UNIT I

AUTOMATA FUNDAMENTALS

1.1 INTRODUCTION TO AUTOMATA THEORY

Automata theory is the study of abstract machines and the computational problems can be solved using these machines. Abstract machines are called automata. The name comes from the Greek word (Αυτόματα).

It means doing something by itself. An automaton can be a finite representation of a formal language that may be an infinite set. Automata are used as theoretical models for computing machines, and are used for proofs about computability. The automata theory is essential for,

- The study of the limits of computation
- Designing and checking the behaviour of digital circuits.
- Pattern searching in Websites
- Verifying systems of all types that have a finite number of distinct states, such as communications protocols or protocols for secure exchange information

1.1.1 INTRODUCTION TO FORMAL LANGUAGES

Formal languages are the system used to train the machines in recognizing certain commands or instructions. These languages are the abstraction of natural languages, since they are expended by the machines. Formal languages are of five types. They are:

- Regular Languages (RL)
- Context free Languages (CFL)
- Context Sensitive Languages (CSL)

- ❒ Recursive Languages
- ❒ Recursively Enumerable Languages (RE)

➠ These languages are recognized by specific automata/machines and grammars.

- ❒ Regular grammars (type 3) and finite automata recognize regular languages.
- ❒ Context free grammars (Type 2) and push down automata recognize context free languages.
- ❒ Context sensitive grammars (Type 1) and Linear Bounded Automata (LBA) recognize context sensitive languages.
- ❒ Unrestricted grammars (phrase structure grammar) (Type 0).
- ❒ Turing machines recognize recursively enumerable languages.

➠ Total Turing Machines (TTM) that halt for every input are used to recognize recursive languages.

1. Formal Language Theory

Formal language theory describes languages as a set of operations over an alphabet. It is closely linked with automata theory, as automata are used to generate and recognize formal languages. Automata are used as models for computation; formal languages are the preferred mode of specification for any problem that must be computed.

2. Computability theory

Computability theory deals primarily with the question of the extent to which a problem is solvable on a computer. It is closely related to the branch of mathematical logic called recursion theory.

3. Models of Computation

The computation models that are developed by formal language theory are ,

i) Finite State Automata

ii) Regular expression

iii) Push down Automata

iv) Linear bounded automata

v) Turing machine

- The computational models and the languages understandable by these models are tabulated below.

Table 1.1 The Computational Models

Machines	Grammars/ Languages	Category	
Finite State Automata (Regular Expression)	Regular	Type 3	Simple
Push Down Automata	Context Free	Type 2	
Linear Bounded Automata	Context Sensitive	Type 1	
Turing Machine	Phrase Structure	Type 0	Complex
Uncomputable			

1.1.2 Basic Mathematical Notation and Techniques

1. Alphabet

An alphabet is a finite, nonempty set of symbols.

Example:

i. $\sum=\{0,1\}$

ii. $\sum=\{a,b,c\}$

2. String

A string over an alphabet is a finite sequence of symbols from that alphabet.

Example:

i. 01001 over $\sum = \{0,1\}$

ii. aaabbbbccc over $\sum = \{a,b,c\}$

3. Length of a string

The length of a string is the count of symbols in that string.

Example:

i. $|01001| = 5$

ii. $|aaabbbbccc| = 10$

iii. $|0^3 1^5| = 8$

4. Power of an alphabet

The power of an alphabet $\sum k$, is the set of all strings over $\sum$ with length k.

Examples:

$$\Sigma = \{0,1\}$$
$$\Sigma^2 = \{00, 01, 10, 11\}$$
$$\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111 \ldots\ldots\ldots\ldots\}$$
$$= \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \ldots\ldots\ldots\ldots\ldots\ldots$$
$$= \Sigma^0 \cup \Sigma^+$$
$$\Sigma^+ = \{0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111 \ldots\ldots\ldots\ldots\}$$
$$= \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots\ldots\ldots\ldots\ldots\ldots$$

5. Language (L)

The language of an Automata is a set of strings accepted by the automata.

Examples:

i. Set of even length strings over an alphabet {a,b}.

ii. Set of odd length strings over an alphabet {0,1}.

6. Set –former notation of a Language

i. L={w |w consists of an equal number of 0's and 1's }

L={ε, 01, 10,0011, 1100, 0101, 1010, 1001, 0110,………}

ii. L= {w |w is a binary integer that is prime }

L={10, 11, 101, 111, …….}

7. Complement of a Language($\overline{L}$)

i. L={w |w contains 101 as substring}

$\bar{L}$ ={w |w does not contain 101}

ii. L={w |w contains abb as substring}

L ={w |w does not contain abb}

1.2 INTRODUCTION TO FORMAL PROOF

A formal proof or derivation is a finite sequence of sentences called well-formed formulas in the case of a formal language each of which is an axiom or follows from the preceding sentences in the sequence by a rule of inference. But in deductive proofs, the truth of a statement is shown by a detailed sequence of steps and reasons.

Some computer scientists take the extreme view that a formal proof of the correctness of a program should go hand-in-hand with the writing of the program itself. We doubt that doing so is productive. Some also say that proof has no place in the discipline of programming.

☞ The slogan "if you are not sure your program is correct, run it and see" is commonly offered by them.

- ☞ Testing programs is surely essential. However, testing goes only so far, since you cannot try your program on every input.
- ☞ To make your iteration or recursion correct, you need to set up an inductive hypothesis, and it is helpful to reason, formally or informally, that the hypothesis is consistent with the iteration or recursion.
- ☞ This process of understanding the workings of a correct program is essentially the same as the process of proving theorems by induction.
- ☞ Automata theory covers methodologies of formal proof. It can be of either :

➢ **Inductive kind**

Recursive proofs of a parameterized statement that use the statement itself with lower values of the parameter.

➢ **Deductive kind**

A sequence of justified steps.

1.2.1 Deductive Proofs

- ➠ A deductive proof consists of a sequence of statements whose truth leads us from some initial statement, called the hypothesis or the given statement(s), to a conclusion statement.
- ➠ Each step in the proof must follow, by some accepted logical principle, from either the given facts, or some of the previous statements in the deductive proof, or a combination of these.
- ➠ The hypothesis may be true or false, typically depending on values of its parameters. Often, the hypothesis consists of several independent statements connected by a logical AND.
- ➠ The theorem that is proved when we go for a hypothesis H to a conclusion C is the statement "if H then C". We say that C is deduced from H.
- ➠ An example theorem of the form "if H then C" will illustrate these points.

Theorem	1

If $x \geq 4$, then $2^x \geq x^2$.

Proof

The hypothesis H is "$x \geq 4$". This hypothesis has a parameter, x and thus is neither true nor false. Rather, its truth depends on the value of x.

H is true for $x = 6$ and false for $x = 2$.

- The conclusion C is "$2^x \geq x^2$". This statement also uses parameter x and is true for certain values of x and not others.

Example:

C is false for $x = 3$, since $2^3 = 8$, which is not as large as $3^2 = 9$. On the other hand, C is true for $x = 4$, since $2^4 = 4^2 = 16$. For $x = 5$, the statement is also true, since $2^5 = 3^2$ is at least as large as $5^2 = 25$.

Perhaps you can see the intuitive argument that tells us the conclusion $2^x = x^2$ will be true whenever $x \geq 4$. We already saw that it is true for $x = 4$. As x grows larger than 4, the left side, 2^x doubles each time x increases by 1.

However, the right side, x^2, grows by the ratio $(x+1/x)^2$.

If $x \geq 4$, then $(x + 1)/x$ cannot be greater than 1.25, and therefore $(x+1/x)^2 = (1.25)2 = 1.5625$.

Since $1.5625 < 2$, each time x increases above 4 the left side 2^x grows more than the right side x^2.

- Thus, as long as we start from a value like $x = 4$ the inequality $2^x \geq x^2$ is already satisfied.

Theorem	2

If x is the sum of the squares of four positive integers, then $2^x \geq x^2$.

Proof

In deductive proof, we go from a hypothesis H to a conclusion C, i.e., if H then C.

Step 1:

x is the sum of the squares of four integers. Let a, b, c, d be four integers.

x = a2 + b2 + c2 + d2

Step 2:

The integers being squared are at least 1.

$a \geq 1; b \geq 1; c \geq 1; d \geq 1.$

Step 3:

Since the integers is at least 1, then its squares is also at least 1.

$a^2 \geq 1; b^2 \geq 1; c^2 \geq 1; d^2 \geq 1$

Step 4:

From Step 1 and Step 3, we can inter that x is sum of four squares and each squares is at least 1. x is at least 1+1+1+1 $x \geq 4$.

Step 5:

Step 4 is the hypothesis of the previous problem ($2^x \geq x^2$ if $x \geq 4$). We can conclude that,

$2^x \geq x^2$ (or) $a^2 + b^2 + c^2 + d^2$

1.2.2 Reduction to Definitions

If the hypothesis does not use familiar terms like integer, multiplication, addition etc., then we can convert all terms in the hypothesis to their definitions.

Theorem	**3**

A set S is finite if there exists an integer n such that S has exactly n elements. $|S| = n$.

Where,

n - Number of elements in the set S.

S and T - Both subsets of some infinite set U.

T - Complement of S (with respect to U) if $S \cup T = U$ and $S \cap T = \Phi$

T - Infinite.

Proof

We can use proof by contradiction. It is a technique where we assume that the conclusion is false. Then use that assumption together with hypothesis, prove the opposite of one of the given statements of the hypothesis. So the only possibility that remains is that the conclusion is true whenever the hypothesis is true.

Here T is finite (because we assume the conclusion is false), but T is infinite .

Given

Let us assume T is finite, along with the statement of the hypothesis, S is finite. i.e., |S| = n for some integer n. |T| = m for some integer m.

Now given statement tells us that S ∪ T = |S| + |T| = n+m, n+m is a integer it follows U is finite. But it contradicts the given statement U is infinite.

So the conclusion is true whenever the hypothesis is true. Therefore T is infinite.

1.2.3 Other Theorem Forms

1. If - then

- The most common forms of if - then statements are if H then C can be rewritten as
  - i. H implies C
  - ii. H only if C
  - iii. C if H
  - iv. Whenever H holds, C follows.
- So the theorem if $x \geq 4$, $2^x = x^2$ can be rewritten as
  - (a) $x \geq 4$, implies $2^x = x^2$
  - (b) $x \geq 4$ only if $2^x = x2$
  - (c) $2x = x_2$ if $x \geq 4$
  - (d) Whenever $x \geq 4$ holds, $2^x = x2$ follows.

2. If - and - only - if statements

The statements of the form “A if and only if B” or “Aiff B” has two if - then statements, is “if A then B” and “if B then A”.

Note:

$\lfloor x \rfloor$ - Floor of real number x, is the greatest integer equal to or less than x.

$\lceil x \rceil$ - Ceiling of real number x, is the least integer equal to or greater than x.

Theorem	**4**

Let x be a real number. Then $\lfloor x \rfloor = \lceil x \rceil$, if and only if x is an integer.

Proof

$\lfloor x \rfloor \leq x$ by definition of floor ... (1.1)

$\lceil x \rceil \geq x$ by definition of ceiling ... (1.2)

We are given with $\lfloor x \rfloor = \lceil x \rceil$

Substituting (1.1) in (1.2), we get,

$$\lceil x \rceil \leq x.$$

Since $\lceil x \rceil \geq x$, by arithmetic inequality we get

$$\lceil x \rceil = x.$$

3. Theorems that appear “Not to be if-then statements.

Theorem	**5**

Sin2ɵ¸ + Cos2ɵ = 1.

It does not have any hypothesis. This theorem can be written in if-then is “if ɵ¸ is an angle, then sin2ɵ + cos2ɵ = 1”.

1.3 ADDITIONAL FORMS OF PROOF

The following are the additional forms of proofs.

- ❒ Proofs about sets
- ❒ Proofs by contradiction
- ❒ Proofs by counter example

1.3.1 Proofs About Sets

- ➠ Sets contain symbols to form character strings.
- ➠ Sets in automata theory are called as languages.

If E and F are two expressions representing sets, the statement E = F means that two sets represented are same. i.e., every element in the set represented by E is in the set represented by F and vice versa.

Example:

Union of sets obeys commutative law. Let S and R be sets. Then $R \cup S = S \cup R$. If E is the expression in $R \cup S$ and F is the expression $S \cup R$, then E = F, an element x is in E iff x is in F.

