0}	{q1}	{q0}
{q1}	{q2}	φ
{q2}	φ	{q0}

DFA:



NFA with ε moves :-



in NFA there can be transition between states with ε.

⇒ In NFA-ε the transition function is a state Q: and an input symbol Σ or an empty string produces a set of possible next states.

Mathematically,

$Q \times \Sigma \cup \{\epsilon\} \rightarrow P(Q)$ → Actual Transition fun of NFA

Formal defn:

A NFA-ϵ can be defined as :

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

where Q : finite set of states.

Σ : finite set of alphabets

$\delta : Q \times \Sigma_\epsilon \to P(Q)$ where

$\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$.

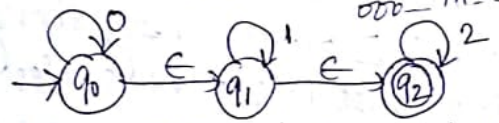
q_0 : initial state $\in Q$

F : Set of final states $\subseteq Q$

Q) Construct an NFA and equivalent DFA over the alphabet Set $\Sigma = \{0,1,2\}$ such that $L = \{0^i 1^j 2^k \mid i,j,k \geq 0\}$

Ans) $L = \{\epsilon, 0, 00, 000, \ldots, 11, 111, \ldots, 22, 222, \ldots, 01, 0011, \ldots, 12, 1122, \ldots, 012, 001122, \ldots\}$

NFA:



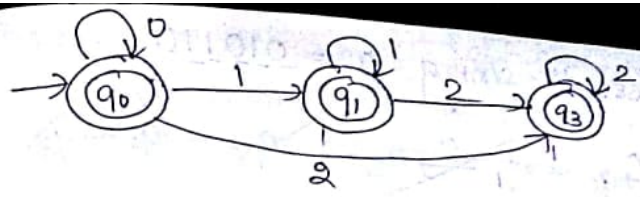
(with ϵ concept)

000 -- ϵ 111 -- ϵ ✓

⇓

000 --- 111 ---

NFA without ϵ concept

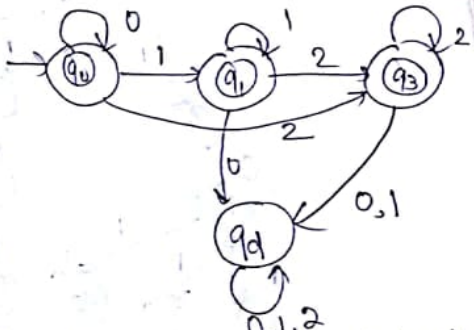


DFA :

Transition table for NFA:

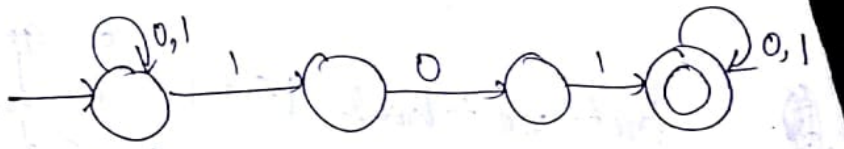
	0	1	2
q0	{q0}		
q1			
q2			

equivalent DFA:

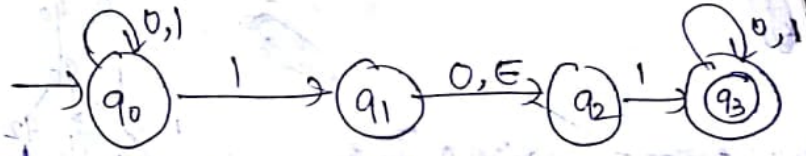


(ii) Q) Construct an NFA-ϵ over the alphabet set $\Sigma = \{0,1\}$ such that all strings containing '101' as a substring or '11' as substring. (i) Design an NFA where 101 is a substring

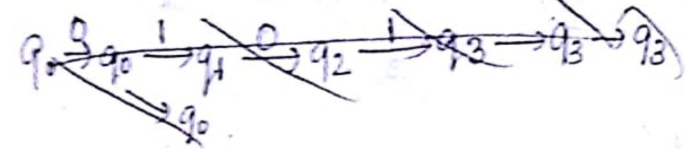
Ans) (i)

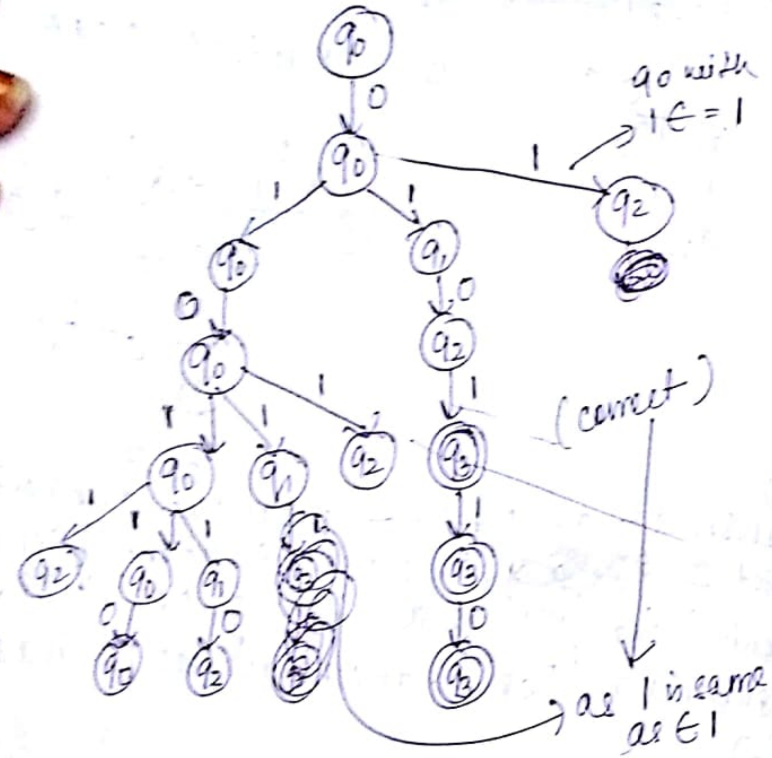


ii)

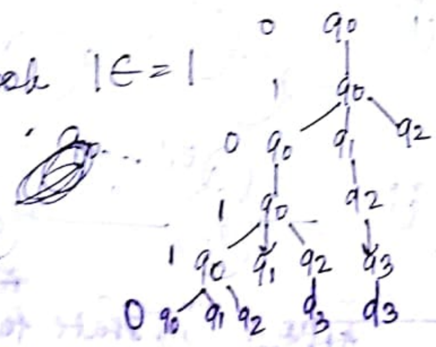


(iii) Process the string $\omega = 010110$.



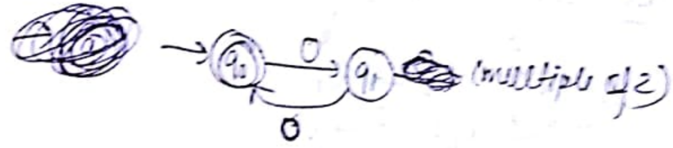


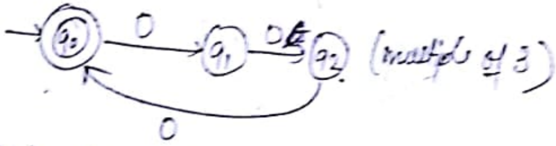
but are to took 1∈ = 1



Q) Design or construct an NFA over the $\Sigma = \{0\}$ such that all strings are multiple of 2 or multiple of 3.

Ans) L = {ε, 00, 0000, 000000, 00000000 . . . .
000, 000000, 000000000 . . . .





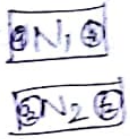
multiple of 2 or 3:

Operations on NFA:

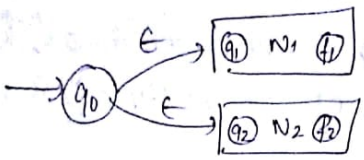
(2nd way): 2 NFA's combined.

If 2 regular languages A_1 and A_2 are recognised by the NFA-∈, N_1 and N_2 respectively then

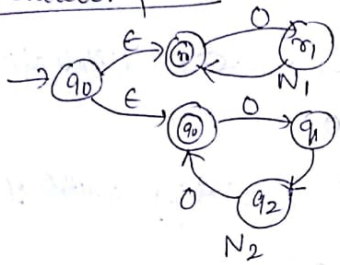
$A_1 \cup A_2$



(i) for union of two NFA's create a initial state for 2 automaton with ∈



Previous ques: (2nd way)



(b) Concatenation : $A_1 . A_2$

For combo initial state of A_1 is initial state of combo & final state of N_2 is final state of combo

1st way: Connect N_1 & N_2 with ϵ

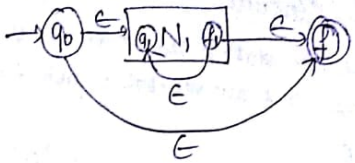
or

2nd way:

combine ~~both~~ final states

(c) A_1^* : (closure)

add initial & final states & connect by using ϵ



q) Construct a DFA by designing the NFA for the following language

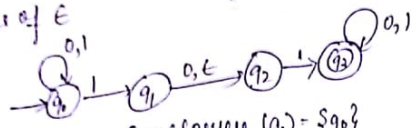
① Set of all strings containing either 101 or 110 as substring.

② Set of all strings such that every '1' is followed immediately by '00'.

③ Set of all strings containing exactly 2 occurrences of '10'.

∈-Closure

∈ of a particular state is the state itself along with the other states that are reachable from the state on a series of ∈



∈-closure $(q_0) = \{q_0\}$
" $(q_1) = \{q_1, q_2\}$
" $(q_2) = \{q_2\}$
" $(q_3) = \{q_3\}$

$\delta(q_0, a) = \in$-closure $(\delta(\in\text{closure}(q), a))$

upto what state transition can q go with input a.

$\delta(q_0, 0) = \in$-closure $(\delta(\in\text{closure}(q_0), 0))$
$= \in$-closure $(\delta(q_0, 0))$
$= \in$-closure (ϕ) q_0)
$= \phi$ $\{q_0\}$.

$\delta(q_0, 1) = \in$-closure $(\delta(\in\text{-closure}(q_0), 1))$
$= \in$-closure $(\delta(q_1, 1))$ ∈-closure $(\delta(q_0, 1))$
$= \in$-closure $\{q_1, q_2\}$ ⇒ ∈-closure $(q_0 q_2)$
$= \{q_1, q_2\}$ ⇒ $\{q_0, q_1, q_2\}$.

	0	1	∈
q_0	ϕ	$q_1 q_0$	ϕ
q_1	q_2	ϕ	q_2
q_2	ϕ	q_3	ϕ
q_3	ϕ_3	ϕ_3	ϕ

Shortcut: make transition table.

use $\Sigma^* a \Sigma^*$

epsilon closure

∈-closure $(q_1) = q_1 q_2$

Let $\delta(q_1, 0)$:

$\Sigma^* 0 \Sigma^*$
$q_1 q_2 \to q_2 \in \phi$
q_2 so $\delta(q_1, 0) = \{q_2\}$

$\delta(q_3, 1) = q_3$

$\delta(q_1, 1)$:

$\Sigma^* 1 \Sigma^*$
$q_1 q_2 \to q_3 \in \phi$
q_3

first find ∈ closure of q_1 then compare transition with 1 then see ∈ of that

so $\delta(q_1, 1) = q_3$

$\delta(q_3, 0) = q_3$

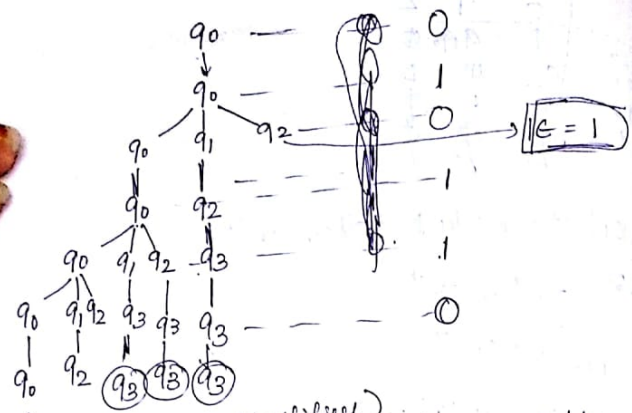
$\Sigma^* 0 \Sigma^*$
$q_3 \to q_3 \to \phi$

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

$\Sigma^* 0 \Sigma^*$

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

$\Sigma^* 1 \Sigma^*$



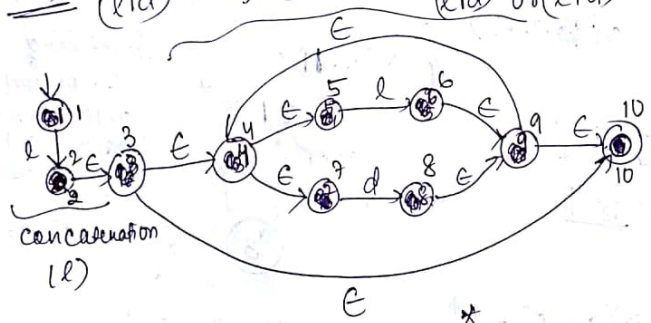
Comp

Q) (Design of C identifiers)
Construct an ϵ-NFA over the alphabet set $\Sigma = \{l, d\}$ for the expression.

$l\,(l|d)^*$ → identifier starts with letter followed by any no. of letter or digit

Ans) $(l|d)^* = \epsilon, (l|d), (l|d)(l|d) \dots$

$(l|d)^*$ or $(l+d)^*$



ϵ

$l\,(l|d)^*$

Find ϵ-closure of all states:

	l	d	ϵ
1	q_1	ϕ	ϕ
2	ϕ	ϕ	q_2
3			
4			
5			
6			
7			
8			
9			
10			

~~ϵ-closure $(q_0) = \{q_0\}$ ϵ-closure $(q_2) = \{q_2, q_3\}$~~

~~$(q_1) = \{q_2, q_3\}$~~

ϵ-closure (1) = {1}
" (2) = {2, 3, 4, 5, 7, 10}
" (3) = {3, 4, 5, 7, 10}
" (4) = {4, 5, 7}
" (5) = {5}
" (6) = {6, 9, 10, 4, 5, 7}
" (7) = {7}
" (8) = {8, 9, 4, 5, 7, 10}.
" (9) = {10, 4, 5, 7, 9}.
" (10) = {10}

$(l|\_)(l|d|\_)^*$
↑ logic of identifier

Theorem:

For every NFA-ϵ there exists an equivalent NFA without ϵ.

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA-ϵ recognising some language A.

NFA without ϵ, $N' = (Q', \Sigma', \delta', q_0', F')$ can be constructed for recognising the same language A.

① $Q' = Q$

② $\Sigma' = \Sigma$

③ For any $q \in Q$ and $a \in \Sigma$

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

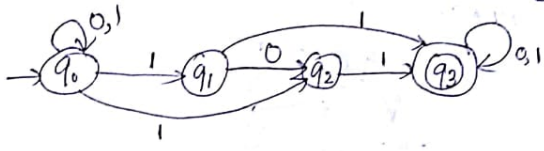
④ $q_0' = q_0$

⑤ $F' = \begin{cases} F \cup \{q\}, & \text{if } \epsilon\text{-closure}(q) \text{ contains any accepting state of NFA-}\epsilon \\ F, & \text{otherwise.} \end{cases}$

Q) 

Design NFA without ϵ

Ans) Previously δ' calculated for this ex.



Theorem:

For every NFA-ϵ there exist an equivalent DFA.

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA-ϵ that recognises some language A.

We can construct a DFA, $M = (Q', \Sigma', \delta', q_0', F')$ to recognise the same language A.

① $Q' = P(Q)$.

② $\Sigma' = \Sigma$

~~③ $\delta'(q, a) = \epsilon\text{-closure}(\delta(q, a))$~~

④ $q_0' = \epsilon\text{-closure}(q_0)$

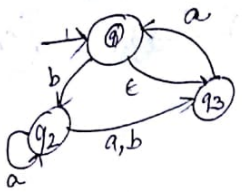
⑤ $F' = F \cup R$, R contains atleast one state which is final from NFA

③ $\delta'(R, a) = \epsilon\text{-closure}\left(\bigcup_{q \in R} \delta(q, a)\right)$

↳ If $R = (q_1, q_2)$ then this will be

ϵ-closure of (q_1, a) $\cup$ ϵ-closure of (q_2, a)

Q) Construct an equivalent DFA for the following NFA :-



Ans) For NFA:

	a	b	ε
→ q_1	ϕ	$\{q_2\}$	$\{q_3\}$
q_2	$\{q_2, q_3\}$	$\{q_3\}$	ϕ
q_3	$\{q_1\}$	ϕ	ϕ

ϵ-Closure$(q_1) = \{q_1, q_3\}$
" $(q_2) = \{q_2\}$
" $(q_3) = \{q_3\}$

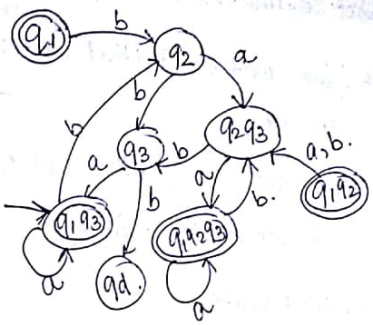
On DFA:

	a	b
ϕ	ϕ	ϕ
q_1	ϕ	q_2
q_2	q_2q_3	q_3
q_3	q_1q_2	ϕ → dummy state
q_1q_2	q_2q_3	q_2q_3
q_1q_3	q_1q_3	q_2
q_2q_3	$q_1q_2q_3$	q_3
$q_1q_2q_3$	$q_1q_2q_3$	q_2q_3

$\delta'(q_1, a)$
$= \epsilon$-closure $(\delta(q_1, a))$
$= \epsilon$-Closure (ϕ)
$= \phi$

$\delta'(q_3, a)$
$= \epsilon$-closure (q_1)
$= q_1q_2$

$\delta'((q_1q_2), a)$
$= \epsilon$-closure $(\delta(q_1, a))$
$\cup \epsilon$-Closure $(\delta(q_2, a))$
$= \phi \cup q_2q_3 = q_2q_3$

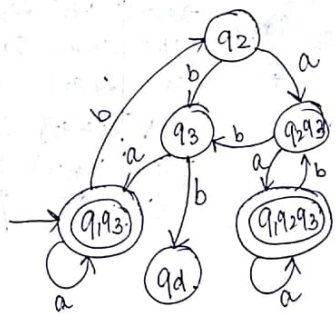


Remove the unreachable states :-

In transition table find states that are not there in right hand side:

q_1, q_1q_2 are not there

these are unreachable states.



Using subset construction method (DFA from NFA)

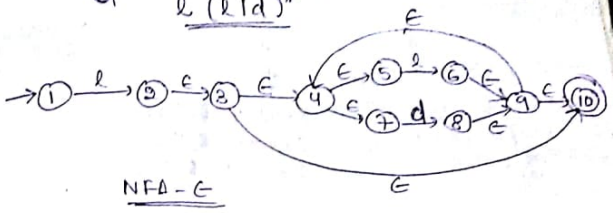
Step1:
From ∈NFA find out the initial states:
Find its ∈-closure.
In Previous ex: q_1 is initial state of NFA-∈
∈-closure(q1) = {q1, q3}
Start with that state
Step2: Apply the transition.
Step3: Then add states that are new & apply transitions

	a	b
→ {q1,q3}	{q1,q3}	{q2}
{q2}	{q2,q3}	{q3}
{q3}	q1	ϕ

	a	b
{q1,q3}	{q1,q3}	{q2}
{q2}	{q2,q3}	{q3}
{q3}	{q1,q3}	ϕ
{q2,q3}	{q1,q2,q3}	{q3}
{q1,q2,q3}	{q1,q2,q3}	{q2,q3}
ϕ	ϕ	ϕ

Q) Find out the DFA of the ∈-NFA

l (l|d)*



NFA-∈

Ans) Initial state: ∈-Closure {1} = {1}

DFA transition table:-

	l	d
→ {1}	{2}	ϕ
{2}	{3}	ϕ
{3}	{4,10}	ϕ
{4,10}	{5,7}	ϕ
{5,7}	{6}	{8}
{6}	ϕ	{9}
{8}	ϕ	{9}
{9}	{4,10}	ϕ

δ'(1,l) = ∈-closure(2)

NFA transition table

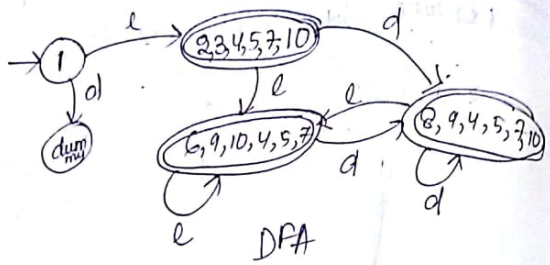
	l	d	∈
→1	{2}	ϕ	ϕ
2	ϕ	ϕ	{3}
3	ϕ	ϕ	{4,10}
4	ϕ	ϕ	{5,7}
5	{6}	ϕ	ϕ
6	ϕ	ϕ	{9}
7	ϕ	{8}	ϕ
8	ϕ	ϕ	{9}
9	ϕ	ϕ	{4,10}
10	ϕ	ϕ	ϕ

DFA transition table

	l	d
→{1}	{2345710}	ϕ

Q: DFA transition table

	a	b
→ {1}	{2,3,4,5,7,10}	ϕ
{2,3,4,5,7,10}	{6,9,10,4,5,7}	{8,9,4,5,7,10}
{6,9,10,4,5,7}	{6,9,10,4,5,7}	{8,9,4,5,7,10}
{8,9,4,5,7,10}	{6,9,10,4,5,7}	{8,9,4,5,7,10}



DFA

Regular Expression:

The declarative way of representing a regular language.

FA - accepts → Regular language

i) we can write a regular expression for any FA

ii) we can design a FA (Finite or Infinite language)

Regular operators:

→ union (∪, +, |) eg: (a ∪ b) (is a regular expression)

→ Concatenation (·) eg: ab or (a ∪ b)aa.

→ Star (*) eg: $(a \cup b)^*$, a^* (repeated again & again)

Eg: Find the regular expression for the set of strings over the alphabet set: Σ = {a, b}

(i) where the length of the string is exactly 2

Ans) $\alpha = \{aa, ab, bb, ba\}$

aa + ab + ba + bb

⇒ a(a+b) + b(a+b)

⇒ (a+b)(a+b) → regular expression

Directly:

(a|b)(a|b)

↑ ↑

space

(a+b)(a+b)

1st place can be a or b

2nd place also.

Q: (ii) length of the string is exactly 3.

(iii) length of string is atmost 2.

(iv) length of string is atleast 3.

(v) Strings containing any no. of a's or any no. of b's

(vi) containing all possible strings. (any length)

Ans)

(i) $(a+b)(a+b)(a+b)$

Reg exp: $(a+b)(a+b)(a+b)$

(iii) ~~$(a+b)^*(a+b)^*$~~.

$L = \{\epsilon, a, b, aa, bb, ab, ba\}$ → by eq" method

ϵ b, b ϵ → length 1

ϵ a, a ϵ → length 0

$\epsilon + a + b + aa + bb + ab + ba$

$\Rightarrow \epsilon + a + b +$

$(a+b)$ $(a+b)$

So $(\epsilon + a + b)(\epsilon + a + b)$

regular exp.

(iv) $(a \cup b)(a \cup b)(a \cup b)(a \cup b)^*$

reg exp. $\Downarrow$ $(a \cup b)(a \cup b)(a \cup b)^+$

$L = \{aaa, aab, aba, \ldots, abb, baa, bab, bba, bbb, aaaa, aaab, aaba, aaaa, aaaabb \ldots\}$

(v) $(a^* \cup b^*)^*$ $L = \{\epsilon, a, b, ab \ldots\}$

reg exp.

(vi) $(a+b)^*$.

$a^* = \{\epsilon, a, aa \ldots\}$

$aa^* = \{a, aa, aaa \ldots\}$

$= a^+$

$\Rightarrow$ If empty string should be represented by a regular exp: ϵ.

$\Rightarrow$ If there is no language accepted so: ϕ

① If a language is finite then the regular expression is the union of all strings present in the language.

Eg: $L = \{aa, bba\}$ ← finite language

Reg exp: $aa + bba$ or $aa \cup bba$

② If a language is finite then the language is a regular language and there must be a regular expression existing for the language.

③ If a language is infinite and we can draw a regular expression to represent it then the language is regular.

Eg: $\{a^m b^n \mid n, m \geq 0\}$ — L1

$a^* b^*$

regexp hence L1 is regular

But:

$\{a^n b^n \mid n \geq 0\}$ — L2

→ as we need equal no of a's & b's.

can't design a reg exp so L2 is not regular lang.

④ For a particular regular language there must be different regular expressions.

$\Sigma = \{aa, bb, ab, ba\}$

$aa + bb + ab + ba$

or

$(a+b)(a+b)$

or

$a(a+b) + b(a+b)$

} more than 1 ans.

Q) Design a regular exp for the language:

(i) $L = \{11, 1111, 111111, \ldots\}$

Ans)

$11(11)^* \rightarrow$ regexp.

or

$(11)^+ \rightarrow$ reg exp

(ii) L = set of all strings containing exactly 2 a's.

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

Ans) $L = \{aa, bbaa, aba, aab \ldots\}$

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

(iii) containing atmost 2 a's.

$b^* + b^* a b^* + b^* a b^* a b^*$

or

$b^* a b^* a b^*$ also $b^* \in b^* \in b^* \rightarrow$ same as b^*

$b^* \in b^* a b^* \rightarrow$ same as $b^* a b^*$ so, $\boxed{b^* (a+\epsilon) b^* (a+\epsilon) b^*}$

$b^* a b^* \in b^*$

(iv) Strings ending with aa.

$(a \cup b)^* (aa)$

(v) Strings containing even no. of a's.

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

(vi) Strings starting & ending with same symbol

$\alpha = \{\in, a, b, a\overset{a}{\underset{abb}{b}}a, b \equiv b \ldots\}$

$b + a + a(a \cup b)^* a + b(a \cup b)^* b + \in$

(vii) Strings containing a substring ab.

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

Q) (i) containing a substring either ab or bb.

(ii) Containing a substrings ab and bb

Ans) (i)

~~$(a+b)^* ab (a+b)^* + (a+b)^* bb (a+b)^*$~~

$[(a+b)^* (ab+bb)(a+b)^*]$ correct

(ii)

~~$[(a+b)^* ab (a+b)^*] . [(a+b)^* bb (a+b)^*]$~~.

~~$[(a+b)^* ab . bb (a+b)^*]$~~

~~$[(a+b)^* (ab+bb)(a+b)^*]$~~

(ii) $[(a+b)^* ab (a+b)^* bb (a+b)^*] +$

$[(a+b)^* bb (a+b)^* ab (a+b)^*]$

$\Rightarrow [(a+b)^* (ab(a+b)^* bb + bb(a+b)^* ab)(a+b)^*]$

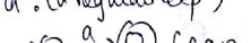
Formal def" of regular exp:

R is said to be a regular expression.

If R is:

(1) a for some symbol a in alphabet Σ.

Ex: a : (a regular exp)

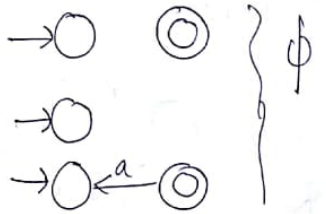


(can be expressed by a FA)

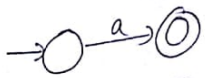
regular exp as we can design a FA.

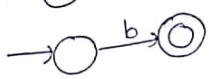
(2) $\in$ is an empty string.

(3) ϕ (not a language) (accepts nothing)

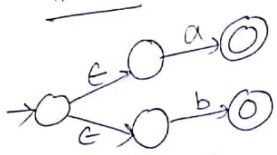


(4) $(R_1 \cup R_2)$ where R_1 and R_2 are 2 regular expressions.

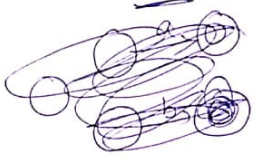
R_1 : 

R_2 : 

$R_1 \cup R_2$:

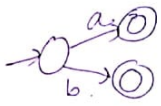


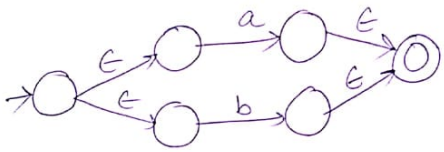
or



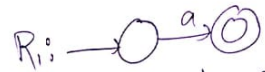
or

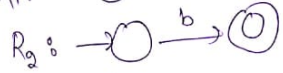
union without ϵ



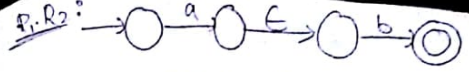


(5) $(R_1 . R_2)$ where R_1, R_2 are regular expressions.

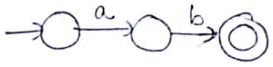
R_1 : 

R_2 : 

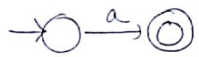
$R_1 . R_2$:

$R_1 . R_2$: 

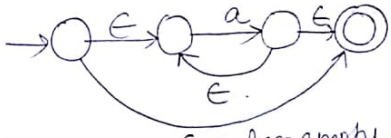
or (without ϵ)



(6) $(R_1)^*$ is a regular exp of R_1 is a regular exp.