Theorem	**6**

Distributive law of union over intersection. Let R, S, T be sets. Then prove that

$$R \cup (S \cap T) = (R \cup S) \cap (R \cup T)$$

Proof

Aiff B has two parts. That are,

- ❒ **If part:** "if B then A"
- ❒ **Only if part:** if A then B, which is equivalent form "A only if B".

Let E is the expression in $R \cap (S \cup T)$ and F is the expression in $(R \cup S) \cap (R \cup T)$.

1.. If part: if x is in E, x is in F

Statement	Justification
(a) x is in $R \cap (S \cup T)$	(a) Given
(b) x is in R or x is in $(S \cap T)$	(a) and definition of union.
(c) x is in R or x is in both S and T	(b) and definition of intersection.
(d) x is in $R \cup S$	(c) and definition of union.
(e) x is in $R \cup T$	(c) and definition of union.
(f) x is in $(R \cup S) \cap (R \cup T)$	(d), (c) and definition of intersection.

2. We can also prove it by "only – if" statement.

Statement	Justification
(a) x is in $(R \cup S) \cap (R \cup T)$	Given
(b) x is in $R \cup S$	(a) and definition of intersection.
(c) x is in $R \cup T$	(a) and definition of intersection.
(d) x is inR or x is in both S and T	(b), (c) and reasoning about unions.
(e) x is in R or x is in $S \cap T$	(d) and definition of intersections.
(f) x is in $R \cup (S \cap T)$	(e) and definition of union.

Example:

"if $x \geq 4$, then $2x \geq x2$". Then the contrapositive of this statement is "if not $2x \geq x2$, then not $x \geq 4$". It is otherwise "not $a \geq b$" is equal to $a < b$.

not $x \geq 4$ is $x<4$

not $2x \geq x2$ is $2x< x2$.

i.e., "if $2x< x2$, then $x < 4$"

1.3.2 Proof by Contradiction

1. It is derived from Latin meaning reduction to the "absurd".
2. In proof by contradiction, we assume that the theorem is false and then show that this assumption leads to an obviously false consequence, called a contradiction.
3. Another way to prove a statement of the form "if H then C" is to prove the statement "H and not C implies falsehood".

Step 1:

State by assuming both the hypothesis H and the negation of the conclusion C.

Step 2:

Compute the proof by showing that something known to be false follows logically from H and C. This form of proof is called proof by contradiction.

Example:

Jack Sees Jill, who has just come in from outdoors. On observing that she is completely dry, he knows that it is not raining.

Proof

His proof = that it is not raining

Assume the negation of conclusion, ie, it is raining, then Jill would be wet. But she is not wet, so it must not be raining.

Theorem	**7**

Let S be a finite subset of some infinite set U. Let T be complement of S with respect to U. Then T is infinite.

Proof

H = S is finite set of U

U is an infinite set

T is the complement of S with respect to U.

Conclusion C = "T is infinite"

- We proceed to prove the theorem by proof by contradiction.
- We assume not C is true. T is finite.

From the assumption S is finite, if T is also finite, then U = S ∪ T is also finite. But hypothesis says U is infinite. Therefore the logical statement is false.

1.3.3 Proofs by Counter Examples

It is an exception to a proposition general rule. i.e. Specific instance of the falsity of a universal quantification.

Example:

The statement "all students are lazy".

Proof

Counter example, a hardworking diligent student counters the statement.

Theorem	**8**

All primes are odd.

Proof

The integer 2 is prime, but 2 is even.

Theorem	**9**

There is no pair of integers a and b such that a mod b = b mod a.

Proof

Let us assume a < b.

a mod b = a a = qb + r

a = 0 x b + a. q = quotient r = remainder.

But b mod a < a, is between 0 - 1. Thus when a < b, b mod a < a mod b, so a mod b = b mod a is impossible. It is same argument for a > b also.

Consider a = b. a mod b = b mod a = 0 (x mod x = 0) by counter example, let us take a = b = 2, a mod b = b mod a = 0 i.e., 2 mod 2 = 2 mod 2 = 0.

1.4 INDUCTIVE PROOFS

1.4.1 Induction on Integers

Proof by Induction is a technique by which the truth of a number of statements can be inferred from the truth of a few specific instances. Suppose, let P(n) be a statement about a non-negative integer n. The principle of mathematical induction is that P(n) follows from:

- P(1)
- P (n-1) implies P(n) for all $n \geq 1$.

Condition (a) is called basis and the condition (b) is called inductive step, because it connects Pn with Pn+1.

1.4.2 Structural Inductions

- In automata theory, there are several recursively defined structures about which we need to prove statements.
- The examples are trees and expressions.
- Like inductions, all recursive definitions have a basis case, where one or more elementary structures are defined, and an inductive step, where more complex structures are defined in terms of previously defined structures.
- Structural induction is a proof method that is used in mathematical logic, computer science, graph theory, and some other mathematical fields. It is a generalization of mathematical induction.
- A recursive definition or inductive definition is one that defines something in terms of itself (that is, recursively), in a useful way.

Example:

Let us take expressions using the arithmetic operators + and *, with both numbers and variables allowed as operands.

Basis

Any number or letter (i.e., a variable) is an expression.

Induction

If E and F are expressions, then so are E + F, E*F, and (E).

Example:

Both 2 and x are expressions by the basis. The inductive step tells us x+2, (x + 2) and 2*(x + 2) are all expressions. Notice how each of these expressions depends on the previous ones being expressions.

When we have a recursive definition, we can prove theorems about it using the following proof form, which is called structural induction.

Let S(X) be a statement about the structures X that are defined by some particular recursive definition.

- As a basis, prove S(X) for the basis structure(s) X.
- For the inductive step, take a. structure X that the recursive definition says is formed from Y 1,Y2,..., Yk. Assume that the statements S(Y1), S(Y2), ….., S(Yk), and use these to prove S(X).

Our conclusion is that S(X) is true for all X. The following Theorem gives the facts and proof for trees and expressions.

Theorem	**10**

Every tree has one more node than it has edges.

Proof

The formal statement S(T) we need to prove by structural induction is: "if T is a tree, and T has n nodes and e edges, then n = e + 1".

Basis

The basis case is when T is a single node. Then n = 1 and e = 0, so the relationship n = e + 1 holds.

Induction

Let T be a tree built by the inductive step of the definition, from root node N and k smaller trees T_1 , T_2 ,..., T_k . We may assume that the statements S(Ti) hold for i = 1, 2,..., k. That is, let T_i have n_i nodes and e_i edges; then $n_i = e_i + 1$.

The nodes of T are node N and all the nodes of the T_i's. There are thus $1 + n_1 + n_2 + ... + n_k$ nodes in T. The edges of T are the k edges we added explicitly in the inductive definition step, plus the edges of the T_i's.

Hence, T has $k + e_1 + e_2 + \ldots. + e_k$ edges ... (1.3)

If we substitute $e_i + 1$ for ni in the count of the number of nodes of T we find that T has $1 + [e_1 + 1] + [e_2 + 1] + \ldots. + [e_k + 1]$ nodes ... (1.4)

Since there are k terms in (1.3), we can regroup (1.4) as

$k + 1 + e_1 + e_2 + \ldots. + ek$ ... (1.5)

This expression is exactly 1 more than the expression (1.3) that was given for the number of edges of T. Thus, T has one more node than it has edges.

Theorem 11

Every expression has an equal number of left and right parentheses.

Proof

Formally, we prove the statement S(G) about any expression G that is defined by the recursion example described earlier the numbers of left and right parentheses in G are the same.

Basis

If G is defined by the basis, then G is a number or variable. These expressions have 0 left parentheses and 0 right parentheses, so the numbers are equal.

Induction

There are three rules whereby expression G may have been constructed according to the inductive step in the definition:

- ❒ G = E + F
- ❒ G = E * F
- ❒ G = (E)

We may assume that S(E) and S(F) are true; that is, E has the same number of left and right parentheses, say n of each, and F likewise has the same number of left and right parentheses, say *m* of each. Then we can compute the numbers of left and right parentheses in G for each of the three cases, as follows:

1. If G = E + F

Then G has n + m left parentheses and n + m right parentheses; n of each come from E and m of each come from F.

2. If G = E * F

The count of parentheses for G is again n + m of each, for the same reason as in case (i).

3. If G = (E)

Then there are n + 1 left parentheses in G -- one left parenthesis is explicitly shown, and the other n are present in E. Likewise, there are n + 1 right parentheses in G; one is explicit and the other n are in E.

In each of the three cases, we see that the numbers of left and right parentheses in G are the same. This observation completes the inductive step and completes the proof.

1.4.3 Mutual Inductions

- ➠ Sometimes, we cannot prove a single statement by induction, but rather need to prove a group of statements S1(n), S2(n),...., Sk(n) together by induction on n.
- ➠ Automata theory provides many such situations. In the following example we sample the common situation where we need to explain what an automaton does by proving a group of statements, one for each state.
- ➠ These statements tell under what sequences of inputs the automaton gets into each of the states.

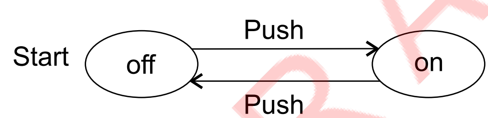
Strictly speaking, proving a group of statements is no different from proving the conjunction (logical AND) of all the statements. For instance, the group of statements $S_1(n)$, $S_2(n)$,...., $S_k(n)$ could be replaced by the single statement

$S_1(n)$ AND $S_2(n)$ AND .... AND $S_k(n)$

However, when there are really several independent statements to prove, it is generally less confusing to keep the statements separate and to prove them all in their own parts of the basis and inductive steps. This sort of proof is called mutual induction. We will illustrate the necessary steps for a mutual recursion.

Example:

Let us take the on/off switch, which can be represented as an automaton. The automaton itself is reproduced as given below.



Since on pushing the button switches the state between on and off, and the switch starts out in the off state, we expect that the following statements will together explain the operation

Push of the switch

1. S1 (n)

The automaton is in state off after n pushes if and only if n is even.

2. S2 (n)

The automaton is in state on after n pushes if and only if n is odd.

We might suppose that S 1 implies S2 and vice-versa, since we know that a number n cannot be both even and odd. However, what is not always true about an automaton is that it is in one and only one state. It happens that the automaton is always in exactly one state, but that fact must be proved as part of the mutual induction.

We give the basis and inductive parts of the proofs of statements S1(n) and S2(n) below. The proofs depend on several facts about odd and even integers:

* if we add or subtract 1 from an even integer.
* We get an odd integer
* If we add or subtract 1 from an odd integer we get an even integer.

Basis

For the basis, we choose n = 0. Since there are two statements, each of which must be proved in both directions (because S1 and S 2 are each "if-and-only-if" statements), there are actually four cases to the basis, and four cases to the induction as well.

i. [S1; If]

Since 0 is in fact even, we must show that after 0 pushes, the automaton is in state off. Since that is the start state, the automaton is indeed in state off after 0 pushes.

ii. [S1; Only-if]

The automaton is in state off after 0 pushes, so we must show that 0 is even. But 0 is even by definition of "even", so there is nothing more to prove.

iii. [S2; If]

The hypothesis of the "if" part of S2 is that 0 is odd. Since this hypothesis H is false, any statement of the form "if H then C" is true, which has discussed earlier. Thus, this part of the basis also holds.

iv. [S2; Only-if]

The hypothesis, that the automaton is in state on after 0 pushes, is also false, since the only way to get to state on is by following an arc labeled Push, which requires that the button be pushed at least once. Since the hypothesis is false, we can again conclude that the if-then statement is true.

1.5 FINITE AUTOMATA

Finite state automaton is an abstract model of a computer. It is represented in the figure. The components of the automaton are: Input Tape, Finite Control and Tape Head.

Input: String

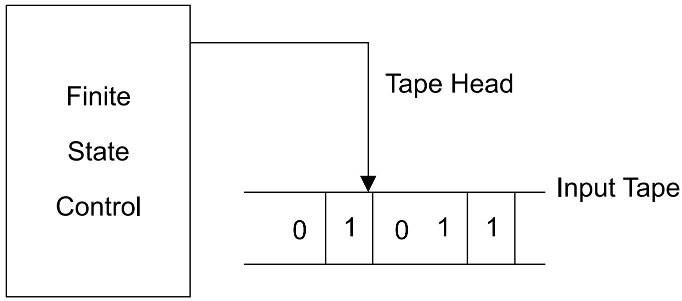


Fig. 1.1 The Working Model of a Finite Automata

Operation

String Processing (scans the string from left to right, one symbol at a time and moves from state to state) using its transition function.

Output: Yes/No (Accepted/Rejected)

1.5.1 Mathematical Representation

A Finite Automaton(FA) is represented by a 5-tuple machine.

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

* Q is a finite non-empty set of states
* Σ is a finite non-empty set of symbols
* (an alphabet)
* $\delta : Q X \Sigma \rightarrow Q$ is the transition function
* $q_0 \epsilon Q$ is the start state
* $F \epsilon Q$ is a set of final states

1. Transition function

It is a function which guides the automata in string processing. It takes two inputs (a state, a symbol) and gives one output (state). Transition function can be represented in three ways. They are,

i. *Diagrammatic representation*

Nodes and edges are used. Nodes represent the states and edges represent the moves. The labels of the edges represent the processing symbols. There are two types of nodes: a) single circled node indicating non-final (non-accepting) state; b) double circled node indicating final state.

ii. *Tabular representation*

It consists of Rows and columns. Rows indicate state and columns indicate symbol. The entries of the table indicate the output state. The arrow and star symbols are used to point out the starting and final states respectively.

iii. *Functional representation*

The name of the function is δ. The input parameters are q,a.

* Where q is a state and a is a symbol. The function returns a state p.

Example:

The automata of the language L={*w* | *w* contains ab}

2. Diagrammatic Representation - Transition diagram

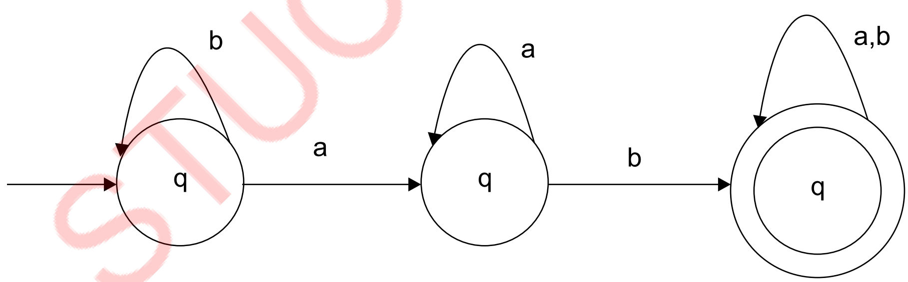


Fig. 1.2 The transition diagram of FA for the language L={w | w contains ab}

δ :

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

Σ = {a,b}

$q_0 = q_0$

F = q_2

Table 1.2 The transition table of FA for the language L={w | w contains ab}

Δ	a	b
$\rightarrow q_0$	q_1	q_0
q_1	q_1	q_2
* q	q	q

3. Functional Representation - Transition functions

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

1.5.2 Types of Finite Automata

1. Deterministic

- If there is exactly one output state in every transition function of an automata, then the automata is called Deterministic finite Automata (DFA)
- A Deterministic finite automaton (DFA) is represented by a 5-tuple machine

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

  * Q is a finite non-empty set of states
  * Σ is a finite non-empty set of symbols
  * (an alphabet)
  * $\delta : Q X \Sigma \rightarrow Q$ is the transition function
  * $q_0 \in Q$ is the start state
  * $F \in Q$ is a set of final states

2. Non-Deterministic

- If there is zero or more output states in any of the transition functions of an automata then that automata is called Non-Deterministic Finite Automata (NFA).
- NFA is the preliminary form of a machine, which can be easily constructed using the basic constraints of a language.
- Then it can be converted into DFA using subset construction method and finally minimization methods are used to reduce the size of the machine.
- A Non-Deterministic finite automaton (NFA) is represented by 5-tuples.

i.e. M = (Q, Σ, δ, q0, F)

* Q is a finite non-empty set of states
* Σ is a finite non-empty set of symbols

  (an alphabet)

* δ: $Q X \Sigma \rightarrow 2^Q$ (subset of Q) is the transition function
* $q_0 \in Q$ is the start state
* $F \in Q$ is a set of final states

3. ε-NFA

- If there is a transition for ε symbol in NFA , then the automata is called ε-NFA. An ε-Non-Deterministic finite automaton (NFA) is represented by 5-tuples.

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

* Q is a finite non-empty set of states
* Σ is a finite non-empty set of symbols

  (an alphabet)

* $\delta : QX \ (\Sigma \ \{ \varepsilon \}) \rightarrow 2^Q$ (subset of Q) is the transition function
* $q_0 \in Q$ is the start state
* $F \in Q$ is a set of final states

1.5.3 Language of an Automata

1. L(M)

- The language of machine M
- Set of all strings machine M accepts

2. L(DFA)

$\{w | \hat{\delta}(q_0, w) = p \in F\}$

Where,

$\hat{\delta}(q_0, w)$ is an extended transition function that takes a state q_0 and a string w and returns a state p which is in F = Regular language.

3. L(NFA)

$\{w | \hat{\delta}(q_0, w) \cap F \neq \varphi)$ - Regular language.

1.6 DETERMINISTIC FINITE AUTOMATA(DFA)

Deterministic finite Automata is a definite model of computation where there is single output for every symbol from every state. The transition table of a DFA will be complete and unambiguous. There would not be any empty entry and multiple entries.

1.6.1 String Processing

- An automata processes the given string and gives Yes/No as the output.
- During string processing, the symbols in the given string are processed one by one, from left to right according to the moves defined by the transition functions of the automata.
- A set of transition function defines an automata.
- During string processing, automata selects the transition function whose input matches with the current state (state and symbol) and performs a move to output state.

1.6.2 String Processing in DFA

Problem	1.1

Let M = (Q, Σ, δ, q_0, F) where Q={ q_0 , q_1 ,q_2 }, Σ,={a,b} F={q_2}

δ	a	b
→q_0	q_1	q_0
q_1	q_1	q_2
* q_2	q_2	q_2

➠ Show that the string w= bbabb is accepted by the given FA, M.

$$\hat{\delta}(q_0,\underline{b}babb) = \hat{\delta}(q_0,\underline{b}abb)$$
$$=\hat{\delta}(q_0,\underline{a}bb)$$
$$=\hat{\delta}(q_1,\underline{b}b)$$
$$= \delta(q_2,\underline{b})$$
$$= q_2 \in F$$

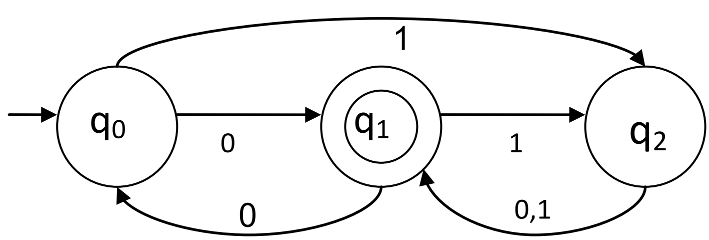
➠ There is a path from starting state to final state.

$$q_0 \xrightarrow{b} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{b} q_2 \in F$$

Therefore the given string is accepted.

Problem	1.2

Consider the following DFA. Compute $\hat{\delta}(q_0,1101)$



$$\hat{\delta}(q_0,1101)=(q_2,101)\ =(\ q_1,01)=\ (q_0,1)=q_2 \in F$$

So the string is not accepted.

1.7 NON-DETERMINISTIC FINITE AUTOMATA(NFA)

NFA is the simple and initial model of computation .Constructing Automata to recognize a Language includes the following steps:

- Design an NFA
- Convert NFA to DFA
- Minimize the DFA

1.7 .1 Designing NFA for a language

It is very easy to design NFA for a language by considering the common (compulsory) part of the strings in a given language. There are two types of NFAs.

- NFA without ε-Transitions
- ε-NFA

➠ Designing NFA without ε-Transitions for a language

Problem	**1.3**

Design an NFA for the following finite languages over the alphabet {a,b}

a. L={ε}

b. L={a}

c. L={b}

d. L={a,b}

e. L={aa,ab}

f. L={aba,abb,aaa}

Solutions:

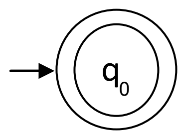
a. L={ε}

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

Where $Q=\{q_0\}$

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

δ : Transition diagram



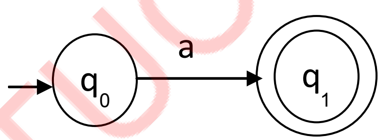
b. L={a}

NFA $M=(Q, \Sigma, \delta, q_0, \{q_1\})$

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

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

δ : Transition diagram



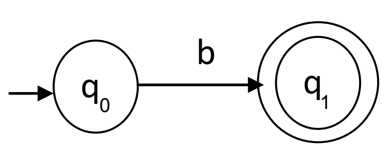
c. L={b}

NFA $M=(Q, \Sigma, \delta, q_0, \{q_1\})$

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

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

δ : Transition diagram



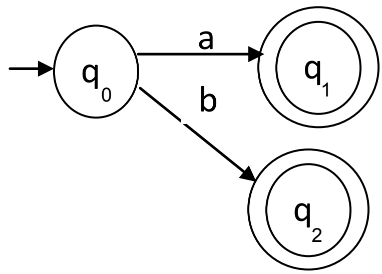
d. L={a,b}

NFA M=(Q, Σ, δ, q_0, {q_1, q_2})

Where Q={ q_0, q_1, q_2}

Σ={a,b}

δ : Transition diagram



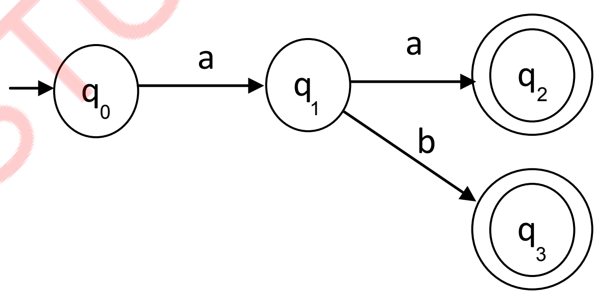
e. L={aa,ab}

NFA M=(Q, Σ, δ, q_0, {q_2, q_3})

Where Q={ q_0, q_1, q_2, q_3}

Σ={a,b}

δ : Transition diagram



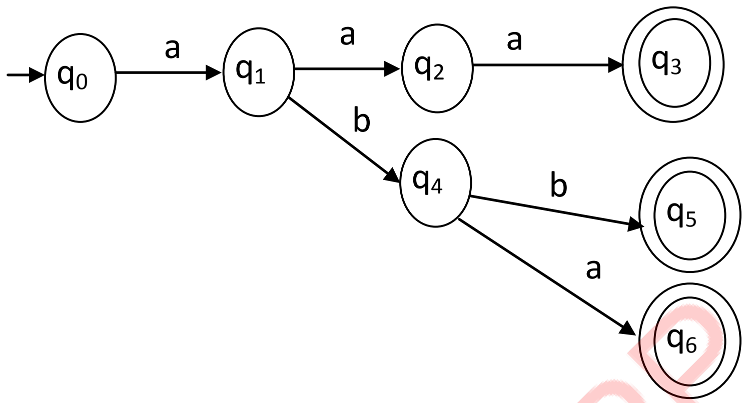
f. L={aba,abb,aaa}

NFA M=(Q, Σ, δ, q_0, {q_3, q_5, q_6})

Where Q={ q_0, q_1, q_2, q_3, q_4, q_5, q_6}

Σ={a,b}

δ : Transition diagram



Problem	1.4

Design an NFA without ε-Transitions for the following infinite languages over the alphabet {a,b}.

a. The set of all strings ending in aa (L={*w*| *w* ends in aa})

b. The set of all strings with the substring aba(L={*w*| *w* has substring aba}).

c. The set of all strings beginning with bb (L={*w*| *w* begins with bb)}).

d. The set of all strings with even number of a's (L={*w*| *w* has even number of a's}).

e. The set of all strings with even number of b's(L={*w*| *w* has even number of b's }).

f. The set of all strings with odd number of a's (L={*w*| *w* has odd number of a's}).

g. The set of all strings with odd number of b's (L={*w*| *w* has odd number of b's}).

h. The set of all strings whose third symbol from the right end is b (L={*w*| *w*'s third symbol from the right end is b}).

i. The set of all strings whose third symbol from the left end is b (L={*w*| *w*'s third symbol from the left end is b}).