R_1 : 

R_1^* :

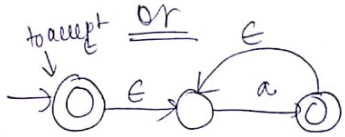


ϵ ← for empty string

R_1^* : or a^*

to accept

or



Equivalence of Regular Expression and Finite Automata

Note:

If R is a regular expression:

$L(R \cup \phi) = R$ (language)

$L(R\phi) = \phi$

ϕ here is behaving as 0 in mathematics.

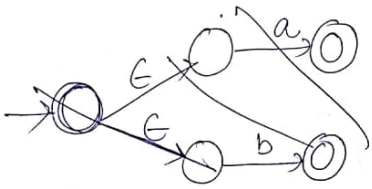
$R\epsilon = R = \epsilon R$

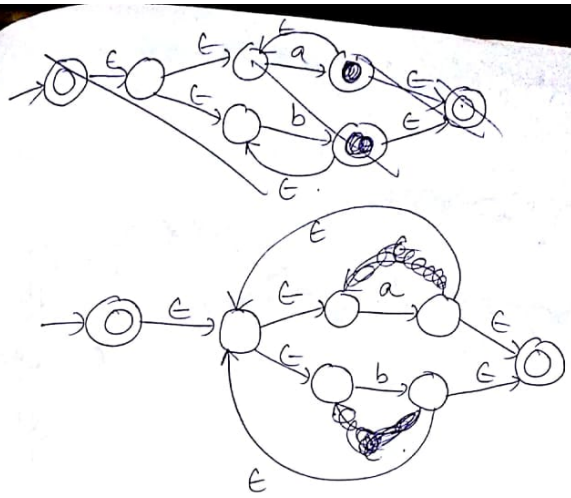
$\phi^* = \epsilon$ (* operation on ϕ gives a string)

$\epsilon^* = \epsilon$

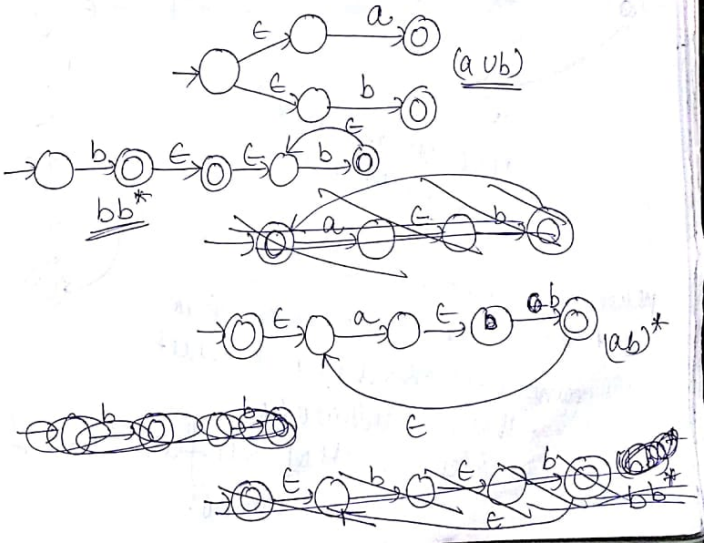
Q) Construct a finite automata (NFA-ϵ) for the regular expression: (i) $(a \cup b)^*$.

Ans)

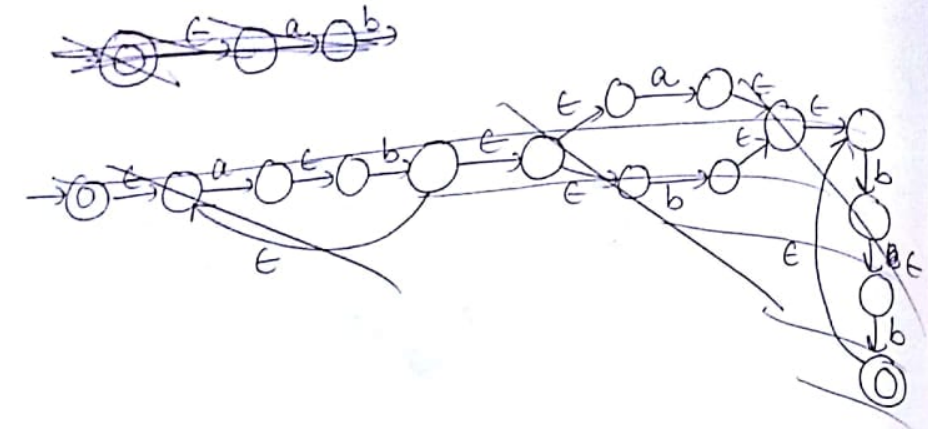


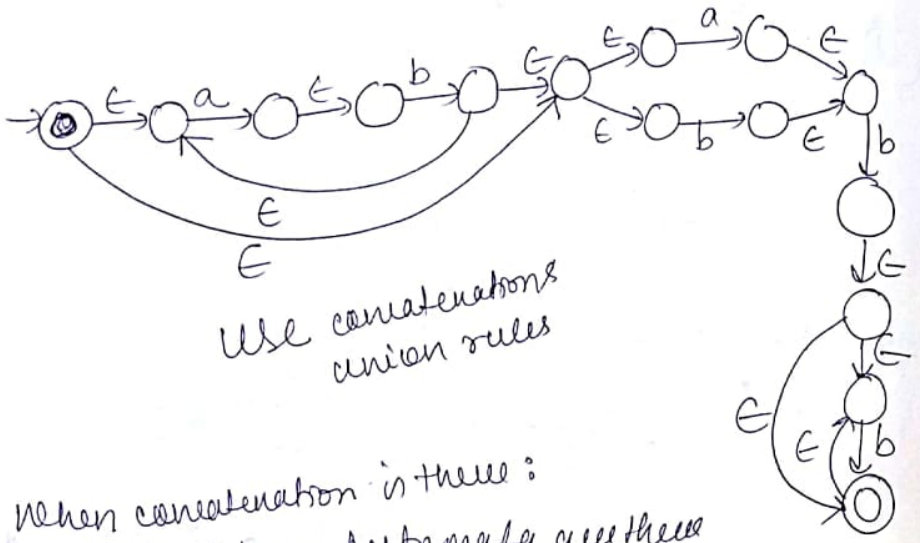


(ii) $(ab)^*(a \cup b)bb^*$



Q: $(ab)^* (a \cup b)(bb)^*$





Use concatenation & union rules

When concatenation is there:
A & B two Automata are there
remove all finals of A & connect them to initial of B.
& final state of B is final state of the combo.

Q1) Write the regular expression over the alphabet set $\Sigma = \{0,1\}$ for the lang. containing string starting with 0 & ending with any no. of 1's.

Q2) write down the reg. exp. over $\Sigma = \{l, d, -\}$ for the C lang. identifier where L → letter, d → digit, _ → underscore

l_d, _ld, ld_, _dl } allowed
d_l, dl_ } not allowed

Ans) (2). $(l + \_)^* . (l + \_ + d)^*$

↓ regular exp.

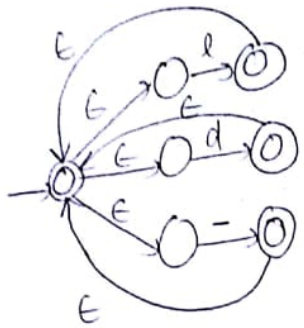
(1). $L = \{0, 01, 011 \ldots, \ldots\}$

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

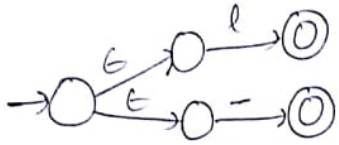
starts with a 0. ending with any no. of 1's.

Design NFA-ϵ for above:

$(l \cup \_)(l \cup d \cup \_)^*$

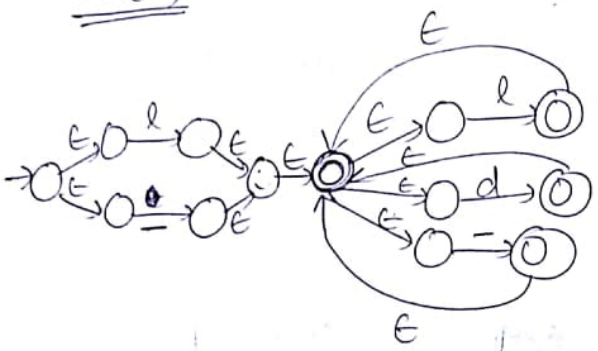


$(l \cup d \cup \_)^*$



$(l \cup \_)$

Now,



Q) Write a regular expression for the language containing even no. of 'a', odd no. of b and not containing a substring ab? Design a DFA for this. (In 5 states).

Ans) $L = \{ bbbaa, baa, b, \ldots \}$

bbbbbaa

$V = \{ b, baa, bb\,baa, bbbbbaa, \ldots \}$

$(bb)^* b(aa)^*$ or $b(bb)^*(aa)^*$

↳ regular expression.

Q) Write the regular expression over the alphabet set $\Sigma = \{0, 1\}$ s.t for strings containing 2 ones not separated by odd no. of elements?

Ans) ~~$L = \{ \epsilon, 11, 0,000 \ldots 1, \ldots \}$~~

$L = \{ \epsilon$
$0, 00, 000, \ldots$
$1, 11, 011, 0011 \ldots \}$

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

Conversion of DFA to Regular Expression.

Process to convert Reg. exp to DFA

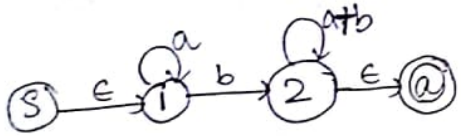
Regular expression → ε-NFA → DFA

DFA → GNFA → Regular expression (through state reduction method)

(Generalised NFA)

DFA to GNFA :-

(2-state DFA), $\Sigma = \{a, b\}$.



(4state GNFA)

To the existing DFA connect a start & accept state (There will be no incoming edge to start state & no outgoing edges from accept state)

In GNFA a,b all are regular expressions

In DFA these are called symbols.

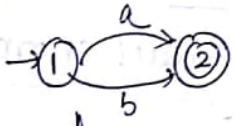
GNFA is the simple NFA where the transition arrow may have regular expressions as its labels, not the symbols from the alphabet.

conditions: • There must be no incoming arrow to the start state, but it has arrows going to every other state (s to 2 connected by ε)

• There is a single accept state. and no outgoing arrows from it.

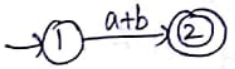
• Except the start & accept state all other states are connected to each other. and to itself also.

If in DFA:



↓

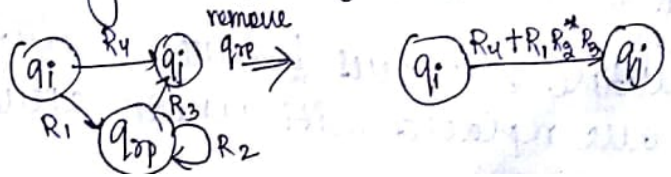
GNFA:



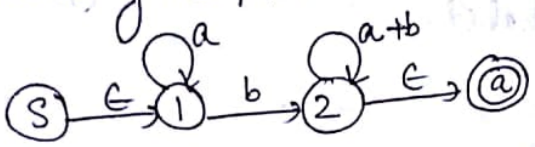
GNFA to Regular exp (by state reduction)

preliminary

Principle

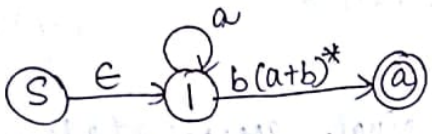


Previously the GNFA:



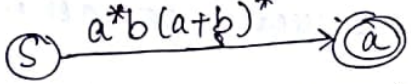
(4 state GNFA)

⇓ reducing (removing 2).



(3 state GNFA)

⇓ eliminating state 1



(2 state GNFA)

hence regular expression is:

$a^* b(a+b)^*$ of DFA

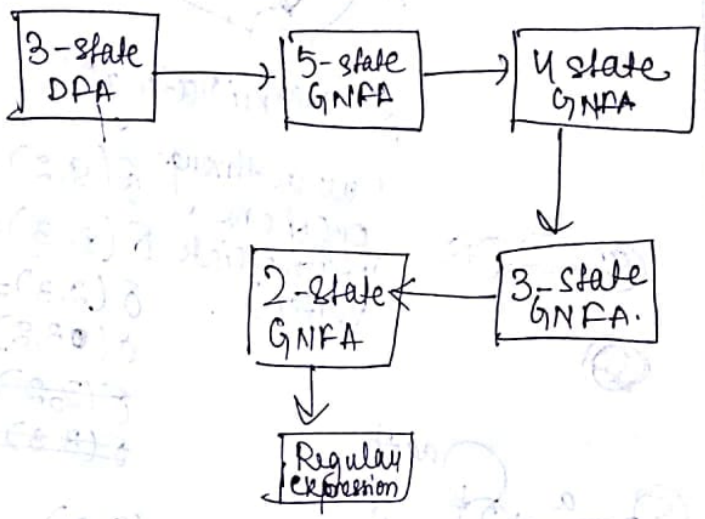
Stages to convert DFA to regular expression:

(I) DFA to GNFA.

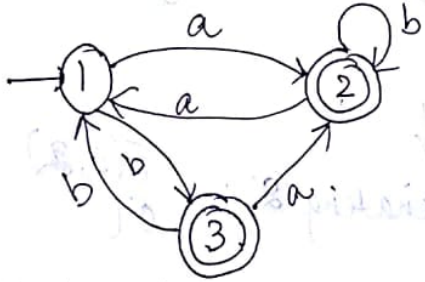
- Add a start state with an ε-arrow to the start state of DFA.
- Add an accept state with an ε arrow from the accept states of DFA.
- Multiple arrows between 2 states are replaced with union operation.
- Add arrows with labels as ϕ where there is no arrow.

(II) GNFA to Regular expression.

Ex:

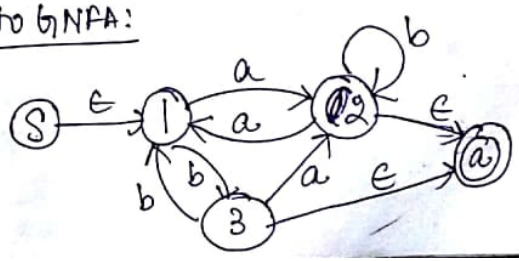


Ex:

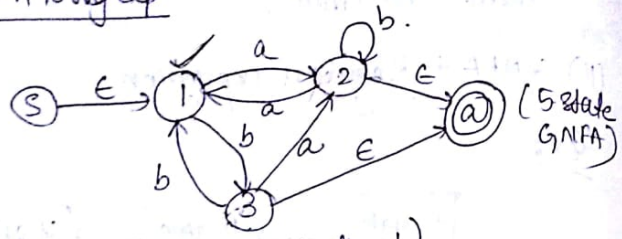


DFA to reg exp:

Ans) DFA to GNFA:



Q. GNFA to reg exp:



(5 state GNFA)

⇓ (eliminating 1).