Solutions:

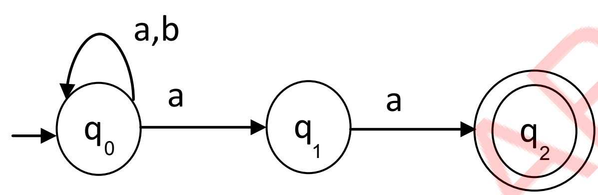
a. The set of all strings ending in aa.

NFA M=(Q, Σ, δ, q_0, {q_2})

Where Q={ q_0, q_1, q_2}

Σ={a,b}

δ : Transition diagram



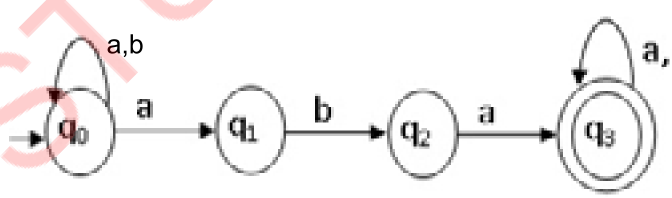
b. The set of all strings with the substring aba.

NFA M = (Q, Σ, δ, q_0, {q_3})

Where Q={ q_0, q_1, q_2, q_3}

Σ={a,b}

δ : Transition diagram



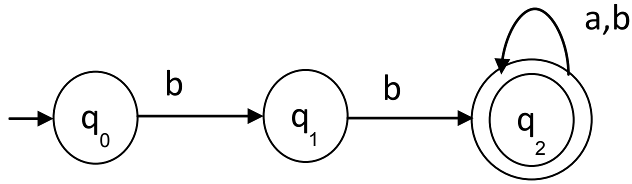
c. The set of all strings beginning with bb.

NFA M=(Q, Σ, δ, q_0, {q_2})

Where Q={ q_0, q_1, q_2}

Σ={a,b}

δ : Transition diagram



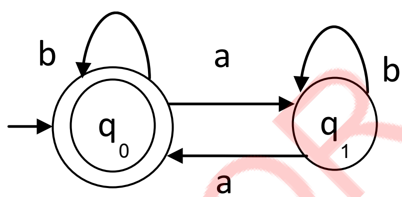
d. The set of all strings with even number of a's.

NFA M=(Q, Σ, δ, q_0, {q_0})

Where Q={ q_0, q_1}

Σ={a,b}

δ : Transition diagram



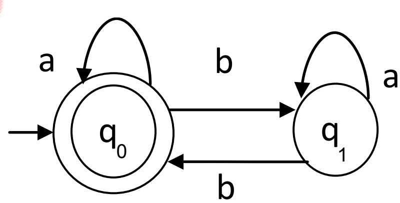
e. The set of all strings with even number of b's.

NFA M=(Q, Σ, δ, q_0, {q_0})

Where Q={ q_0, q_1}

Σ={a,b}

δ : Transition diagram



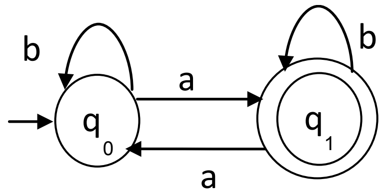
f. The set of all strings with odd number of a's.

NFA M=(Q, Σ, δ, q_0, {q_1})

Where Q={ q_0, q_1}

Σ={a,b}

δ : Transition diagram



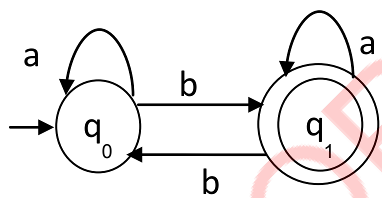
g. The set of all strings with odd number of b's.

NFA $M=(Q, \Sigma, \delta, q_0, \{q_1\})$

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

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

δ : Transition diagram



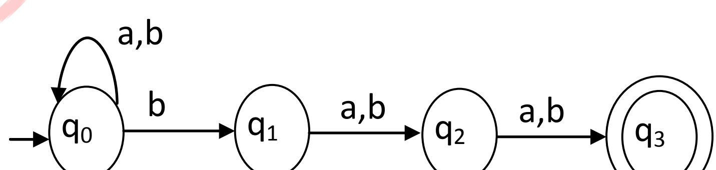
h. The set of all strings whose third symbol from the right end is b.

NFA $M=(Q, \Sigma, \delta, q_0, \{q_3\})$

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

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

δ : Transition diagram



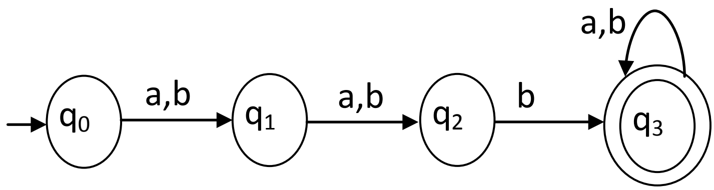
i. The set of all strings whose third symbol from the left end is b.

NFA $M=(Q, \Sigma, \delta, q_0, \{q_3\})$

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

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

δ : Transition diagram



1. Automata for $\overline{L}$

If the automata is given for a language $\bar{L}$, then the automata for can be easily constructed by changing all the non-final states to final states and final states to non-final states.

Given:

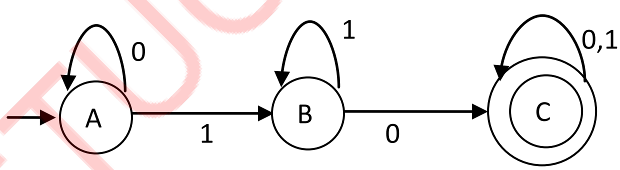
The FA of L= {w | w consists of 10 as substring}

M(L) =(Q, Σ, δ, A, {C})

Where Q={ A, B ,C }

Σ={a,b}

δ : Transition diagram

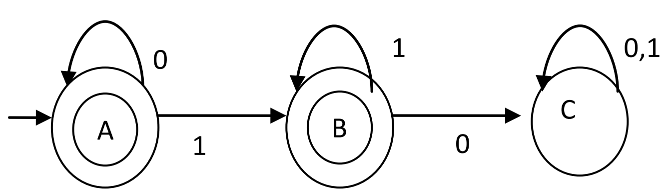


M($\overline{L}$) = (Q, Σ, δ, A, {A, B})

Where Q={ A, B ,C }

Σ={a,b}

δ : Transition diagram



2. Automata for $L_1 \cap L_2$

The intersection of two regular languages can be constructed by taking Cartesian product of states.

Let, $M(L_1) = (Q_1, \Sigma, \delta_1, q_1, F_1)$

$M(L_2) = (Q_2, \Sigma, \delta_2, q_2, F_2)$

Then $M(L_1 \cap L_2) = (Q_2 \times Q_2, \Sigma, \delta, (q_1, q_2), F_2 \times F_2)$

Problem	1.5

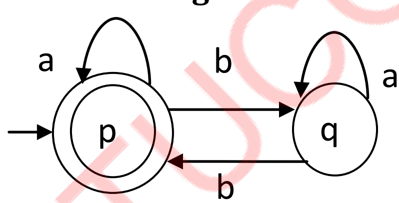
Let L1 = The set of all strings with even number of b's.

NFA $M(L_1) = (Q, \Sigma, \delta_1, p, \{p\})$

Where Q={p, q }

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

δ_1 : Transition diagram



Problem	1.6

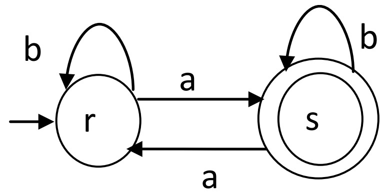
Let L_2 = The set of all strings with odd number of a's.

NFA $M(L_2) = (Q, \Sigma, \delta 2, r, \{s\})$

Where Q={ r, s }

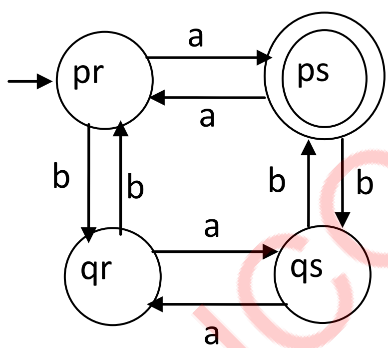
$\Sigma = \{a,b\}$

δ_2 : Transition diagram



Then, M($L_1 \cap L_2$) = ({pr, ps, qr,qs}, {a,b}, δ, pr, ps)

$$\delta(pr,a) = (\delta_1(p,a), \delta_2(r,a))$$
$$= (p,s)$$
$$\delta(pr,b) = (\delta_1(p,b), \delta_2(r,b))$$
$$= (q,r)$$



1.7.2 String Processing in NFA

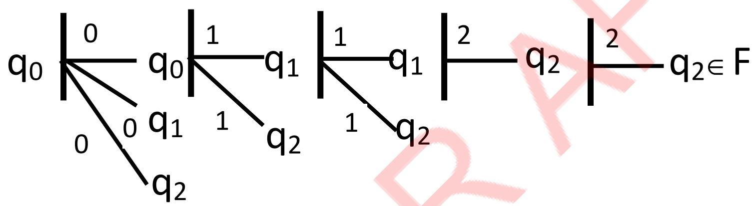
Problem	1.7

For the NFA M given in the following table, test whether the strings 01122, 1221 are accepted by M.

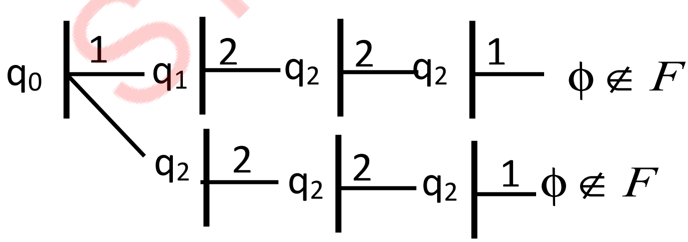
δ	0	1	2
→*q0	{q0, q1, q2 }	{q1, q2 }	{q2 }
*q1	Φ	{q1, q2 }	{q2 }
*q2	Φ	Φ	{q2 }

$$i.\ \ \delta(q_0,\underline{0}1122)=\hat{\delta}(\{q_0,q_1,q_2\},\underline{1}122)$$
$$=\hat{\delta}(\{\delta(q_0,1)\cup\delta(q_1,1)\cup\delta(q_2,1)\},122)$$
$$=\hat{\delta}(\{q_1,q_2\},\underline{1}22)$$
$$=\hat{\delta}(\{q_1,q_2\},\underline{2}2)$$
$$=\hat{\delta}(\{q_2\},\underline{2})$$
$$=q_2\in F$$

There is at least one path from the starting state to final state. Therefore the given string is accepted.



$$ii.\ \ \hat{\delta}(q_0,\underline{1}221)=\hat{\delta}(\{q_1,q_2\},\underline{2}21)$$
$$=\hat{\delta}(\{\delta(q_1,2)\cup\delta(q_2,2)\},21)$$
$$=\hat{\delta}(\{q_2\},\underline{2}1)$$
$$=\hat{\delta}(\{q_2\},\underline{1})$$
$$=\phi\notin F$$



There is no even a single path from starting state to final state. Therefore the given string is not accepted.

1.7.3 Equivalence of NFA and DFA (Converting NFA to DFA)

Theorem 12

A Language L is accepted by some DFA if and only if L is accepted by some NFA.

Proof by induction

The "if" part : If L is accepted by some NFA then L is accepted by some DFA. If $D=\{Q_D, \Sigma, \delta_D, \{q_0\}, F_D\}$ is the DFA constructed from NFA, $N=\{Q_N, \Sigma, \delta_N, q_0, F_N\}$ by the subset construction ,then L(D) = L(N).

Proof

To prove by induction on $|\omega| \hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$ ... (1.6)

Observe that each of the $\hat{\delta}$ functions returns a set of states from Q_N, but $\hat{\delta}_D$ interprets this set as one of the states of Q_D (which is the power set of Q_N), while interprets this set as a subset of Q_N.

Basis

Let $|w| = 0$; that is, $w=e$. By the basis definitions of $\hat{\delta}$ for DFA's and NFA's, both $\hat{\delta}_D(\{q_0\}, \varepsilon)$ and $\hat{\delta}_N(q_0, \varepsilon)$ are $\{q_0\}$

Induction

Let $|w| = n+1$, and assume the statement for length n. Break w as $w=xa$, where a is the final symbol of w.