(we go through only one intermediate vertex).

$\delta(S,2) = \epsilon a = a$

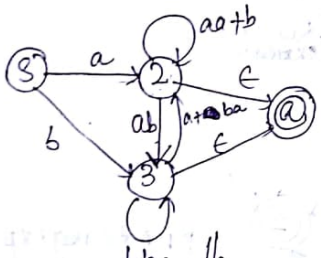
$\delta(S,3) = \epsilon b = b$

$\delta(2,3) = ab$

$\delta(3,2) = ba + a$

~~$\delta(2,2) = baab$~~

~~$\delta(3,3) = aab$~~.



$\delta(2,2) = aa + b$

$\delta(3,3) = bb$

⇓

eliminating ②: $\delta(S,$ ②$)$

$\delta($

Formal Def^n of GNFA

A GNFA or generalised NFA can be defined as a 5-tuple $(Q, \Sigma, \delta, q_{start}, q_{accept})$ where

Q = set of states

Σ = is input alphabet.

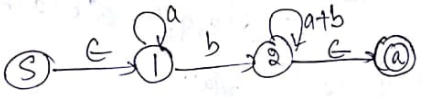
δ = transition function can be defined as:

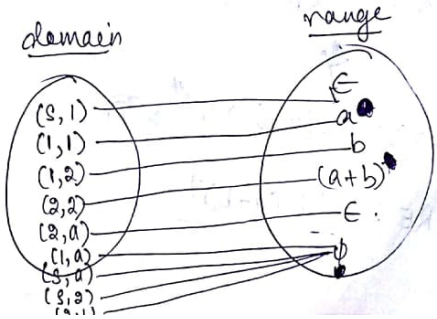
$\delta : (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \to R$

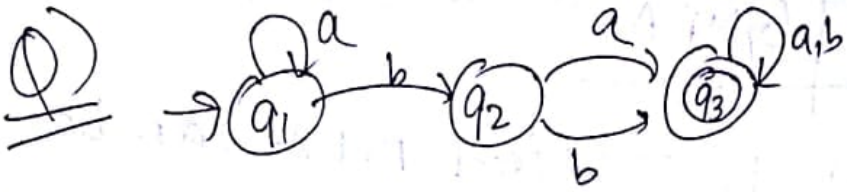
where R is set of regular expressions over Σ.

q_{start} = start state. q_{accept} = accept state.

⇒



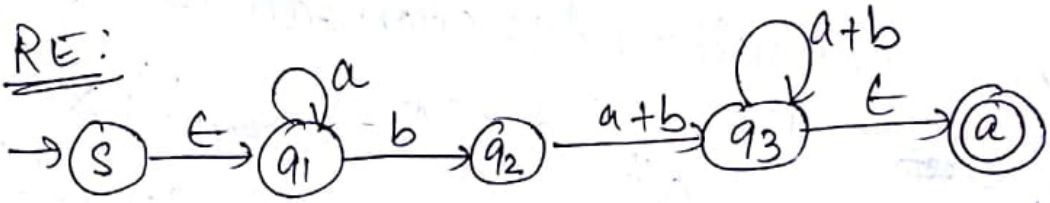


Q. Q) 

Find RE.

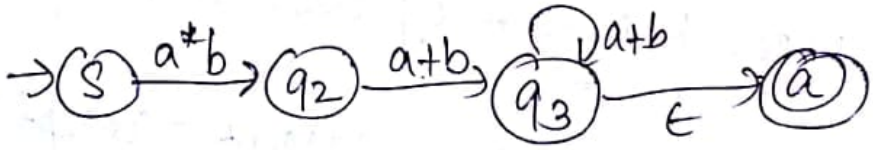
Ans) NFA:

RE:



⇓ reducing q_1

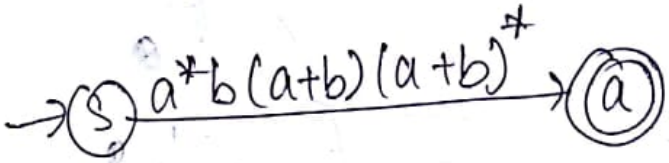
$\delta(S, q_2) = a^* b.$



⇓ reducing q_2

$\delta(S, q_3) = a^* b\,(a+b)$

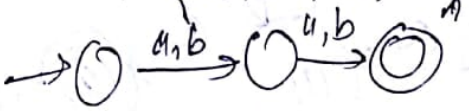
⇓



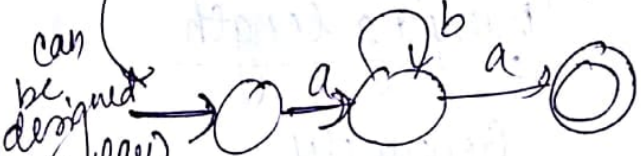
So RE : $a^* b\,(a+b)^+$

Non Regular languages

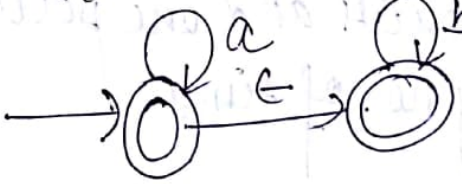
Language
- non regular.
- regular
  - Finite
    - ↓ Regular.
    - $L = \{ab, ba, bb, aa\}$



  - Infinite.
    - $L = \{aa, aba, abba, \ldots\}$
    - can be designed (regular) infinite



$L = \{a^n b^m / n, m \geq 0\}$ { regular }



$L = \{a^n b^n / n \geq 0\}$.

(can't design a machine for this language)

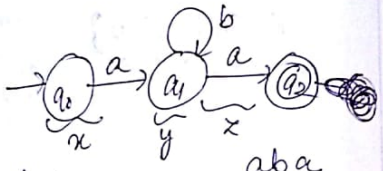
A language is said to be non regular if it is not recognised by any finite state machine.

Pumping lemma for regular languages:

Every regular language is associated with a special property i.e:

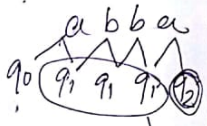
All strings in a language can be pumped if its length is at least a certain value called 'pumping length'.

Generally pumping length = no. of states.



The strings whose length is atleast the pumping length can be divided in 3 parts and one part is repeated any no. of times.

aba
ab ba ...
x y z
y repeated



The pumping lemma can be used to prove that a lang. is not regular.

Regular lang ⇒ satisfy pumping lemma

Not " → doesn't " "

Pumping lemma:- If A is a regular language then there is a number p (called the pumping length) where if S is any string in language A whose length is atleast p, then S can be divided into three pieces S = xyz, satisfying the following cond's.

(i) For each $i \geq 0$, $xy^iz \in A$.

(ii) $|y| > 0$ i.e $y \neq \epsilon$.

(iii) $|xy| \leq p$.

eg) $\alpha = \{a^n b^n \mid n \geq 0\}$.

PTO.

Steps to show that L is non regular using pumping lemma

Step1: Assume that L is regular.

Step2: So the pumping lemma guarantees that L can be pumped.
Let P be the pumping length.

Step3: Take a string S in L of length atleast P i.e $|S| \geq P$

Step4: Divide S into 3 parts i.e $S = xyz$.

Step5: Show that $xy^ix \notin L$ for some i.

Step6: Consider all possible ways of dividing S into xyz.

Step7: Show that none of the cases can satisfy all the conditions of the pumping lemma at the same time.

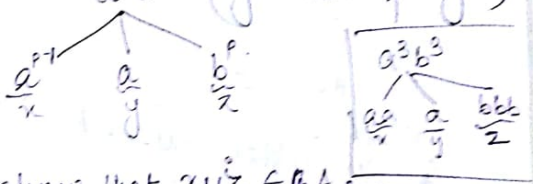
Step8: It is a contradiction.

Ex1: Show that $A = \{a^n b^n \mid n \geq 0\}$ is non regular.

Proof:

Assume that A is regular language.
So it has a pumping length i.e P
Take a string $S = a^P b^P$ in A.
Now $|S| = 2P \geq P$.
Divide S into 3 parts i.e $S = xyz$.

(i) $a^P b^P$ (y consists of only a's)



Now show that $xy^iz \in A, i \geq 0$

For $i = 2$:

$xy^iz = xy^2z = xyyz$

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

$= a^{P+1} b^P \notin A$

(as no. of a's & b's are not equal)

(ii) $a^P b^P$ (y consists of only b's)

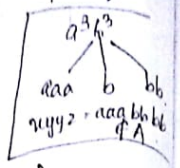
Now show that $xy^iz \in A, i \geq 0$.

for $i = 2$: $xyiz = xyz = xyyz$

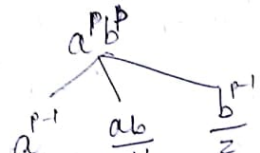
$= a^p b\, b^p b^{p-1}$

$= a^p b^{p+1} \notin A$

(as no. of a's & b's are not equal)



(ii) $a^p b^p$ (y consists of both a and b)



Now to show $xy^i z \in A$, $i \geq 0$

for $i=2$: $xy^2z = xyyz$

$= a^{p-1} ab\, ab\, b^{p-1} \notin A$ (as a is coming after b)

Hence non of the cases are satisfying the cond^n 1 of the pumping lemma.

So it is a contradiction to our assumption.

Hence A is not regular.

Qx: Prove that $B = \{w \mid w$ has equal no. of 0's and 1's$\}$ is non regular.

Proof: Assume that B is regular.

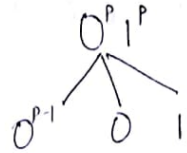
So it has pumping length p

Take a string S in B of length at least p i.e $|s| \geq p$.

Let $S = 0^p 1^p$.

Divide S into 3 parts: $xyz = s$

(i) $0^p 1^p$ (y contains only 0)

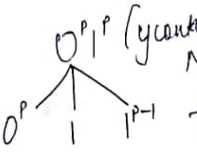


Now to show $xy^i z \in A$, $i \geq 0$.

for $i=2$: $xy^2z = xyyz$

$= 0^{p-1} 00\, 1^p = 0^{p+1} 1^p \notin A$

(ii) $0^p 1^p$ (y contains only 1)



Now to show $xy^i z \in A, i \geq 0$

for $i=2$: $xy^2z = xyyz = 0^p 11 1^{p-1}$

$= 0^p 1^{p+1} \notin A$

(iii) $0^p 1^p$ (y contains both 0 & 1)

To show $xy^i z \in A$, $i \geq 0$.

for i=2: $xyyz = xyyz = 0^{p-1}0101^{p-1}$

$= 0^{p+1}1^{p+1} \in A$ (satisfies 1st cond^n & 2nd.

2nd cond^n: $|01| > 0$
y

3rd cond^n:

$|0^{p-1}01| = |0^p 1| = p+1 \not\leq p$

x y. as 3rd cond^n not satisfied so not regulaer (as it is a contradiction)

Chapter-2: Context Free Grammar (CFG)

Languages

Generator → Grammar.

Acceptor → Finite Automation. Regular Expression

$S \rightarrow 0S1 \mid \epsilon$ (Grammar with three rules and 2 rules)

↳ Terminals

$S \rightarrow 0S1$
$0\epsilon1 = 01$

$S = 0S1$
$= 00S11$
$= 0011$

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

* A grammar is a collection of production rules also known as substitution Rules.

- A rule has variables and strings, separated by an arrow. $S \rightarrow 0S1$ (variables → non terminals, strings)

- Variable / Non Terminals :- uppercase letters.
- Terminals :- lowercase letters
- String consists of variables or non terminals and Terminals
- Start variable :- left side variable of the topmost rule.

Ex: Consider the fol. grammar G_1:

$A \to 0A1$
$A \to B$
$B \to \#$

Ans) Variables :- $\{A, B\}$
Terminals :- $\{0, 1, \#\}$
Start variable :- A

Derivation:

$A \Rightarrow 0A1$
$\Rightarrow 0B1$ $(A \to B)$
$\Rightarrow 0\#1$ $(B \to \#)$

$A \overset{*}{\Rightarrow} 0\#1$

↑ A produces $0\#1$ using production rule or substitution

The sequence of substitutions to obtain a string from a grammar is known as derivation

Q) Is $A \overset{*}{\Rightarrow} 0^3\#1^3$?

Ans)

$A \Rightarrow 0A1$
$\Rightarrow 00A11$ $(A \to 0A1)$
$\Rightarrow 000A111$ $(A \to 0A1)$
$\Rightarrow 000B111$ $(A \to B)$
$\Rightarrow 0^3\#1^3$ $(B \to \#)$

Yes, $A \overset{*}{\Rightarrow} 0^3\#1^3$

For the language:

$L = \{0^n\#1^n \mid n \geq 0\}$

The language generated by the context free grammar is known as context free language.

Context free language = Regular languages + Additional languages.

For a language L of a grammar G is denoted as :-

$$L(G) = \{w \in \Sigma^+ \mid S \overset{+}{\Rightarrow} w\}$$

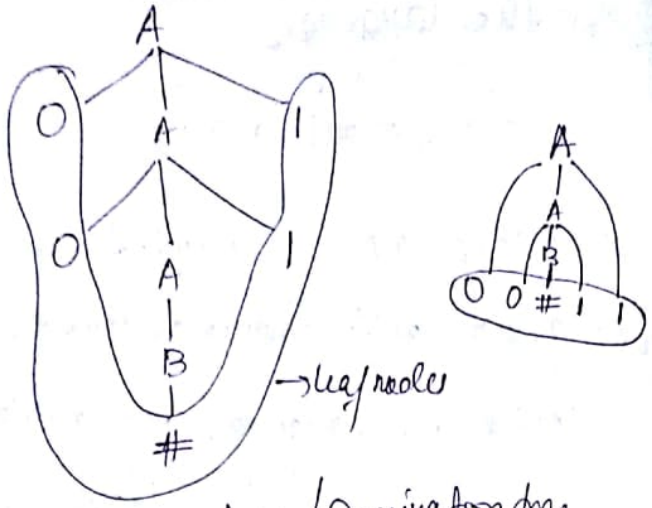
↓

Lang $G \to$ is a set consisting of string w belonging to the alphabet set Σ such that S produces w using 0 or more production rules.

Parse Tree

$A \overset{*}{\Rightarrow} 00\#11$

$A \Rightarrow 0A1$
$\Rightarrow 00A11$
$\Rightarrow 00B11$
$\Rightarrow 00\#11$



Parse tree / Derivation tree.

It is the graphical representation of the process of deriving a string from a grammar.

Components:

1. Root – Start variable
2. Non-leaf node – Variables.
3. Leaf node – Terminals.

Formal Def^n of CFG

A context-free Grammars (CFG) is a 4 tuple (V, Σ, R, S) where

i) V is the set of variables.

ii) Σ is the set of terminals, disjoint from V.

iii) R is the set of production rules with each rule consisting of variables & strings.

iv) $S \in V$ is the start variable.

Here $\alpha \in (V \cup \Sigma)^*$ for $A \to \alpha$.

$A \in V$

Ex:

$S \to 0S0 | 1S1$

$S \to 0 | 1 | \epsilon$.

Here $V = \{S\}$

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

R: $S \to 0S0 | 1S1$
$S \to 0 | 1 | \epsilon$.

$S = \{S\}$

Leftmost Derivation & Rightmost Derivation.

Leftmost Derivation:

→ Consider the following Grammare :-

$S \to aSS \mid aAS \mid \epsilon$

$A \to ba \mid SbA$

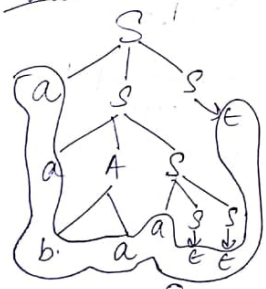
Is $S \overset{*}{\Rightarrow} aabaa$.

A→S) lmd : (Leftmost variable is expanded first).

~~$S \Rightarrow aSS$~~
~~$\Rightarrow aaSSS$.~~
~~$\Rightarrow aabaSS$.~~

~~$S \Rightarrow aAS$~~
~~$\Rightarrow aSbAS$~~
~~$\Rightarrow a$~~

$S \Rightarrow a\underline{S}S$
~~$\Rightarrow aaSSS$~~
$\Rightarrow aa\underline{A}SS$ (S→aAS)
$\Rightarrow aababSS$ (A→ba)
$\Rightarrow aababaSSS$ (S→aSS)
$\Rightarrow aababaa\epsilon\epsilon\epsilon$ (S→ϵ)
$\Rightarrow aababaa$



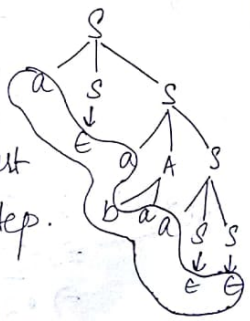
$\therefore S \overset{*}{\underset{lmd}{\Rightarrow}} aabaa$

lmd is obtained by applying the production to the leftmost variable of the string in each step.

$\therefore S \overset{*}{\underset{lmd}{\Rightarrow}} aabaa$

Rightmost Derivation:

rmd : $S \Rightarrow aS\underline{S}$
$\Rightarrow aSaA\underline{S}$ (S→aAS)
~~$\Rightarrow aSaA\epsilon\epsilon$~~
~~$\Rightarrow aSaaba$~~
$\Rightarrow aSaA\underline{a}SS$ (S→aSS)
$\Rightarrow aSaA\underline{a}a\epsilon\epsilon$ (S→ϵ)
$\Rightarrow aSaba\underline{a}a$ (A→ba).
$\Rightarrow aababaa$ (S→ϵ)



rmd is obtained by applying the production to the rightmost variable of the string in each step.

$\therefore S \overset{*}{\underset{rmd}{\Rightarrow}} aabaa$

Ex: Consider the foll. grammar :—

<EXPR> → <EXPR> + <TERM> | <TERM>

<TERM> → <TERM> × <FACTOR> | <FACTOR>

<FACTOR> → (<EXPR>) | a

Show that a + a × a and (a + a) × a can be generated from this grammar?

Ans)

<EXPR> ⇒ <EXPR> + <TERM>

⇒ <TERM> + <TERM>

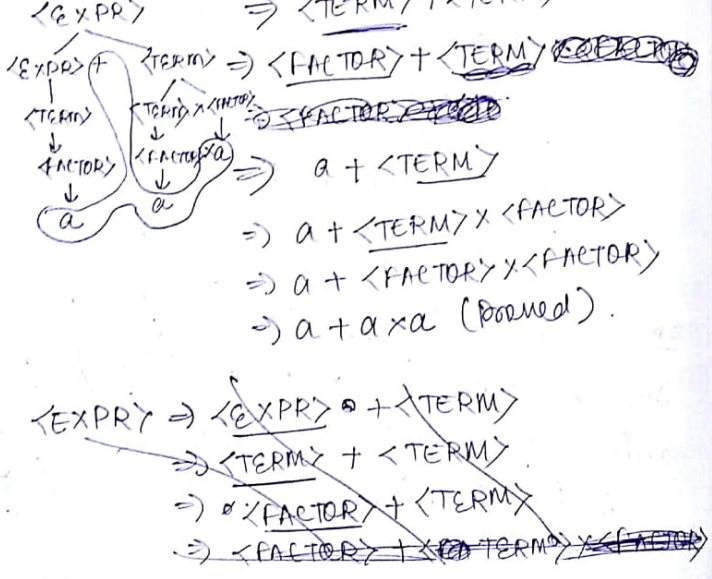
⇒ <FACTOR> + <TERM>

⇒ a + <TERM>

⇒ a + <TERM> × <FACTOR>

⇒ a + <FACTOR> × <FACTOR>

⇒ a + a × a (Proved).



⇒ (<EXPR>) + <TERM>

⇒ (<EXPR> + <TERM>) + <TERM>

⇒ (<TERM> + <TERM>) + <TERM>

⇒ (<FACTOR> + <TERM>) + <TERM>

⇒

<EXPR> ⇒ <TERM>

⇒ <TERM> × <FACTOR>

⇒ <FACTOR> × <FACTOR>

⇒ (<EXPR>) × <FACTOR>

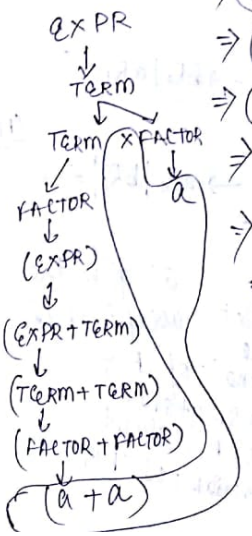
⇒ (<EXPR> + <TERM>) × <FACTOR>

⇒ (<TERM> + <TERM>) × <FACTOR>

⇒ (<FACTOR> + <TERM>) × <FACTOR>

⇒ (a + <FACTOR>) × <FACTOR>

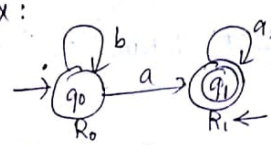
⇒ (a + a) × a (Proved)



Designing CFG

(i) Conversion of DFA to CFG.

Ex:



$L = \{a, ba, baba, aa, aab, \ldots\}$?
(strings containing atleast 1 a)

Now we consider 2 variables R_0, R_1 for the 2 states q_0, q_1 respectively.

rules: (for transition)

$\delta(q_0, b) = q_0 : R_0 \to bR_0$
$\delta(q_0, a) = q_1 : R_0 \to aR_1$
} $R_0 \to bR_0 | aR_1$

$\delta(q_1, a) = q_1 : R_1 \to aR_1$
$\delta(q_1, b) = q_1 : R_1 \to bR_1$
} $R_1 \to aR_1 | bR_1 | \in$

} CFG

* The lefthand side of top most rule is the start variable.

alphabet are the terminals of CFG & the start state of DFA is start variable of CFG.

as q_1 is the accept state

Steps:

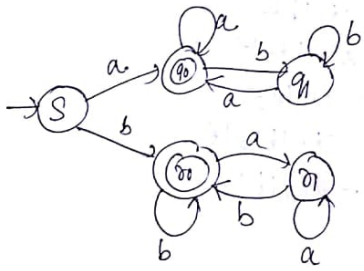
(i) make R_i as the variable of CFG for each state q_i of DFA.

(ii) Add the rule $R_i \to aR_j$ in the CFG if $\delta(q_i, a) = q_j$ is the transition in DFA.

(iii) Add the rule $R_j \to \in$ in the CFG if q_j is the accept state of DFA.

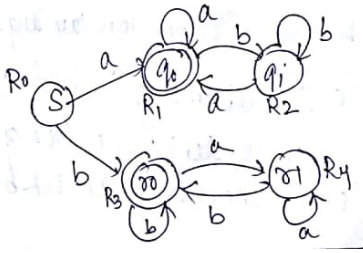
(iv) make R_0 as the start variable if q_0 is the initial/start state of DFA.

Ex2:



Convert to CFG

Ans)



Let $S = R_0$
$q_0 = R_1$
$q_1 = R_2$
$q_0 = R_3$
$q_1 = R_4$

$\delta(s,a) = q_0 : R_0 \to aR_1$ } $R_0 \to aR_1 | bR_3$
$\delta(s,b) = r_0 : R_0 \to bR_3$

$\delta(q_0,a) = q_0 : R_1 \to aR_1$ } $R_1 \to aR_1 | bR_2 | \epsilon$
$\delta(q_0,b) = q_1 : R_1 \to bR_2$

$\delta(q_1,a) = q_0 : R_2 \to aR_1$ } $R_2 \to aR_1 | bR_2$
$\delta(q_1,b) = q_1 : R_2 \to bR_2$

$\delta(r_0,a) = r_1 : R_3 \to aR_4$ } $R_3 \to aR_4 | bR_3 | \epsilon$
$\delta(r_0,b) = r_0 : R_3 \to bR_3$

$\delta(r_1,a) = r_1 : R_4 \to aR_4$ } $R_4 \to aR_4 | bR_3$
$\delta(r_1,b) = r_0 : R_4 \to bR_3$

CFG: $R_0 \to aR_1 | bR_3$
$R_1 \to aR_1 | bR_2 | \epsilon$
$R_2 \to aR_1 | bR_2$
$R_3 \to aR_4 | bR_3 | \epsilon$
$R_4 \to aR_4 | bR_3$

(ii) Second category: (for non regular lang. (no DFA))
The strings with 2 substrings that are linked together for some context free languages that needs to remember an unbounded amount of info for one substring that needs to be verified for the second substring.

$L = \{\epsilon, 01, 0011, 000111, \ldots\}$

$S \to uSv$

In these cases add the rule $S \to uSv$ where u refers to 1st substring & v refers to second substring.
For above L : $S \to 0S1 | \epsilon$

$S \to 0S1$ $S \to 0S1$
$\to 0\epsilon1$ $\to 00S11$
So $L = \{0^n1^n | n \geq 0\} \to 01$ $\to 00\epsilon11$
$\to 0011$
and so on....

→ Regular language is subset of CFG.

(iii) Third category: (combine 2 CFG's)
Individual grammars can be combined by using a new start variable ie:
$S \to S_1 | S_2 | \ldots | S_k$ where $S_1, S_2, \ldots S_k$ are

the start variables for k-different CFG.

Ex:

$S \to S_1 | S_2$; $L = L_1 \cup L_2$

$S_1 \to 0S_1 1 | \epsilon$; $L_1 = \{0^n 1^n | n \geq 0\}$

$S_2 \to 1S_2 0 | \epsilon$; $L_2 = \{1^n 0^n | n \geq 0\}$

new start variable

Conversion of CFG to DFA:

$S \to aA | bB$
$A \to \epsilon | bA$
$B \to bA | aS$ } CFG.

Draw the NFA:

Ans)

$S \to aA$.

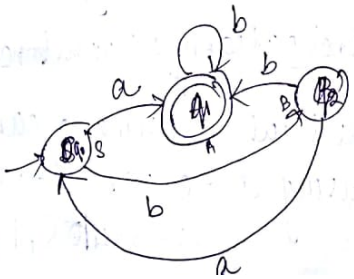
$\delta(S, a) = A$.

$\delta(S, b) = B$.

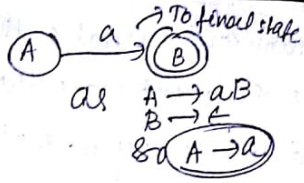
$\delta(A, b) = A$. (accept state)

$\delta(B, a) = S$

$\delta(B, b) = A$.



9) $A \to a$ means



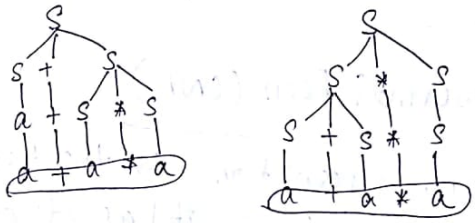
Ambiguity in CFG

$G: S \to S+S | S*S | a$

Is $a+a*a$ in $L(G)$?

(i) $S \Rightarrow \underline{S}+S$
$\Rightarrow a+\underline{S}$
$\Rightarrow a+\underline{S}*S$
$\Rightarrow a+a*\underline{S}$
$\Rightarrow a+a*a$

(ii) $S \Rightarrow \underline{S}*S$
$\Rightarrow \underline{S}+S*S$
$\Rightarrow a+\underline{S}*S$
$\Rightarrow a+a*\underline{S}$
$\Rightarrow a+a*a$



The string $a+a*a$ is generated ambiguously using the grammer.

If a grammar generates the same string in several different ways then the string is derived ambiguously in that grammar.

If a grammar generates the same string ambiguously then the grammar is ambiguous

A string w is derived ambiguously in the grammar G if it has two different leftmost derivations or diff parse trees.

(The same string is generated by 2 diff. left most derivation)

Chomsky Normal Form (CNF)

In CNF there is a restriction on the RHS of the production rule, i.e it has almost two variables or one single terminal in RHS.

$S \to AB$

$A \to a$

So a CFG will be in CNF if every rule is of the form $A \to BC$ or $A \to a$ where 'a' is terminal symbol and B, C are the variables & $S \to \epsilon$ is permitted where S is the start variable.

Conversion of CFG to CNF

Step-1: Add a new start variable S_0 and the rule $S_0 \to S$ if S occurs in the right hand side (RHS) of any rule.

Step-2: Eliminate all ϵ rules of the form $A \to \epsilon$, A is not the start variable.

Step-3: Eliminate all unit productions of the form $A \to B$.

Step-4: Convert the remaining rules into appropriate form.

Elimination of ϵ-rules

CFG: (Given)

$S \to ABAC$

$A \to aA \mid \epsilon$

$B \to bB \mid \epsilon$

$C \to c$

Removing $A \to \epsilon$: If A is there in RHS of any rule replace that with ϵ and add it to the rule.

$S \to ABAC$

$S \to BAC$

$S \to ABC$

$S \to BC$

$A \to aA$
$A \to a$

So the new set of rules are:

$S \to ABAC \mid BAC \mid ABC \mid BC$
$A \to aA \mid a$
$B \to bB \mid \epsilon$
$C \to c$

removing $B \to \epsilon$:

$S \to ABAC \mid BAC \mid ABC \mid BC \mid AAC \mid AC \mid C$.
$A \to aA \mid a$
$B \to bB \mid b$
$C \to c$.

Procedure: To remove $A \to \epsilon$, for each occurrence of A in RHS of a rule, a new rule is added with A deleted.

if $R \to uAv$ then $R \to uv$ is added.

So it becomes:

$R \to uAv \mid uv$.

if $R \to uAvAw$ then the new rules are: $R \to uAvAw$.