By the inductive hypothesis,

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

Let both these sets of N's states be $\{P_1, P_2, \ldots P_k\}$ .i.e.

$$\hat{\delta}_D(\{q_0\}, x) = \hat{\delta}_N(q_0, x) = \{p_1, p_2, \ldots p_k\} \quad \ldots (1.7)$$

The inductive part of the definition of for the NFA's say that

$$\hat{\delta}_N(q_0, w) = \bigcup_{i=1}^{k} \delta_N(p_i, a) \qquad \text{... (1.8)}$$

The subset construction, on the other hand , says that

$$\delta_D(\{p_1, p_2, ...p_k\}, a) = \bigcup_{i=1}^{k} \delta_N(p_i, a) \qquad \text{... (1.9)}$$

From (1.7) and (1.9), the inductive part of the definition of $\hat{\delta}$ for DFA is written as:

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

$$= \delta_D(\{p_1, p_2, ...p_k\}, a) = \bigcup_{i=1}^{k} \delta_N(p_i, a) \qquad \text{... (1.10)}$$

Thus, equations (1.8) and (1.10) demonstrate that

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

When we observe that D and N both accept w if and only if $\hat{\delta}_D(\{q_0\}, w)$ or $\hat{\delta}_N(q_0, w)$ respectively, contain a state in F_N.

Hence, L(D) = L(N) is proved.

The "only if" part

If L is accepted by some DFA then L is accepted by some NFA.

We have only to convert a DFA into identical NFA.Put intuitively, if we have the transition diagram for a DFA, we can also Interpret it as the transition diagram of an NFA, which happens to have exactly one choice of transition in any situation.

More formally, let D={Q, Σ, δ_D, q_0,F} be a DFA. Define N={Q, Σ, δ_N, q_0,F} to be the equivalent NFA.

Where, δ_N is defined by the rule:

If $\delta_D(q, a) = p$ then $\delta_N(q, a) = \{p\}$

It is then easy to show by induction on $|w|$, that if $\hat{\delta}_D(q_0, w) = p$ then $\hat{\delta}_N(q_0, w) = \{p\}$

As a consequence, ω is accepted by D if and only if it is accepted by N; i.e ., L(D)=L(N).

Subset construction method (with ‘Lazy Evaluation’) is used to convert NFA to DFA. In this method the transition functions are generated only for reachable states.

Method 1

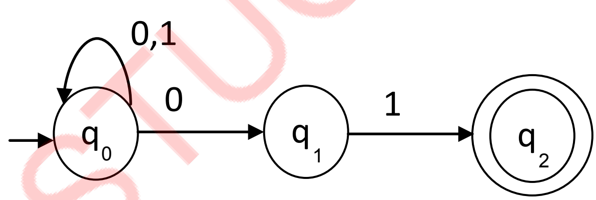
Steps

1. Include the starting state of NFA (q0) in DFA as starting state of DFA.
2. Find the transition for all the symbols from q0
3. If the output state is new state, include it in DFA and find the transition for all the symbols from that state.
4. Repeat step3 until there are no more new states.
5. The state which includes final state of NFA is the final state of DFA.

Problem	1.8

Construct the DFA for the L={w|w ends in 01}

Transition Diagram of NFA



Transition Table of NFA

δ_D	0	1	2
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$	$\{q_2\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_2\}$
* $\{q_0, q_2\}$	$\{q0, q_1\}$	$\{q_0\}$	$\{q_2\}$

Step 1:

Include q_0

Step 2:

Find transitions for 0,1 from q_0.

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

$\delta(q_0, 1) = \{q_0\}$ - Existing state

Step 3:

Find transitions for 0,1 from new state.

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

$$= \{q_0, q_1\} \cup \varphi = \{q_0, q_1\} \quad \text{Existing state}$$

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

$$= \{q_0\} \cup \{q_2\} = \{q_0, q_2\} \quad \text{New state}$$

Step 4:

Repeat step 3 for new state (s).

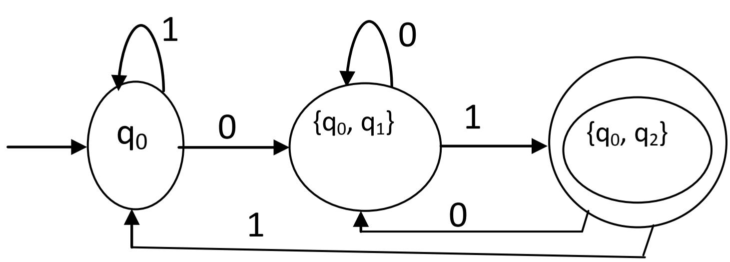
$$\delta(\{q_0, q_2\}, 0) = \delta(q_0, 0) \cup \delta(q_2, 0)$$

$$= \{q_0, q_1\} \cup \varphi = \{q_0, q_1\} \quad \text{Existing state}$$

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

$$= \{q_0\} \cup \varphi\} = \{q_0\} \quad \text{New state}$$

Transition Diagram of DFA



Method 2

Input: Transition table of NFA

Output: Transition table of DFA

Steps

1. Draw the transition table for NFA (if not given)
2. Copy the first row of NFA table (transition function of start state) to DFA table.
3. The entries are considered as states of DFA.
4. If there is any new state, find the transition function for that new state using the following formula:

$$\delta_D(\{q_1,...,q_k\},a) = \bigcup_{i=1}^{k} \delta_N(q_i,a)$$

5. Continue Step 4 until no more new states.

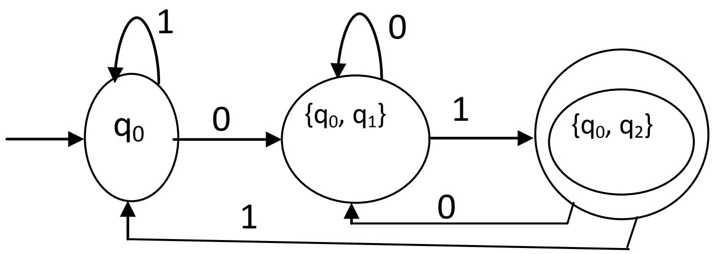
Transition Table of DFA

δ_D	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0,q_2\}$
* $\{q_0,q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$

Procedure

- Copy the first row. $\{q_0, q_1\}$ is the new state.
- Union of q_0 row and q_1 row. $\{q_0, q_2\}$ is the new state.
- Union of q_0 row and q_2 row.
- No more new states. So Stop

Transition Diagram of DFA



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

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

$q_0 = q_0$

$F = \{q_0,q_2\}$

Problem	1.9

Consider the following NFA. Convert it into DFA.

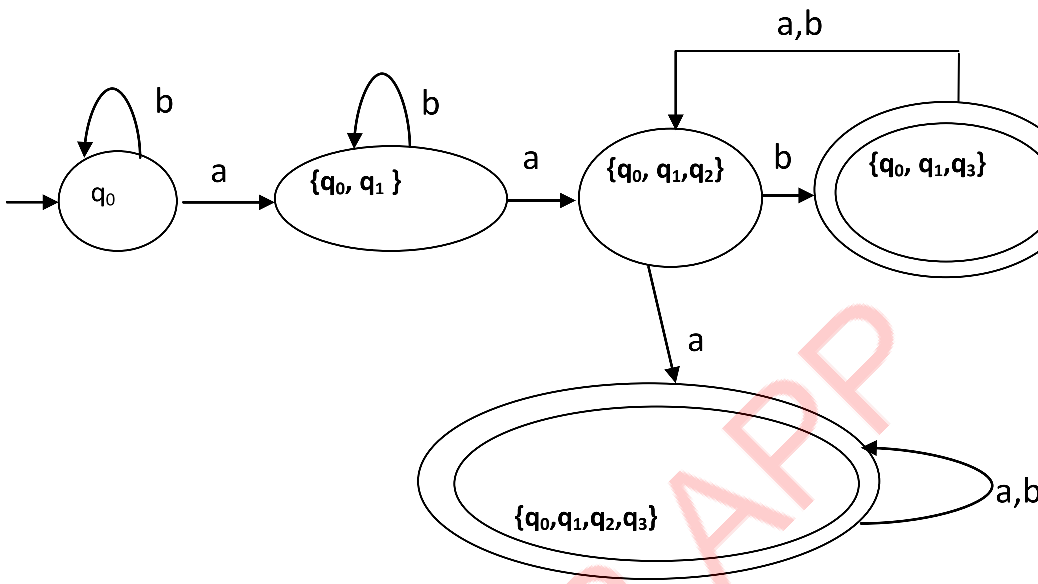
Transition Table of NFA

δ_N	a	b
$\rightarrow q_0$	$\{q_0,q_1\}$	$\{q_0\}$
q_1	q_2	q_1
q_2	q_3	q_3
$* q_3$	-	q_2

Procedure

- Copy the first row.
- Identify the new state.
- Find the transition for new state using Union operation.
- Stop, if no more new states.

Transition Diagram of DFA



Problem 1.10

Convert to the DFA the following NFA.

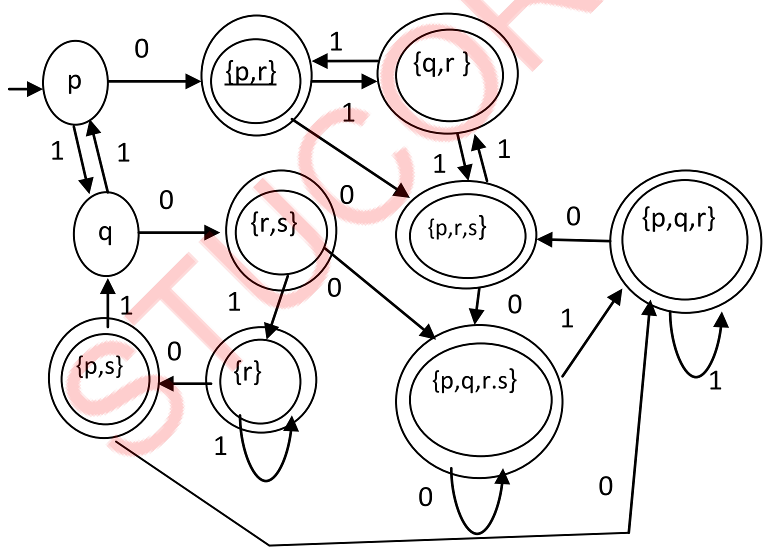
δ_N	0	1
→p	{ p,r }	{q}
q	{ r,s }	{ p}
*r	{ p,s}	{ r }
* s	{ q,r }	-

Transition Table of DFA

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

*{ p,r }	{ p,r,s }	{ q,r }
*{ r,s }	{ p,q,r,s }	{ r}
*{ p,r,s }	{ p,q,r,s }	{ q,r }
*{ r}	{ p,s}	{ r }
*{ p,q,r,s }	{ p,q,r,s }	{ p,q,r }
*{ p,s}	{ p,q,r }	{ q}
*{ p,q,r }	{ p,r,s }	{ p,q,r }
*{ q,r }	{ p,r,s }	{ p,r }

Transition Diagram of DFA



Problem	1.11

Convert the following NFA to a DFA and informally describe the language it accepts.

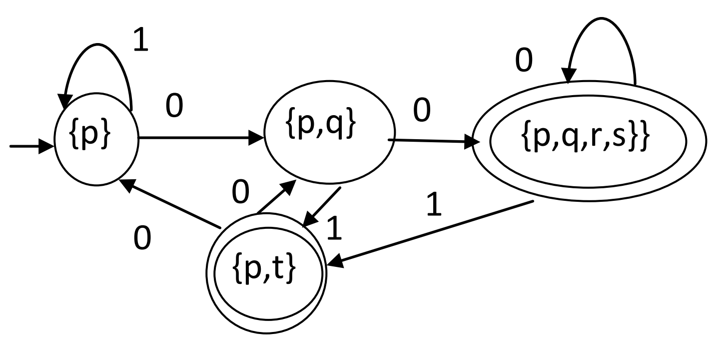
Transition table of given NFA

δ_N	0	1
→p	{ p,q }	{ p}
q	{ r,s }	{ t}
r	{ p,r}	{ t }
* s	-	-
*t	-	-

Transition table of DFA

δ_D	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}

Transition Diagram of DFA

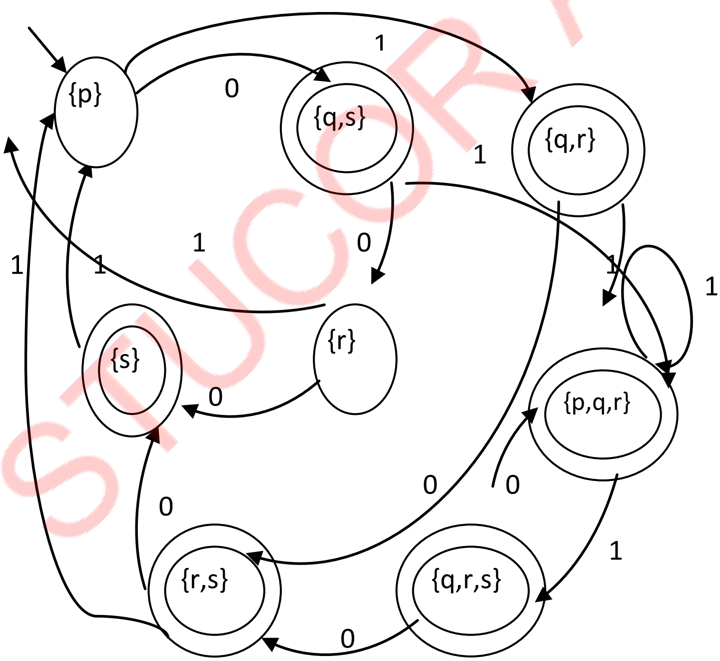


Problem	1.12

Convert to a DFA the following NFA.