$R \to uAvw$
$R \to uvAw$
$R \to uvw$.

i.e, can be combined as:
$R \to uAvAw \mid uAvw \mid uvAw \mid uvw$.

faster way :- (eliminating ϵ rules)

$S \to ABAC$
$A \to aA \mid \epsilon$
$B \to bB \mid \epsilon$
$C \to c$

first find nullable variables (that produces ϵ)

Nullable variables = {A, B}

Removal of ϵ rules:

$S \to ABAC \mid BAC \mid ABC \mid BC \mid AAC \mid AC \mid C$.
$A \to aA \mid a$
$B \to bB \mid b$
$C \to c$

Elimination of unit Rules:-

Let the unit rule is of the form $A \to B$.

unit productions → S→A, A→B, B→D, D→a. (no meaning)

Directly
D→a
S→a

Let A→B then add A→a
B→a and eliminate A→B.

Eg
Consider the foll. grammar.
S→XY
X→a
Y→Z|b
Z→M
M→N
N→a

(I) eliminate M→N.
On removal of M→N add M→a

S→XY
X→a
Y→Z|b
Z→M
M→a N→a

(II) eliminate X→M
On removal of Z→M, add Z→a.
S→XY
X→a
Y→X|b.
Z→a
M→a
N→a

Z→✗→a.

(III) eliminate Y→X.

Y→✗→a.

S→XY
X→a
Y→a|b
X→a
M→a
N→a

Procedure: Steps to remove A→B
① Add the rule A→x if B→x occurs
② Remove A→B

Hence evident that Z, N, N are unreachable from start symbol S so we eliminate.

Eg:

$S \to Aa \mid B$

$B \to A \mid bb$

$A \to a \mid bc \mid B$

removing unit productions

(i) $B \to A$

add $B \to a$

$B \to bc$

add $B \to bc$

$B \to B$

add $B \to B$

(ii) $S \to A$

add $S \to a$

$S \to bc$

add $S \to bc$

$S \to B$

add $S \to B$

$S \to Aa \mid a \mid bc \mid B$

$B \to bb \mid a \mid bc$

$A \to a \mid bc$

removing $S \to B$:

$S \to B \to bb$

$\to A \to a$

$\to bc$

add $S \to bb \mid Aa \mid a \mid bc$.

removing $B \to A$

$B \to A \to a$

$\to bc$

$\to B$

add $B \to bb \mid a \mid bc$.

Now $A \to B \to bb$ add $A \to bb$

$\to A$

Eg:

$S \to Aa \mid bb \mid a \mid bc$

$B \to bb \mid a \mid bc$

$A \to a \mid bc \mid bb$

Q) $S \to AB$

$A \to a$

$B \to C \mid b$

$C \to D$

$D \to E$

$E \to a$

removing unit productions:

removing $B \to C$:

$B \to C \to D \to E \to a$

add $B \to a$

removing $C \to D$:

$C \to D \to E \to a$.

add $C \to a$.

removing $D \to E$:

$D \to E \to a$

add $D \to a$.

So: $S \to Aa$

$A \to a$

$B \to a \mid b$

$C \to a$

$D \to a$

$E \to a$

→ not reachable from S.

Convert the remaining rules into app. form:

$S \to bA \mid aB$
$A \to bAA \mid aS \mid a$
$B \to aBB \mid bS \mid b.$

$\Rightarrow$

$S \to C_b A \mid C_a B$
$A \to C_b D \mid C_a S \mid a$
$B \to C_a E \mid C_b S \mid b.$

where $C_b \to b$, $C_a \to a$, $D \to AA$, $E \to BB$ are added

(In CNF either 2 variables or single terminal allowed)

Procedure:

① Replace each rule of the form $A \to U_1 U_2 \dots U_k$, $k \geq 3$.
where U_i is a variable or terminal
with $A \to U_1 A_1$, $A_1 \to U_2 A_2$, $\dots$, $A_{k-2} \to U_{k-1} U_k$

② Replace each U_i with a variable i.e
$B_i = U_i$

Eg: $A \to aA_1$

Ex: Find the equivalent CFG in CNF. for the given grammar:

$S \to ASA \mid aB$
$A \to B \mid S$
$B \to b \mid \epsilon$.

(HW)

Step 1: add new start variable as S occurs in RHS:

$S_0 \to S$
$S \to ASA \mid aB$
$A \to B \mid S$
$B \to b \mid \epsilon$

Step 2: removing $B \to \epsilon$

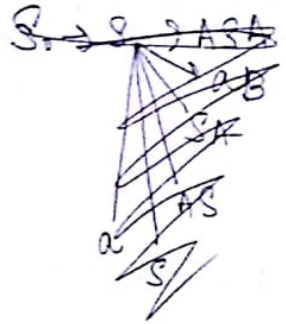
$S_0 \to S$
$S \to ASA \mid aB \mid a$
$A \to B \mid S \mid \epsilon$.
$B \to b$

nullable var = {B, A}

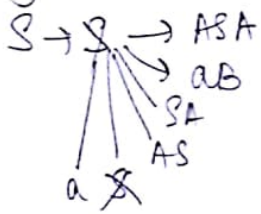
removing $A \to \epsilon$.

$S_0 \to S$
$S \to AS \mid SA \mid ASA \mid aB \mid a \mid S$
$A \to B \mid S$
$B \to b$.

Step 3: removing unit production rules:



removing $S \to S$:



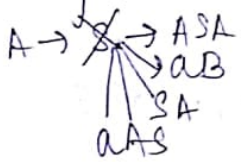
so $S \to ASA|aB|SA|AS|a$

removing : $A \to B$:

$A \to B \to b$

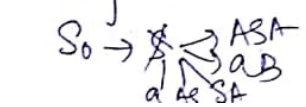
so $A \to b|S$.

removing : $A \to S$



so $\boxed{A \to b|ASA|aB|SA|AS|a}$

removing $S_0 \to S$:



so $S_0 \to ASA|aB|SA|AS|a$.

so finally : $S_0 \to ASA|aB|SA|AS|a$

$S \to ASA|aB|SA|AS|a$

$A \to b|ASA|aB|SA|AS|a$

$B \to b$.

step 4: Converting remaining rules into appropriate form :-

replace $SA = A_1$

$Ca = a$.

CNF

$S_0 \to AA_1|CaB|SA|AS|a$

$S \to AA_1|CaB|SA|AS|a$

$A \to b|AA_1|CaB|SA|AS|a$.

$B \to b$

$A_1 \to SA$ $Ca \to a$.

Push Down Automata (PDA)

$L = \{w$ contains atleast one $1\}$

$L = \{0^n 1^n | n \geq 0\} \to$ not a regular language.

Regular grammar → generates → Regular language → recognised/accepted by → FA.

CFG → Generates context Free Language → recognized by → PDA → has additional memory (Stack)

contains both regular as well as some other language.

FA → finite memory only remembers current state.

PDA → has additional memory.

Push Down Automata : is a computational model used to recognise the context Free Language in the similar way the Finite Automata (FA) is used for regular languages.

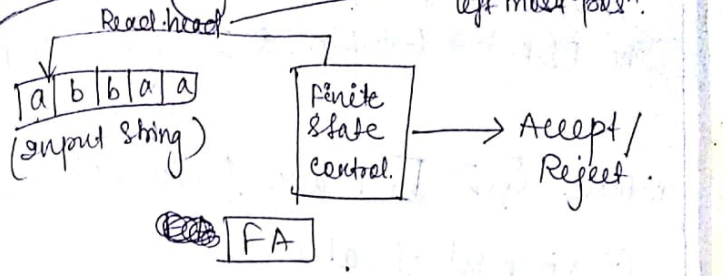
→ PDA is the non-deterministic Finite Automata (NFA) with an extra component called Stack.

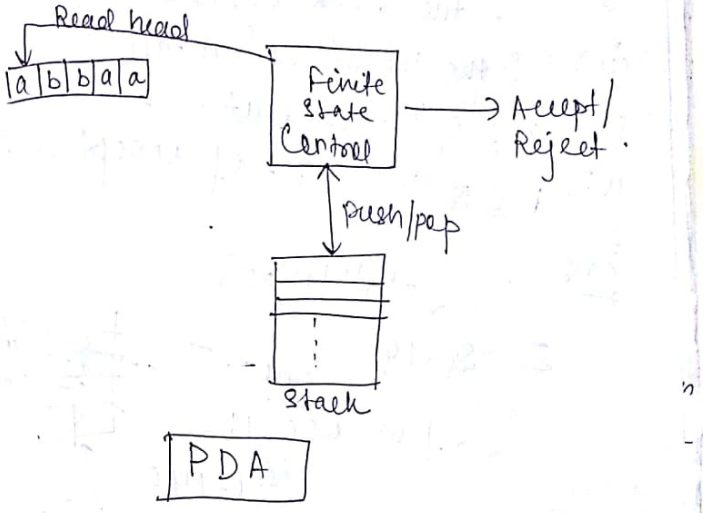
PDA = NFA + Stack.

Stack provides the additional memory

PDA is more powerful than Finite State Machine (FSM)

Schematic Diagram:





Components: (i) Finite state control :- It consist of states and transitions.

(ii) Input string.

(iii) Stack with infinite size.

Formal Def" of PDA :-

The PDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ (Γ : big gamma)

where Q, Σ, Γ, F are finite sets where

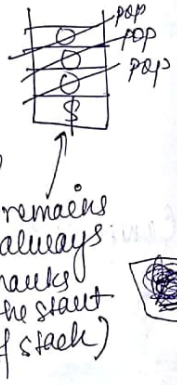
(i) Q is a set of states
(ii) Σ is the input alphabet
(iii) Γ is the stack alphabet.
(iv) δ is the transition function.
(v) $q_0 \in Q$ is the start state.
(vi) $F \subseteq Q$ is the set of accept state.

Ex: $L = \{0^n 1^n \mid n \geq 0\}$

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

If $w = 000111$ (push for 3, 1 pop 0)

So $\Gamma = \{\$, 0\}$.

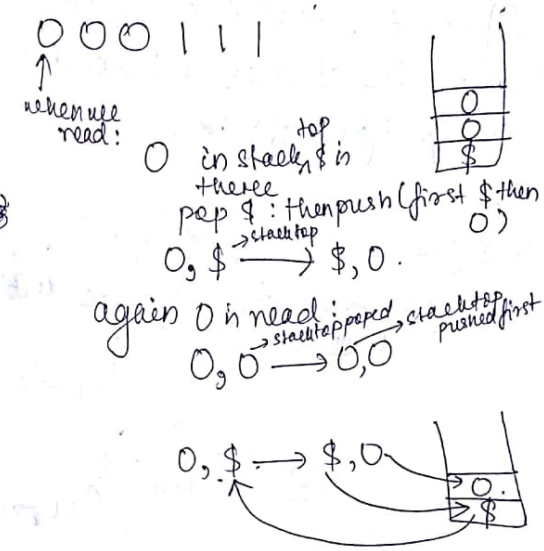


remains always (marks the start of stack)

If by end of w we get $ then string is accepted.

Transition function : δ

whenever we read a stack means we pop an element



The transition fun' in PDA takes 3 input parameters :- (i) current state
(ii) Input Symbol.
(iii) top element of stack.

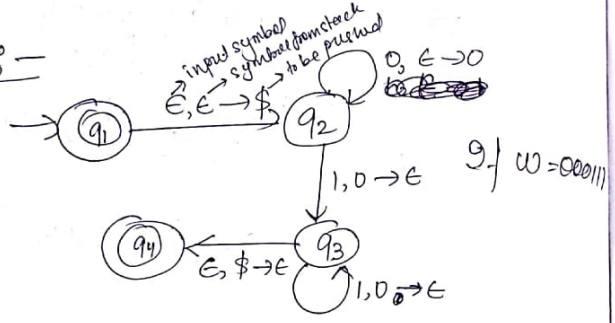
So, it in the triplet $\delta(q, a, x)$ where

(i) q : current state
(ii) a ; input symbol
(iii) x : top element in stack

If $a = \epsilon$, then the state changes without reading anything (from string).

If $x = \epsilon$, then the stack is not read or state changes without reading a symbol from stack.

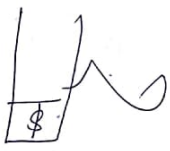
Ex:-



If $w = 000111$

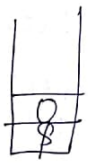
$\delta(q_1, \epsilon, \epsilon) = q_2$

↑ without reading stack or anything, travel to q_2. (push \$)

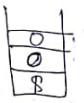


: $\delta(q_2, 0, \epsilon) = q_2$

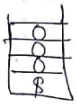
if 0 then without reading stack push 0.



$\delta(q_2, 0, \epsilon) = q_2$



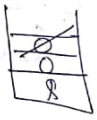
$\delta(q_2, 0, \epsilon) = q_2$



$1: \delta(q_2, 1, 0) = q_3$

↑ 1 in inputs, 0 in read or poped & nothing (ϵ) is pushed.

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



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

Now:

$\delta(q_3, \epsilon, \$) = q_4$ (accept)

↑ no input in w then \$ is poped & reaches accept state.

Range The output of a transition function is a pair (p, γ) where

p : next state.

γ : element/symbol to be pushed into stack.

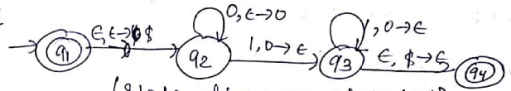
Hence: δ is defined as:

$$\delta : (Q \times \Sigma_\epsilon \times \Gamma_\epsilon) \longrightarrow P(Q \times \Gamma_\epsilon)$$

Domain — range.

as NFA so power set.

Ex:



(State diagram of PDA M_1)

Formal Description of M_1

M_1 is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$

where (i) $Q = \{q_1, q_2, q_3, q_4\}$.

(ii) $\Sigma = \{0, 1\}$

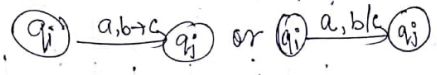
(iii) $\Gamma = \{0, \$\}$

(iv) q_0 is the start state

(v) $\{q_1, q_4\}$ is the set of accept states.

Transition function:

In PDA, the transition is represented as



(i) $a, b \to c$: for the input symbol a, then b is popped from stack & 'c' is pushed into the stack. (i.e 'c' replaces 'b')

(ii) $a, b \to \epsilon$: for input symbol a, pop b.

(iii) $a, \epsilon \to c$: for input symbol a, push c

(iv) $\epsilon, \epsilon \rightarrow \epsilon$: only state changes.

For the above PDA:

transition function (δ) is:

$\delta(q_1, \epsilon, \epsilon) = \{(q_2, \$)\}$

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

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

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

$\delta(q_3, \epsilon, \$) = \{(q_4, \epsilon)\}$

or

Input symbol:	0			1			ϵ (blank or ϕ)		
Stack symbol:	0	\$	ϵ	0	\$	ϵ	0	\$	ϵ
q_1									$(q_2, \$)$
q_2			$(q_2, 0)$	(q_3, ϵ)					
q_3				(q_3, ϵ)				(q_4, ϵ)	
q_4									

Processing a string

Let $w = 0011$

$(q_1, 0011, \epsilon)$ (initial state, w, stack top is empty initially.)