	0	1
→p	{q,s}	{q}
*q	{r}	{q,r}
r	{s}	{p}
*s	-	{p}

Transition Diagram of DFA



Language of DFA

➠ The language of a DFA is defined by,

$$L(DFA)=\{w \mid \hat{\delta}(q_0, w) \text{ is in } F\}$$

➠ And the language of a NFA is defined by,

$$L(NFA) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \varphi\}$$

* Where q_0 is the start state
* F is the set of final states and
* w is a string.
* L(DFA) and L(NFA) are called Regular Languages.

1.8 FINITE AUTOMATA WITH EPSILON TRANSITIONS

Finite Automata with Epsilon transitions is also called as ε-NFA . It contains epsilon edges. In transition table a column is allocated for epsilon and it gives the output for epsilon input.

☞ A Non-Deterministic finite automaton with ε- Transitions (NFA) is represented by 5-tuples.

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

* Q is a finite non-empty set of states.
* Σ is a finite non-empty set of symbols (an alphabet)
* $\delta : Q X \Sigma \cup \{\varepsilon\} \rightarrow 2^Q$ is the transition function
* $q_0 \in Q$ is the start state
* $F \in Q$ is a set of final states

Transition Table of ε-NFA

δ_N	ε	a	b	c
→p	Φ	{p}	{q}	{r}
q	{p}	{q}	{r}	Φ
*r	{q}	{r}	Φ	{p}

ε-Closure

Epsilon closure of a state is the set of all states that are reachable by following the transition function from the given state through ε edge.

Problem	1.13

Consider the ε-NFA. Compute ε-Closure for each state.

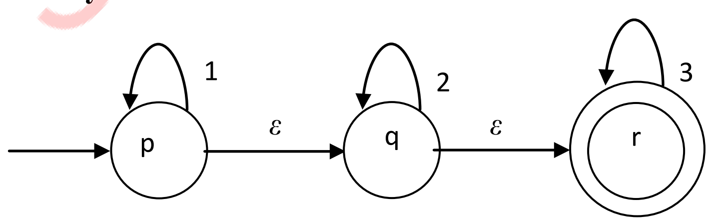
δ_N	ε	0	1	2
$\rightarrow q_0$	q_1	q0	Φ	Φ
q_1	q_2	Φ	q_1	Φ
$*q_2$	Φ	Φ	Φ	q_2

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

1.8.1 Designing an ε-NFA or NFA with ε-Transitions

Problem	1.14

Design an ε-NFA for the language which consists of strings that has 1's followed by 2's followed by 3's.

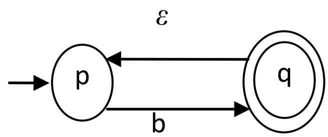


Q = {p,q,r}

Σ = {1,2,3}

Problem	1.15

Design an ε-NFA for the language b+.

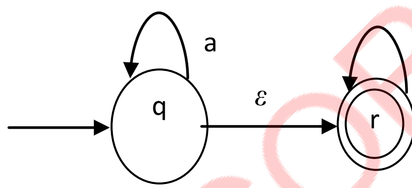


Q = {p,q}

Σ = {b}

Problem	1.16

Design an ε-NFA for the language which consists of strings with all a's followed by all b's.



Q = {q,r}

Σ = {a,b}

1.8.2 String Processing in ε-NFA

Problem	1.17

For the ε-NFA M given in the following table, test whether the strings aabccand abba are accepted by M.

δ	ε	a	b	c
→p	{q}	{p}	Φ	Φ
q	{r}	Φ	{q}	Φ
*r	Φ	Φ	Φ	{r}

Step 1:

Compute ε-Closure [states that can be reached by traveling along zero or more ε arrows] for all states .

- ε-Closure (p) = {p,q,r} $\left[\hat{\delta}(p,\varepsilon)\right]$
- ε-Closure (q) = {q,r} $\left[\hat{\delta}(q,\varepsilon)\right]$
- ε-Closure (r) = {r} $\left[\hat{\delta}(r,\varepsilon)\right]$

Step2:

Start with ε-closure (p)= {p,q,r}

Where, p is the starting state of given ε –NFA.

1. (p)= {p,q,r}

$$
\begin{aligned}
&\hat{\delta}(\{p,q,r\},\underline{a}abcc)\\
&\quad = \hat{\delta}(\varepsilon-closure(\delta(p,a)\cup\delta(q,a)\cup\delta(r,a)),\underline{a}bcc)\\
&\quad = \hat{\delta}(\varepsilon-closure(p),abcc)\\
&\quad = \hat{\delta}(\{p,q,r\},\underline{a}bcc)\\
&\quad = \hat{\delta}(\varepsilon-closure(\delta(p,a)\cup\delta(q,a)\cup\delta(r,a)),bcc)\\
&\quad = \hat{\delta}(\{p,q,r\},\underline{b}cc)\\
&\quad = \hat{\delta}(\varepsilon-closure(\delta(p,b)\cup\delta(q,b)\cup\delta(r,b)),\underline{c}c)\\
&\quad = \hat{\delta}(\{q,r\},\underline{c}c)\\
&\quad = \hat{\delta}(\varepsilon-closure(\delta(q,c)\cup\delta(r,c)),\underline{c})\\
&\quad = \hat{\delta}(\{q,r\},\underline{c})\\
&\quad = r\in F
\end{aligned}
$$

✻ Therefore the given string is accepted.

2. *w=abba*

$$
\begin{aligned}
&\hat{\delta}(\{p,q,r\},\underline{a}bba)\\
&\qquad = \hat{\delta}(\varepsilon - closure(\delta(p,a)\cup\delta(q,a)\cup\delta(r,a)),\underline{b}ba)\\
&\qquad = \hat{\delta}(\varepsilon - closure(p),bba)\\
&\qquad = \hat{\delta}(\{p,q,r\},\underline{b}ba)\\
&\qquad = \hat{\delta}(\varepsilon - closure(\delta(p,b)\cup\delta(q,b)\cup\delta(r,b)),ba)\\
&\qquad = \hat{\delta}(\{q,r\},\underline{b}a)\\
&\qquad = \hat{\delta}(\varepsilon - closure(\delta(q,b)\cup\delta(r,b)),\underline{a})\\
&\qquad = \hat{\delta}(\{q,r\},\underline{a})\\
&\qquad = \varphi \notin F
\end{aligned}
$$

✽ Therefore the given string is not accepted.

1.8.3 Equivalence of ε-NFA and DFA.

An ε-NFA can be converted into DFA. The subset construction method (with ‘Lazy Evaluation’) is used to convert ε-NFA to DFA. In this method the transition functions are generated only for reachable states.

Input: Transition table of ε-NFA

Output: Transition table of DFA

Theorem

A language L is accepted by some -NFA if and only if L is accepted by some DFA.

Proof

1. If part: If the L is accepted by some DFA then L is accepted by some -NFA

Suppose L=L(D)for some DFA. Turn D into an ε-NFA by adding transitions $\delta(q,\varepsilon)=\varphi$ for all states q of D. Technically we must also convert the transitions of D on input symbols, example, $\delta_D(q,a)=p$, into an NFA-transition to the set containing only p, that is, $\delta_E(q,a)=\{p\}$

Thus, the transitions of E and D are the same , but E explicitly states that there are no transitions out of any state on ε.

2. Only -If part: If the L is accepted by some -NFA then L is accepted by some DFA.

Let E = {Q_E, Σ, δ_E, q_0, F_E} be an ε -NFA. Apply the modified subset construction to produce the DFA.

D = {Q_D, Σ, δ_D, q_D, F_D}

- We need to show that L(D) = L(E), and we do so by showing that the extended transition functions of E and D are the same.
- Formally, we show $\hat{\delta}_E(q_0, w) = \hat{\delta}_D(q_D, w)$ by induction on the length of ω .

Basics

- If $|w|$ then w=ε.
- We know $\hat{\delta}_E(q_0, \varepsilon) = ECLOSURE(q_0)$
- We also know that $q_D = ECLOSURE(q_0)$, because that is how the start state of D is defined.
- Finally, for a DFA, we know that $\hat{\delta}(p, \varepsilon) = p$ for any state p, so in particular $\hat{\delta}_D(q_D, \varepsilon) = ECLOSURE(q_0)$.
- We have thus proved that $\hat{\delta}_E(q_0, \varepsilon) = \hat{\delta}_D(q_D, \varepsilon)$.

Induction

- Suppose $w=xa$.
  - Where, a is the final symbol of w and assume that the statement holds for x.
- That is, $\hat{\delta}_E(q_0, x) = \hat{\delta}_D(q_D, x)$.
- Let both these sets of states be {p_1, p_2,....p_k}. By the definition of $\hat{\delta}$ for ε-NFA's, we compute $\hat{\delta}_E(q_0, w)$ by,

i. Let $\{r_1, r_2, \ldots r_m\}$ be $\bigcup_{i=1}^{k} \delta_E(p_i, a)$.

ii. Then $\hat{\delta}_E(q_0, w) = \bigcup_{j=1}^{m} ECLOSURE(r_j)$

* If we examine the construction of DFA D in the modified subset construction, we see that $\delta_D(\{p_1, p_2, \ldots p_k\}, a)$ is constructed by the same above two steps (i) and (ii).

* Thus, $\hat{\delta}_D(q_D, w)$, which is $\delta_D(\{p_1, p_2, \ldots p_k\}, a)$ is the same set as $\hat{\delta}_E(q_0, w)$.

* We have now proved that $\hat{\delta}_E(q_0, w) = \hat{\delta}_D(q_D, w)$ and completed the inductive part.

Steps to convert ε-NFA to DFA

a. Compute the ε-Closure for each state.

b. Draw the transition table for ε-NFA (if not given)

c. Start state of DFA is ε-Closure(q_0)

- Where q_0 is the start state of ε-NFA .

d. Find the transition function for ε-Closure(q_0).

e. The entries are considered as states of DFA.

f. If there is any new state, find the transition function for that new state using the following formula:

$$\delta_D(\{q_1, \ldots, q_k\}, a) = \bigcup_{i=1}^{k} \varepsilon - closure(\delta_N(q_i, a))$$

g. Continue the above step 'f' until no more new states.

1.8.4 Applications and Limitations of FA

1. Applications of FA

i. Text Search

a. **News Analyst** - Searches on-line news

b. **Shopping robot** – Searches current prices charged for an item

c. **Amazon.com** - Search some keywords

d. **Lexical analyzer of a compiler** - Identifies the token

* Verifying the working of a physical system

* Design and construction of Softwares

ii. Advantages of Finite set of states in Automata

- Implement a system with a fixed set of resources
- Implementing a system within a hardware circuit
- Complementing a system using software with a finite set of codes.

2. Limitations of FA

- Some languages are not regular – i.e. we cannot construct FA

Example:

* $B = \{0^n 1^n \mid n \geq 0\}$ is NOT regular!

* $L=ww^R$

* $L=WW$

* $L=WCWR$

* $C = \{ w \mid w$ has equal number of 1s and 0s$\}$

1.8.5 Complex Problems

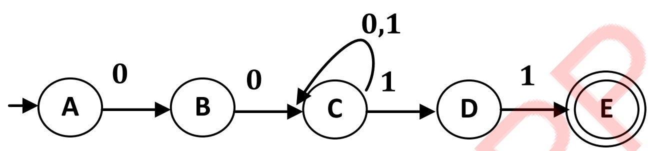
1. Design a NFA that accepts set of all strings that begins with 00 and ends with 11. Convert it into DFA.