$\vdash$ (turnstyle notation) (processing ϵ)

$(q_2, 0011, \$)$

$\vdash$ $(q_2, 011, 0\$)$

$\vdash$ $(q_2, 11, 00\$)$

$\vdash$ $(q_3, 1, 0\$)$

$\vdash$ $(q_3, \epsilon, \$)$

$\vdash$ $(q_4, \epsilon, \epsilon)$ (accepted)

→ On receiving an input string 'w', the PDA goes from configuration to configuration. So, the configuration of PDA consist of:-

(i) State : q

(ii) remaining input string : w

(iii) stack contents : (γ)

So it is represented as a triple (q, w, γ) where $\gamma \in \Gamma^*$

Such a triple is called instantaneous Description (ID) of PDA

$\vdash$ (turnstyle) notation to represent the moves in PDA

Given:

$(q_2, 0011, \$)$

$\vdash$

$(q_2, 011, 0\$)$

The transition fun is $\delta(q_2, 0, \epsilon) = \{(q_2, 0$

input — element popped — element pushed

$\Rightarrow$ Let $\delta(q, a, x) = \{(p, \alpha)\}$.

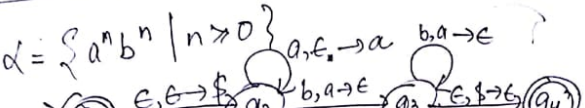
then $(q, aw, x\beta) \vdash (p, w, \alpha\beta)$.

where $w \in \Sigma^*$

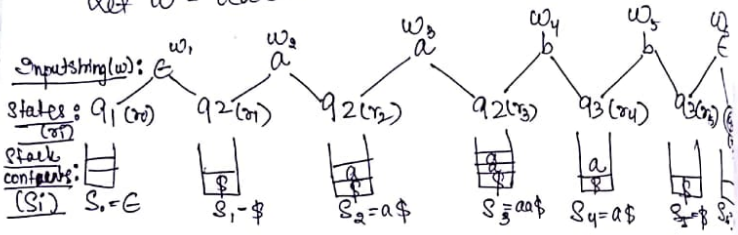
$\beta \in \Gamma^*$.

Formal Definition of Computation

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



Let $w = aabb$.



Let $M = (Q, \Sigma, \Gamma, \delta, q_0, f)$ be the PDA.

It accepts the input string $w = w_1 w_2 \ldots w_n$ where each $w_i \in \Sigma_\epsilon$ by going through the sequence of states $r_0, r_1, \ldots, r_m$ where each $r_i \in Q$ and $S_0, S_1, \ldots, S_m$ be the stack contents where each $S_i \in \Gamma^*$ and satisfy the following three conditions :-

(i) $r_0 = q_0$ (start state) and $S_0 = \epsilon$ (empty stack)

It states that the computation always starts from the start state and with an empty stack.

(ii) for $i = 0, 1, \ldots m-1$

$(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$

stack top.

where $S_i = at$

$S_{i+1} = bt$. also $a, b \in \Gamma_\epsilon$.

and $t \in \Gamma^*$.

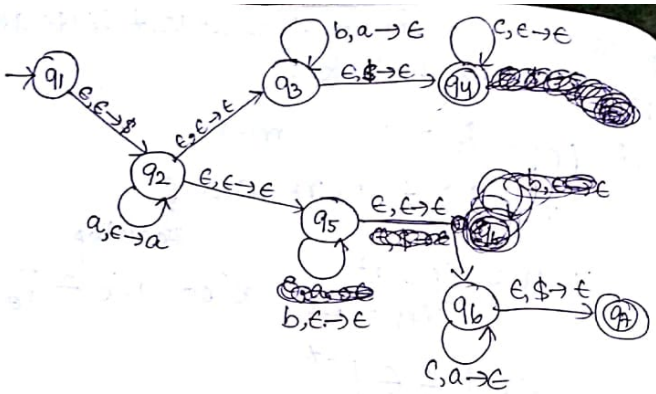
(iii) $r_m \in f$

that means that the PDA accepts the input string when it reaches the final/accept state.

Ex: Design a PDA that recognises the language: $L = \{a^i b^j c^k \mid i, j, k \geq 0$ and $i = j$ or $i = k\}$.

Ans) $L = \{aabbc, aaabbccc, \ldots\}$.

PTO



Q) Show the PDA configurations for input string $w = aaabbbcc$. (for above ques)

Ans)

$(q_1, aaabbbcc, \epsilon)$

$\vdash$

$(q_2, aaabbbcc, \$)$

$\vdash$

$(q_2, aabbbcc, a\$)$

$\vdash$

$(q_2, abbbcc, aa\$)$

$\vdash$

$(q_2, bbbcc, aaa\$)$

$(q_3, bbcc, aa\$)$

$\vdash$

$(q_3, bcc, a\$)$

$\vdash$

$(q_3, cc, \$)$

$\vdash$

(q_4, cc, ϵ)

$\vdash$

(q_4, c, ϵ)

$\vdash$

$(q_4, \epsilon, \epsilon)$

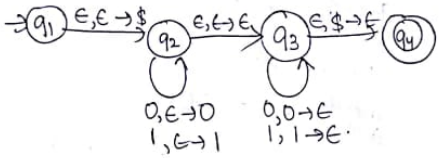
↳ accept state.

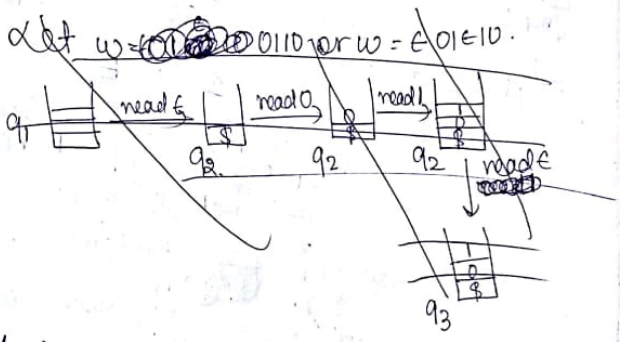
Ex: Construct a PDA for the language:

$L = \{ww^R \mid w \in \{0,1\}^*\}$

Ans)

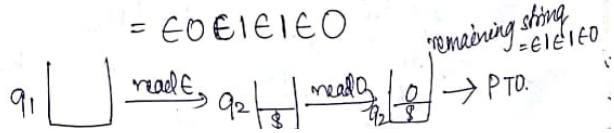
$L = \{\underbrace{0110}_{w\,w^R}, 010010, \ldots\}$

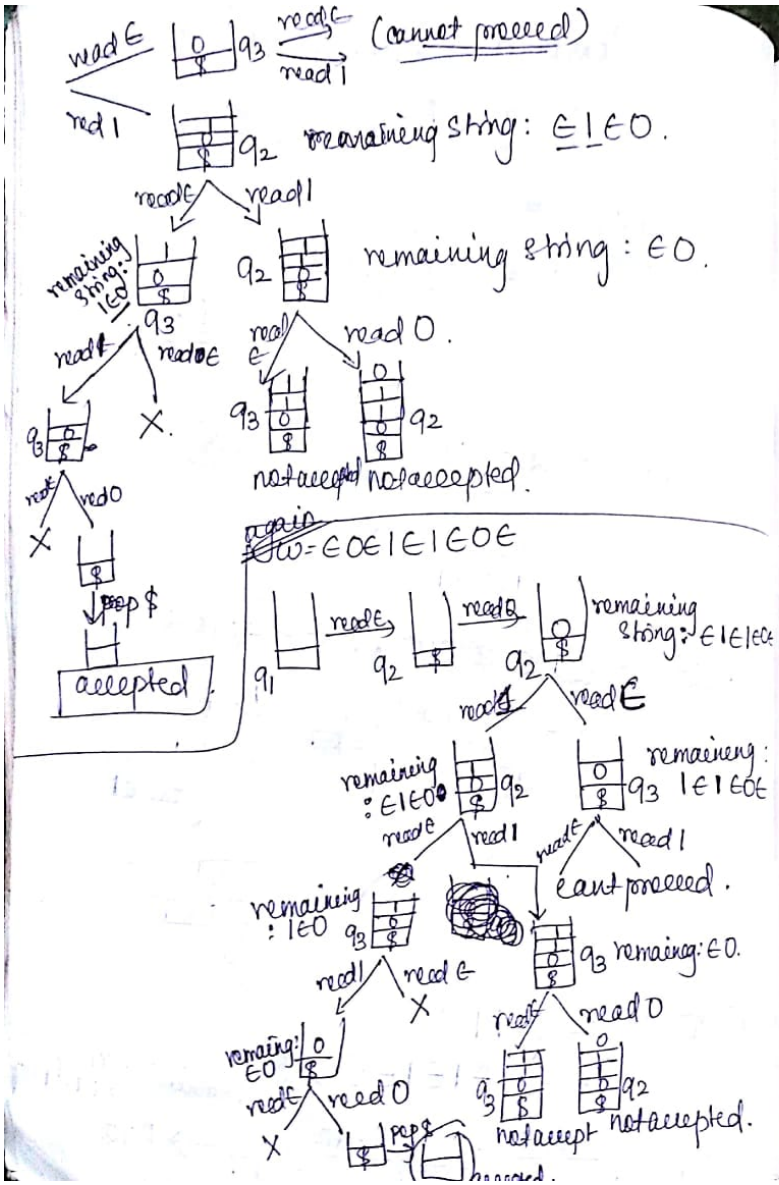


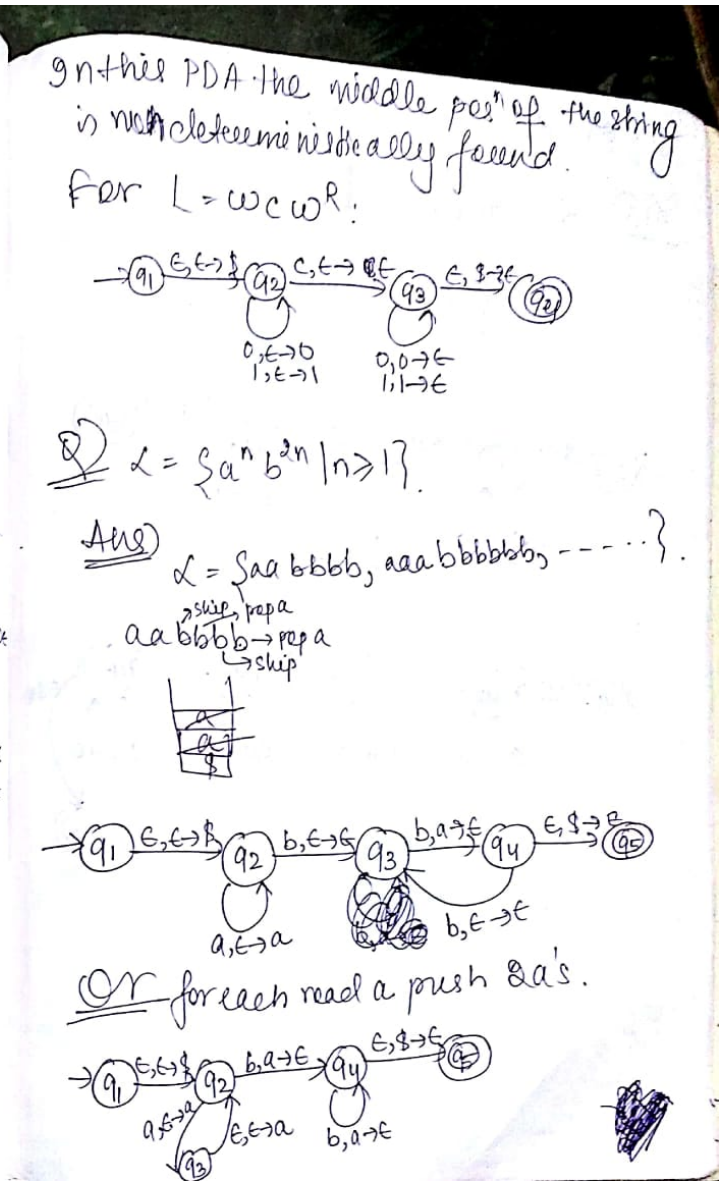


let $w = 0110$

$= \epsilon 0 \epsilon 1 \epsilon 1 \epsilon 0$



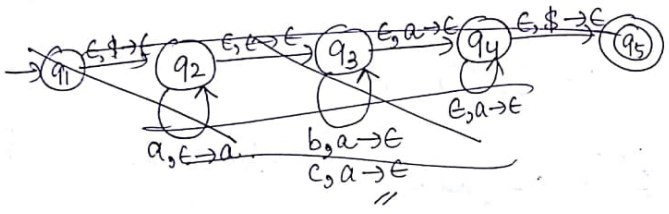


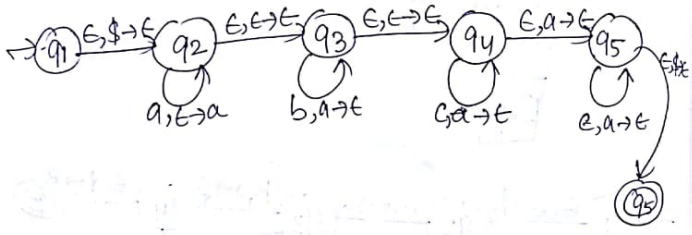


Q) Ex: $L = \{a^i b^j c^k \mid i = j + k\}$.

Ans) ~~$L = \{aabbccc, \ldots\}$~~

$L = \{aaabc, \ldots\}$.





Non Context Free Languages

⇒ The languages that are not recognized by any PDA are known as Non-context free languages.

Ex: $L = \{a^n b^n c^n \mid n \geq 0\}$

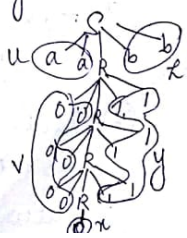
$L = \{ww \mid w \in \{0,1\}^*\}$.

Pumping lemma: can be used to prove that some languages are not context free.

The pumping lemma states that every context free lang. has a special value called the pumping length such that all strings whose length is atleast the pumping length can be pumped that means the string can be divided into 5 parts such that the second and fourth part can be repeated any number of times and the generated strings still remain in the language.

Ex: $S \to aaRbb$

$R \to 00R11 \mid 0 \mid 1$



$uv^i xy^i z \in L$

If A is a CFL, then there is a number p (called the pumping length) where if S is any string in A of length atleast p i.e $|S| \geq p$, then S can be divided into 5 parts i.e $S = uvxyz$ that satisfy the following conditions:-

(i) For each $i \geq 0$, $uv^ixy^iz \in A$.

(ii) $|vy| > 0$.

(iii) $|vxy| \leq p$.

To prove that a language A is not context free using pumping lemma follow these steps:-

1. Assume that A is context free
2. It has a pumping length p.
3. All strings of length atleast p can be pumped.
4. Take a string S in A such that $|S| \geq p$
5. Divide S into 5 parts i.e $S = uvxyz$
6. Show that $uv^ixy^iz \notin A$ for some i.
7. Consider the different ways that S can be divided.
8. Show that None of the cases need satisfy all the 3 cond's at the same time.
9. So, S cannot be formed which is a contradiction. That means, A is not a CFL

Ex1:

Prove that $L = \{a^n b^n c^n \mid n \geq 0\}$ is not CFL using pumping lemma.

Proof:

Assume that L is a context free language (CFL)

Acc^n to pumping lemma, L has a pumping length p. and any string in L of length atleast p can be pumped.

Take a string $S = a^p b^p c^p$ i.e $|S| = 3p \geq p$.

Divide S into 5 parts i.e $S = uvxyz$

case 1: both v and y contain same type of input symbols.

Let p=4 (using ex)

$S = a^4 b^4 c^4$

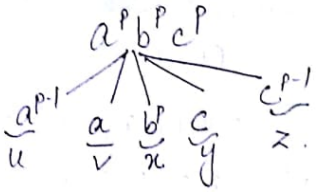
aaaa bbbb cccc.
u v x y z

let for i=2: $uv^2xy^2z = uvvxyyz$

$= aa.aaaa.bbbbcccccc.$

$= a^6 b^4 c^6 \notin L$

neither ex (general case):



for i=2:

uv^2xy^2z

$\Rightarrow a^{p-1} a^2 b^p c^2 c^{p-1}$

$= a^{p+1} b^p c^{p+1}$

$\notin L$

case 2: Either v or y contain diff type of symbols

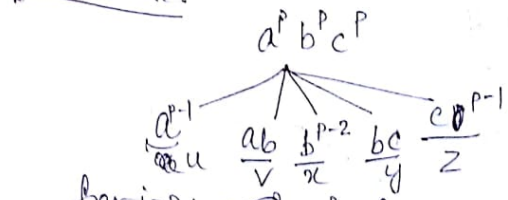
for p=4:

aaaa bbbb cccc
u v x y z.

for i=2: uv^2xy^2z = aaa abab bbbbcbcb ccc

$\notin L$

general case:



for i=2: uv^2xy^2z

$= a^{p-1} abab\ b^{p-2} bcbc\ c^{p-1}$

$\notin L$.

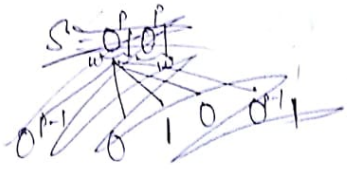
This string doesnot satisfy the ordering of the symbols in the language.

Condn 1 is not satisfied hence it is not a context free language.

Pg. 128-129 / examples.

Ex: $L = \{ww \mid w \in \{0,1\}^*\}$ is not CFL

Ans)



Consider $S = 0^p 1^p 0^p 1^p$

Let p=5:

S = 00000 11111 00000 11111

Case 1: vxy doesnot straddle any

boundary.

Let

$\underbrace{00000}_{u}\underbrace{11111}_{vxy}\underbrace{000000}_{z}\underbrace{11111}$

~~for i=2, uvvxyyz~~ for i=2:

$uvvxyyz$

$= 00000\,1111111\,00000\,11111$

$= 0^5 1^7 0^5 1^5 \notin L$

Case 2:

vxy (crosses) straddles the 1st boundary.

$\underbrace{00000}_{u}\underbrace{11}_{v}\underbrace{11}_{x}\underbrace{11}_{y}\underbrace{00000\,11111}_{z}$

For i=2:

$uvvxyyz$

$\Rightarrow 00000\,11111\,11\,00000\,11111$

$\Rightarrow 0^5 1^7 0^5 1^5 \notin L$

Case 3: vxy straddles the 2nd boundary.

~~00000 11111 00000 11111~~

$\underbrace{00000\,11111}_{u}\underbrace{00000}_{v}\underbrace{11}_{x}\underbrace{11}_{y}\underbrace{111}_{z}$

For i=2: $uvvxyyz$

$= 00000\,11111\,00\,00000\,11111\,111$

$= 0^5 1^5 0^7 1^7 \notin L$

case 4: vxy straddles the midpoint.

$\underbrace{00000\,11111}_{u}\underbrace{11}_{v}\underbrace{00}_{x}\underbrace{00}_{y}\underbrace{00\,11111}_{z}$

for i=2: $uvvxyyz$

$\Rightarrow 00000\,111\,11110\,0000\,00111\,11$

$= 0^5 1^7 0^7 1^5 \notin L$

So, L cannot be pumped that means it is a contradiction. So L is not a CFL

Equivalence of PDA with CFG.

Theorem:

A language is context free if and only if some Push Down Automata recognises it.

(i) A language is context free if some PDA recognises it i.e (CFG → PDA)

(ii) If some PDA recognizes a language then

it is context free. (PDA → CFG)

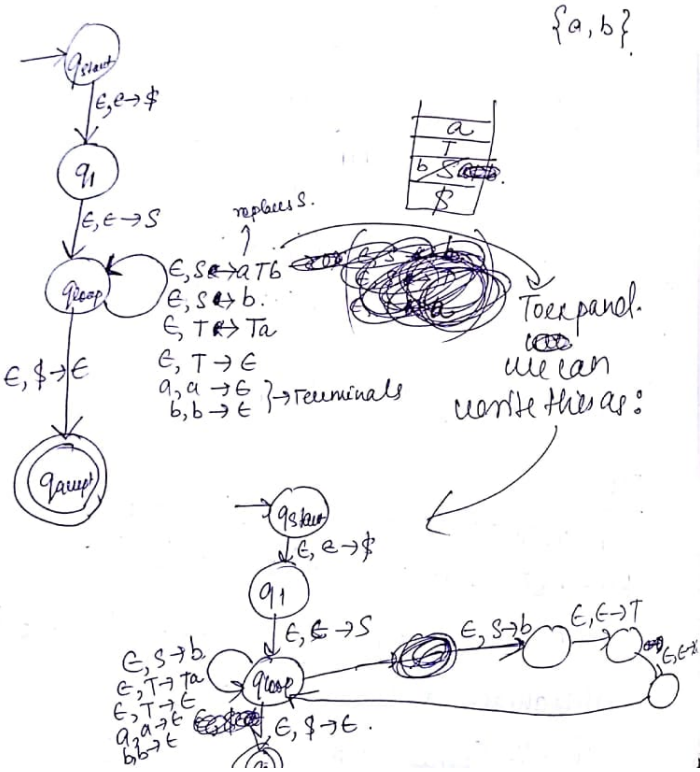
CFG → PDA

Consider the following CFG :-

$S \rightarrow aTb \mid b.$

$T \rightarrow Ta \mid \epsilon$

Terminals: {a, b}



Let w = aab.

$(q_{start}, aab, \epsilon)$

$\downarrow$

$(q_1, aab, \$)$

$\downarrow$

$(q_{loop}, aab, S\$)$

$\downarrow$

$(q_{loop}, aab, aTb\$)$

$\downarrow$

$(q_{loop}, ab, Tb\$)$ → shld be replaced such that ⓐ shld be ab.

$\downarrow$

$(q_{loop}, ab, Tab\$)$

$\downarrow$

$(q_{loop}, ab, \epsilon ab\$)$

$\downarrow$

$(q_{loop}, ab, b\$)$

$\downarrow$

$(q_{loop}, \epsilon, \$)$

$\downarrow$

$(q_{accept}, \epsilon, \epsilon)$

Construction of PDA:

① The states of PDA are:

$Q = \{q_{start}, q_{loop}, q_{accept}\} \cup E$ where E is set of some intermediate states

11) The transition fun's are defined as:

$\delta(q_{start}, \epsilon, \epsilon) = \{(q_{loop}, S\$)\}$

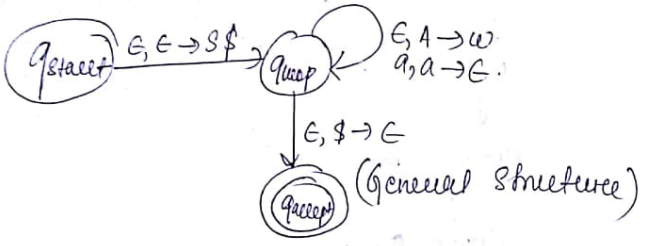
For any rule $A \to w$ in CFG,

$\delta(q_{loop}, \epsilon, A) = \{(q_{loop}, w)\}$

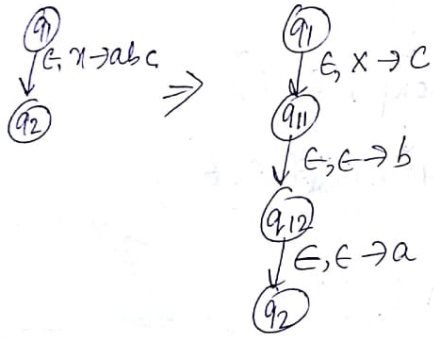
For all the terminals a,

$\delta(q_{loop}, a, a) = \{(q_{loop}, \epsilon)\}$

Also: $\delta(q_{loop}, \epsilon, \$) = \{(q_{accept}, \epsilon)\}$



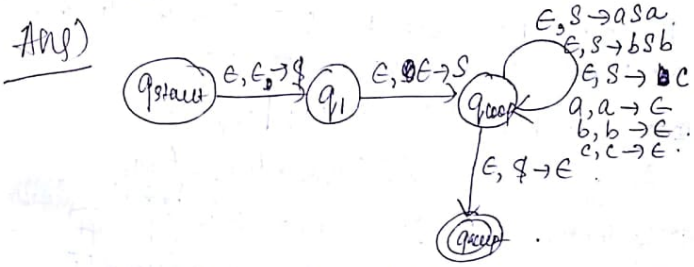
(General structure)



Q) Design a PDA for:

$S \to aSa \mid bSb \mid c.$

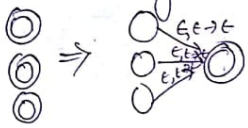
Ans)



Conversion of PDA → CFG

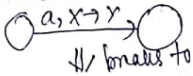
It is a 2 step process: ① Simplify the PDA
② Build the CFG.

Simplify the PDA: ① It has single accept state.



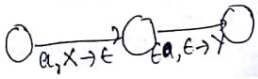
② It empties the stack before accepting that means it should start with an empty stack also finish with an empty stack.

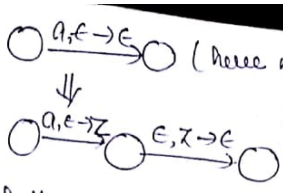
③ Each transition is either a push operation or a pop operation but not both at the same time.



(never push pop happening at same time).

// breaks to.





Build the CFG: Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$ be the PDA & G be its equivalent CFG.

① The variables of G are $\{A_{pq} \mid p, q \in Q\}$

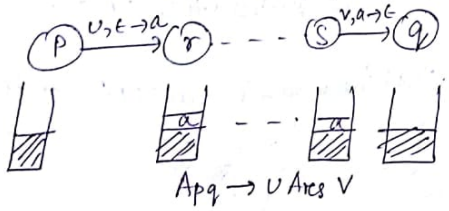
Eg: $q_1 \to q_2 \to q_3$: $A_{q_1q_2}$, $A_{q_2q_3}$, $A_{q_1q_3}$ → these are the variables

② The start variable of the grammar G is $A_{q_0, q_{accept}}$ and

the terminals will be the input symbols.

③ The rules of the grammar are defined as follows:-

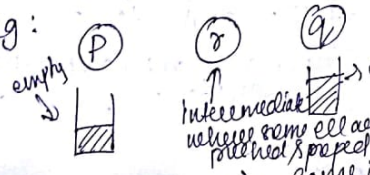
(i) For each $p, q, r, s \in Q$: If $\delta(p, u, \epsilon)$ contains (q, a) and $\delta(s, v, a)$ contains (r, ϵ) then add the rule: $A_{pr} \to u A_{qs} v$ in the grammar G.



(ii) For each $p, q, r \in Q$ add the rule:

$A_{pq} \to A_{pr} A_{rq}$.

Eg:



where some all are pushed & popped

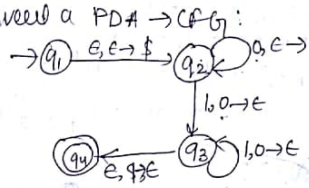
States remains same in all 3 states.

(iii) For any state $p \in Q$ add the rule:

$A_{pp} \to \epsilon$.

Example:

Convert a PDA → CFG:



Ans) start variable is: $A_{q_1q_4}$.

Between q_1 & q_4 we can add rule 3(i) as q_1 pushes $ & $ is popped by q_4 with input symbol ε.

$A_{q_1q_4} \to \epsilon A_{q_2q_3} \epsilon$.

For q_2 & q_3 again 3(i) can be applied:

$A_{q_2q_3} \to a A_{q_2q_3}$ b

also:

For q_2 we can apply 3(i):

$A_{q_2q_3} \to a A_{q_2q_2}$ b

Finally:

$A_{q_2q_2} \to \epsilon$

$A_{q_3q_3} \to \epsilon$ (not reachable, so not included)

Hence CFG: $A_{q_1q_4} \to \epsilon A_{q_2q_3} \epsilon$

$A_{q_2q_3} \to a A_{q_2q_3}$ b | $a A_{q_2q_2}$ b

$A_{q_2q_2} \to \epsilon$.

Deterministic PDA :- (D-PDA)

In D-PDA it has atmost one way to proceed. acc" to its transition function. The ϵ moves are allowed in DPDA :- (I) ϵ-input move. $\delta(q, \epsilon, x)$
$L = \{0^n 1^n / n > 0\}$ (II) ϵ-stack move $\delta(q, a, \epsilon)$
It can also be considered as
$\delta(q, \epsilon, \epsilon)$

Transitions are :
$\delta(q, a, x)$,
$\delta(q, \epsilon, x)$,
$\delta(q, a, \epsilon)$
$\delta(q, \epsilon, \epsilon)$

In DPDA exactly one of the values is not ϕ

- ww^R :- NPDA (we cannot get the mid point)
- wcw^R :- DPDA (we can get the mid point)

Formal Def" :-

A DPDA is a 6 tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where Q, Σ, Γ, are finite states and :-

1. Q is the set of states
2. Σ is the input alphabet
3. Γ is the stack alphabet
4. δ is the transition fun' defined as :- $\delta(Q \times \Sigma_\epsilon \times \Gamma_\epsilon) \to Q \times \Gamma_\epsilon$ U ϕ
5. q_0 is the start state
6. F is the set of accept states

In DPDA
$|\delta(q, a, x)|$ where $a \in \Sigma$ or $a = \epsilon$
$x \in \Gamma$ or $x = \epsilon$

$|\delta(q, a, x)| \leq 1$