Analysis

- Here we have two parts:
  - Begins with string1

- ❒ Ends with string2

- Let string1 be considered as s1s2 and string 2 be considered as s3s4 where s1,s2,s3 and s4 are substrings.
- For all s2 and s3, if s2≠s3, we can easily construct the NFA.
- In this problem there is no such s2 and s3 where s2=s3. Therefore we can construct the NFA in one step as follows:



- The DFA of this machine is given below:

δ_D	0	1
→ {A}	{B}	-
{B}	{C}	-
{C}	{C}	{C,D}
{C,D}	{C}	{C,D,E}
* {C,D,E}	{C}	{C,D,E}

Note: It is difficult to draw the NFA for the following languages wheres2=s3.

- Set of all strings that begins with 01 and ends with 11 [s2=1]
- Set of all strings that begins with 01 and ends with 10 [s2=1]
- Set of all strings that begins with 01 and ends with 01 [s2=01]
- Set of all strings that begins with 10 and ends with 10 [s2=10]
- Set of all strings that begins with 00 and ends with 00 [s2=00]
- Set of all strings that begins with 11 and ends with 11 [s2=00]

For these kinds of problems we can use the intersection property of regular languages.

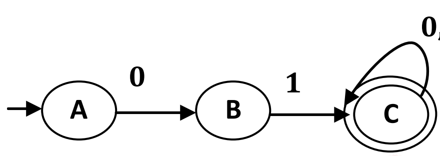
Problem	1.18

Design a DFA that accepts set of all strings that begins with 01 and ends with 11.

➠ There are three steps, that are given below.

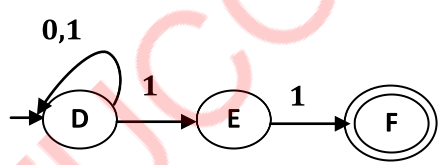
Step 1:

Design a DFA that accepts set of all strings that begins with 01



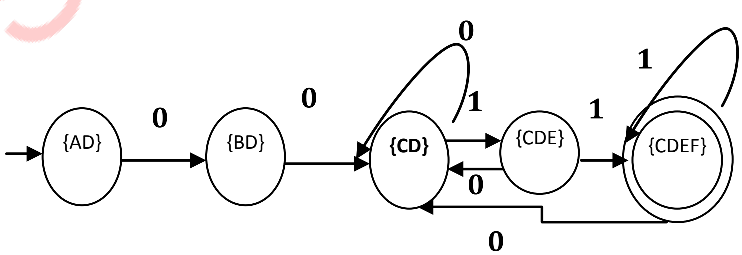
Step 2:

Design a DFA that accepts set of all strings that ends with 11.



Step 3:

Intersection between two DFAs (Lazy Evaluation-processing only reachable nodes)



1.8.6 PROBLEMS

1. Consider the following ε-NFA. Covert it into DFA

Transition Table of ε-NFA

δ_N	ε	a	b	c
→p	Φ	{p}	{q}	{r}
q	{p}	{q}	{r}	Φ
*r	{q}	{r}	Φ	{p}

Step 1:

Compute ε-Closure [states that can be reached by traveling along zero or more ε arrows] for all states.

- ε-Closure (p) = {p} $\left[\hat{\delta}(p,\varepsilon)\right]$
- ε-Closure (q) = {p,q} $\left[\hat{\delta}(q,\varepsilon)\right]$
- ε-Closure (r) = {p,q,r} $\left[\hat{\delta}(r,\varepsilon)\right]$

Step 2:

Start with ε-closure (p)= {p}

Where, p is the starting state of given ε –NFA.

Step 3:

Find the transition for{p}

$$\delta_D(\{p\},a) = \varepsilon - closure(\delta_N(p,a))$$
$$= \varepsilon - closure(p)$$
$$= \{p\} \qquad \textit{New State}$$
$$\delta_D(\{p\},b) = \varepsilon - closure(\delta_N(p,b))$$
$$= \varepsilon - closure(q)$$
$$= \{p,q\} \qquad \textit{New State}$$
$$\delta_D(\{p\},c) = \varepsilon - closure(\delta_N(p,c))$$
$$= \varepsilon - closure(r)$$
$$= \{p,q,r\}$$

Step 4:

Find the transition for {p,q}

$$\delta_D(\{p,q\},a) = \varepsilon - closure(\delta_N(p,a) \cup \delta_N(q,a))$$
$$= \varepsilon - closure(p,q)$$
$$= \{p,q\}$$
$$\delta_D(\{p,q\},b) = \varepsilon - closure(\delta_N(p,b) \cup \delta_N(q,b))$$
$$= \varepsilon - closure(q,r)$$
$$= \{p,q,r\}$$
$$\delta_D(\{p,q\},c) = \varepsilon - closure(\delta_N(p,c) \cup \delta_N(q,c))$$
$$= \varepsilon - closure(r)$$
$$= \{p,q,r\}$$

Step 5:

Find the transition for {p,q,r}

$$\delta_D(\{p,q,r\},a) = \varepsilon - closure(\delta_N(p,a) \cup \delta_N(q,a) \cup \delta_N(r,a))$$
$$= \varepsilon - closure(p,q,r)$$
$$= \{p,q,r\}$$
$$\delta_D(\{p,q,r\},b) = \varepsilon - closure(\delta_N(p,b) \cup \delta_N(q,b) \cup \delta_N(r,b))$$
$$= \varepsilon - closure(q,r)$$
$$= \{p,q,r\}$$
$$\delta_D(\{p,q,r\},c) = \varepsilon - closure(\delta_N(p,c) \cup \delta_N(q,c) \cup \delta_N(r,c))$$
$$= \varepsilon - closure(p,r)$$
$$= \{p,q,r\}$$

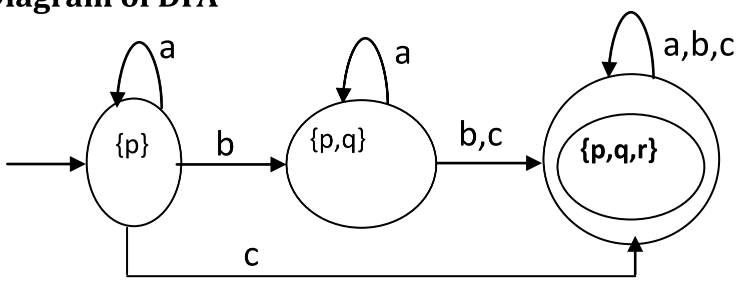
Step 6:

No more new states. Stop the process.

Transition Table of DFA

δ_D	a	b	c
$\rightarrow$ {p}	{p}	{p,q}	{p,q,r}
{p,q}	{p,q}	{p,q,r}	{p,q,r}
* {p,q,r}	{p,q,r}	{p,q,r}	{p,q,r}

Transition Diagram of DFA



2. Consider the following ε-NFA. Covert it into DFA

δ	ε	a	b	c
→p	{q,r}	Φ	{q}	{r}
q	Φ	{p}	{r}	{p,q}
*r	Φ	Φ	Φ	Φ

Step 1:

Compute ε-Closure [states that can be reached by traveling along zero or more ε arrows] for all states .

- ε-Closure (p) = {p,q,r} $\left[\hat{\delta}(p,\varepsilon)\right]$
- ε-Closure (q) = {q} $\left[\hat{\delta}(q,\varepsilon)\right]$
- ε-Closure (r) = {r} $\left[\hat{\delta}(r,\varepsilon)\right]$

Step 2:

Start with ε-closure (p)= {p,q,r}

Where, p is the starting state of given ε - NFA

Step 3:

Find the transition for {p,q,r}

$$\delta_D(\{p,q,r\},a) = \varepsilon - closure(\delta_N(p,a) \cup \delta_N(q,a) \cup \delta_N(r,a))$$
$$= \varepsilon - closure(p)$$
$$= \{p,q,r\}$$

$$\delta_D(\{p,q,r\},b) = \varepsilon - closure(\delta_N(p,b) \cup \delta_N(q,b) \cup \delta_N(r,b))$$
$$= \varepsilon - closure(q,r)$$
$$= \{q,r\}$$ New State

$$\delta_D(\{p,q,r\},c) = \varepsilon - closure(\delta_N(p,c) \cup \delta_N(q,c) \cup \delta_N(r,c))$$
$$= \varepsilon - closure(p,q,r)$$
$$= \{p,q,r\}$$

Step 4:

Find the transition for {q,r}

$$\delta_D(\{q,r\},a) = \varepsilon - closure(\delta_N(q,a) \cup \delta_N(r,a))$$
$$= \varepsilon - closure(p)$$
$$= \{p,q,r\}$$

$$\delta_D(\{q,r\},b) = \varepsilon - closure(\delta_N(q,b) \cup \delta_N(r,b))$$
$$= \varepsilon - closure(r)$$
$$= \{r\}$$ New State

$$\delta_D(\{q,r\},c) = \varepsilon - closure(\delta_N(q,c) \cup \delta_N(r,c))$$
$$= \varepsilon - closure(p,q)$$
$$= \{p,q,r\}$$

Step 5:

Find the transition for {r}

$$\delta_D(\{r\},a) = \varepsilon - closure(\delta_N(r,a))$$
$$= \varepsilon - closure(\varphi)$$
$$= \varphi$$ Dead State

$$\delta_D(\{r\},b) = \varepsilon - closure(\delta_N(r,b))$$
$$= \varepsilon - closure(\varphi)$$
$$= \varphi$$

$$\delta_D(\{r\},c) = \varepsilon - closure(\delta_N(r,c))$$
$$= \varepsilon - closure(\varphi)$$
$$= \varphi$$

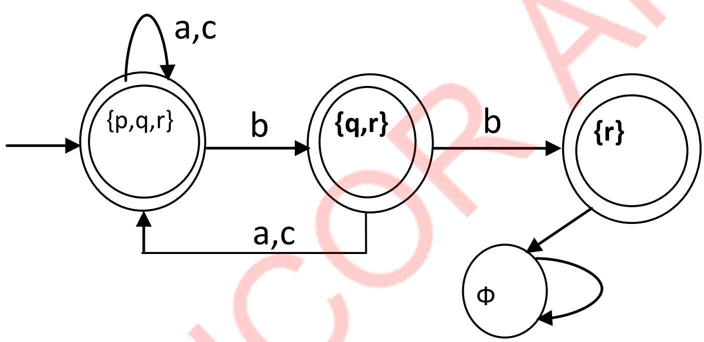
Step 6:

No more new states. Stop the process.

Transition Table of DFA

δ_D	a	b	c
→*{p,q,r}	{p,q,r}	{q,r}	{p,q,r}
*{q,r}	{p,q,r}	{r}	{p,q,r}
*{r}	Φ	Φ	Φ

Transition Diagram of DFA



3. **Consider the following ε-NFA. Covert a,b,c it into DFA.**

Transition Table of ε-NFA

δ_N	ε	0	1	2
→ q_0	q_1	q_0	Φ	Φ
q_1	q_2	Φ	q_1	Φ
*q_2	Φ	Φ	Φ	q_2

Step 1:

Compute ε-Closure [states that can be reached by traveling along zero or more ε arrows] for all states .

- ε-Closure (q_0) = { q_0, q_1, q_2} $\left[\hat{\delta}(q_0, \varepsilon)\right]$

- ε-Closure (q_1) = { q_1 ,q_2} $\left[\hat{\delta}(q_1,\varepsilon)\right]$
- ε-Closure (q_2) = { q_2} $\left[\hat{\delta}(q_2,\varepsilon)\right]$

Step 2:

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

Where, q_0 is the starting state of given ε –NFA.

Step 3:

Find the transition for { q_0, q_1, q_2}

$$\begin{aligned}\delta_D(\{q_0,q_1,q_2\},0) &= \varepsilon-closure(\delta_N(q_0,0)\cup\delta_N(q_1,0)\cup\delta_N(q_2,0))\\ &= \varepsilon-closure(q_0)\\ &= \{q_0,q_1,q_2\}\end{aligned}$$

$$\begin{aligned}\delta_D(\{q_0,q_1,q_2\},1) &= \varepsilon-closure(\delta_N(q_0,1)\cup\delta_N(q_1,1)\cup\delta_N(q_2,1))\\ &= \varepsilon-closure(q_1)\\ &= \{q_1,q_2\} \qquad \text{New State}\end{aligned}$$

$$\begin{aligned}\delta_D(\{q_0,q_1,q_2\},2) &= \varepsilon-closure(\delta_N(q_0,2)\cup\delta_N(q_1,2)\cup\delta_N(q_2,2))\\ &= \varepsilon-closure(q_2)\\ &= \{q_2\} \qquad \text{New State}\end{aligned}$$

Step 4:

Find the transition for {q_1,q_2}

$$\begin{aligned}\delta_D(\{q_1,q_2\},0) &= \varepsilon-closure(\delta_N(q_1,0)\cup\delta_N(q_2,0))\\ &= \varepsilon-closure(\varphi)\\ &= \varphi \qquad \text{Dead State}\end{aligned}$$

$$\begin{aligned}\delta_D(\{q_1,q_2\},1) &= \varepsilon-closure(\delta_N(q_1,1)\cup\delta_N(q_2,1))\\ &= \varepsilon-closure(q_1)\\ &= \{q_1,q_2\}\end{aligned}$$

$$\begin{aligned}\delta_D(\{q,r\},2) &= \varepsilon-closure(\delta_N(q_1,2)\cup\delta_N(q_2,2))\\ &= \varepsilon-closure(q_2)\\ &= \{q_2\}\end{aligned}$$

Step 5:

Find the transition for $\{q_2\}$

$$\begin{aligned}
\delta_D(\{q_2\},0) &= \varepsilon-closure(\delta_N(q_2,0)) \\
&= \varepsilon-closure(\varphi) \\
&= \varphi \qquad \text{Dead State} \\
\delta_D(\{q_2\},1) &= \varepsilon-closure(\delta_N(q_2,1)) \\
&= \varepsilon-closure(\varphi) \\
&= \varphi \qquad \text{Dead State} \\
\delta_D(\{q_2\},2) &= \varepsilon-closure(\delta_N(q_2,2)) \\
&= \varepsilon-closure(q_2) \\
&= \{q_2\}
\end{aligned}$$

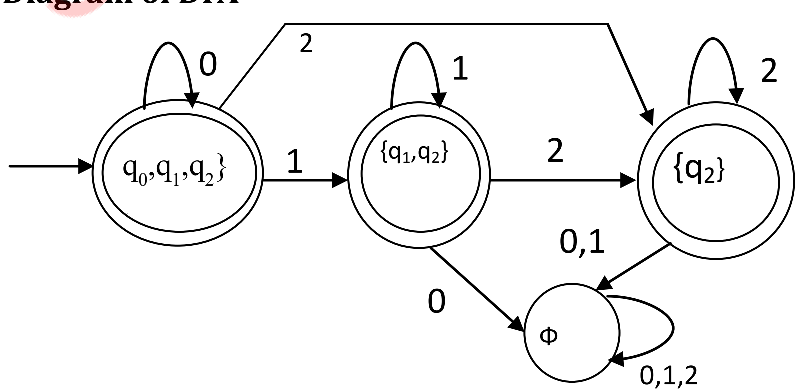
Step 6:

No more new states. Stop the process.

Transition Table of DFA

δ_D	0	1	2
$\rightarrow * \{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
$* \{q_1, q_2\}$	Φ	$\{q_1, q_2\}$	$\{q_2\}$
$* \{q_2\}$	Φ	Φ	$\{q_2\}$

Transition Diagram of DFA



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

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

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

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

4. Consider the following ε-NFA. Covert it into DFA

δ_N	ε	a	b
→ p	{r}	{q}	{p,r}
q	Φ	{p}	Φ
*r	{p,q}	{r}	{p}

Step 1:

Compute ε-Closure [states that can be reached by traveling along zero or more ε arrows] for all states.

- ε-Closure (p) = {p,q,r} $\left[\hat{\delta}(p,\varepsilon)\right]$
- ε-Closure (q) = {q} $\left[\hat{\delta}(q,\varepsilon)\right]$
- ε-Closure (r) = {p,q,r} $\left[\hat{\delta}(r,\varepsilon)\right]$

Step 2:

Start with ε-closure (p)= { p, q, r}

Where, p is the starting state of given ε –NFA

Step 3:

Find the transition for { p,q,r}

$$\begin{aligned}\delta_D(\{p,q,r\},a) &= \varepsilon-closure(\delta_N(p,a)\cup\delta_N(q,a)\cup\delta_N(r,a))\\ &= \varepsilon-closure(q\cup p\cup r)\\ &= \varepsilon-closure(p,q,r)\\ &= \{p,q,r\}\end{aligned}$$

$$\begin{aligned}\delta_D(\{p,q,r\},b) &= \varepsilon - closure(\delta_N(p,b)\cup\delta_N(q,b)\cup\delta_N(r,b))\\ &= \varepsilon - closure(\{\{p,r\}\cup\varphi\cup\{p\}\})\\ &= \varepsilon - closure(\{p,r\})\\ &= \{p,q,r\}\end{aligned}$$

Transition Table of DFA

δ_D	a	b
→*{ p,q,r}	{ p,q,r}	{ p,q,r}

REVIEW QUESTIONS

1. Convert the following NFAs to a DFA .

a.

	a	b
→p	{p,q}	p
q	r	r
r	s	-
*s	s	s

b.

δ	a	b
→ p	{q,s}	{q}
* q	{r }	{q,r }
r	{s}	{p}
* s	ϕ	{p}

c.

δ	a	b
→ p	{p,q}	{p}
q	{r,s }	{t}
r	{p,r}	{t}
* s	ϕ	ϕ
* t	ϕ	ϕ

2. Consider the following ε- NFA. Compute the ε- Closure of each state and find it's equivalent DFA .

a.

δ	ε	a	b	c
→p	{q,r}	-	{q}	{r}
q	-	{p}	{r}	{p,q}
*r	-	-	-	{r}

b.

δ	ε	a	b	c
→p	ϕ	{p}	{q}	{r}
q	{p}	{q}	{r}	ϕ
*r	{q}	{r}	ϕ	{p}

3. Construct a minimized DFA for the DFA given below.

a.

δ	0	1
→ A	B	E
B	C	F
*C	D	H
D	E	H
E	F	I
*F	G	B
G	H	B
H	I	C
*I	A	E

b.

δ	0	1
→ A	B	A
B	A	C
C	D	B
*D	D	A
E	D	F
F	G	E
G	F	G
H	G	D

4. Construct (DFA) an Automata for the following Language

a. D = { *w* | *w* has equal number of occurrences of 01 and 10}

b. D ={{ *w* | *w* begins with 10 and ends with 10}

c. D ={{ *w* | *w* begins with 01 and ends with 01}

d. D ={{ w | w begins with 10 and ends with 00}

D ={{ w | w begins with 10 and ends with 01}

5. Consider the following ε-NFA.

δ	ε	0	1
→ p	{r}	{q}	{p,r}
q	Φ	{p }	Φ
* r	{p,q}	{r}	{p}

a. Compute the ε-closure of each state.

b. List all the possible strings of length 3 or less accepted by the automaton.

c. Convert the automaton to a DFA.

d. Compute $\hat{\delta}(q_0, 0110)$, where q_0 is the start state.

6. Obtain the DFA equivalent to the following ε-NFA.

	ε	a	b	c
→p	-	{p}	{q}	{r}
q	{p}	{q}	{r}	-
*r	{q}	{r}	-	{p}

7. Let L be a language accepted by a NFA then show that there exists a DFA that accepts L.

8. Design a NFA that accepts set of all strings that begins with bb and ends with aa. Convert it into DFA.

9. Construct a minimized DFA for the DFA given below.

δ	0	1
→ a	b	c
b	c	d
c	c	d
*d	d	d
*e	e	e
*f	f	e

10. Design a NFA that accepts empty string or string starts and ends with 0. Convert it into DFA.

11. Define NFA. Explain its significance. Convert the given NFA to DFA. Prove that both NFA and DFA accepts the string 0110.

UNIT-2

REGULAR EXPRESSIONS AND LANGUAGES

* REGULAR EXPRESSION:-

The language accepted by finite automata are easily described with simple expression is called Regular Expression. The method of representing language each regular language are denoted as $r \Rightarrow L(r)$

* REGULAR LANGUAGE:-

A language accepted by finite automata is called regular language.

* RULES FOR DETERMINING A REGULAR EXPRESSION OVER AN INPUT ALPHABET:-

(1) ϕ is a regular expression that denotes the empty set { }

(2) ϵ is a regular expression that denotes the $\{\epsilon\}$

(3) For each 'a' in Σ, a is a regular expression that denotes the set contain $\{a\}$

(4) If r and s are regular expression that denotes the language $L(r)$ and $L(s)$

(i) $(r)/(s)$ (or) $(r)+(s) = L(r) \cup L(s)$

(ii) $(r).(s) = L(r)\,L(s)$

(iii) $(r)^* = (L(r))^*$

REPRESENTATION OF REGULAR EXPRESSION

Write a regular expression for the language accepting

① All combinations of a's

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

$R.E = a^*$

② All combinations of a's except null string

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

$R.E = a^+$

③ containing any number of 0's and 1's

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

$R.E = (0+1)^*$ (or) $(0/1)^*$

④ containing 0's and 1's except null string.

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

$R.E = (0+1)^+$ (or) $(0/1)^+$

⑤ Starting with 101

$L = \{101, 10100, 1010101, \ldots\}$

$R.E = 101(0+1)^*$

⑥ Ending with 110

$L = \{00110, 010110, 110110 \ldots\}$

$R.E = (0+1)^* 110$

⑦ String containing substring as 1001

$L = \{010010, 10100101, \ldots\}$

$R.E = (0+1)^* 1001 (0+1)^*$

⑧ String starting with 1 and ending with 0.

$L = \{10, 1000, 1100 \ldots\}$

$R.E = 1(0+1)^* 0$

⑨ Any number of a's followed by any number of b's followed by atleast 1c.

$L = \{abc, aabbc, aaabbbcc\}$

$R.E = a^* b^* c^+$

⑩ Exactly string length is 2

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

$R.E = (0+1)(0+1)$ (or) $(0+1)^2$

⑪ Atleast string length is 2

$L = \{00, 010, 0110, \ldots\}$

$R.E = (0+1)(0+1)(0+1)^*$

⑫ String length is atmost 2.

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

$R.E = (\varepsilon+0+1)(\varepsilon+0+1)$

(13) String length with even

$L = \{01, 0110, 111000 \ldots\}$

$R.E = ((0+1)(0+1))^*$

(14) String length with odd

$L = \{1, 010, 11001, \ldots\}$

$R.E = (0+1)((0+1)(0+1))^*$

(or)

$R.E = ((0+1)(0+1))^*(0+1)$

(15) Number of 'a' is exactly 2

$L = \{baab, abab, bbaa \ldots\}$

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

(16) Number of 'a' is atleast 2 over the input alphabet Σ.

$L = \{baaab, babba \ldots\}$

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

(17) Atmost two 'a'

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

$R.E = b^*(a+\epsilon)b^*(a+\epsilon)b^*$

(18) Number of 'a' is even

$L = \{aab, babaabaa \ldots\}$

$R.E = (b^* a b^* a b^*)^*$

(19) Number of 'a' is odd

$L = \{ab, bababa, \ldots\}$

$R.E = (b^* a b^* a b^*)^* a$.

(20) Starting & ending with different symbol.

$L = \{ab, abb, aab, aabb \ldots\}$

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

(21) Starting & ending with same symbol.

$L = \{aba, baab, bbb \ldots\}$

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

(22) Third character from right end of string is always b.

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

$L = \{abaa, babab, ababaa \ldots\}$

(23) From the left end 4th one should be a.

$L = \{abbab, aabaa, babab \ldots\}$

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

(24) String length divisible by 3

(or) $|w| \% 3$

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

(25) $|w| \equiv 2 \mod 3$

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

(26) Write Regular Expression to denote the language which accept all the strings which begins (or) ends with either 00 (or) 11.

soln:-

$L_1 = (00+11)(0+1)^*$ $\quad$ $L_2 = (0+1)^*(00+11)$

$$R.E = L_1 + L_2$$

$$R_1 = (00+11)(0+1)^* + (0+1)^*(00+11)$$

* REGULAR EXPRESSION INTO FINITE AUTOMATA CONVERSION USING THOMSON'S RULE:-

* THEOREM:-

Every language defined by the regular expression is also defined by finite automata (or)

Let 'r' be a regular expression then there exists a NFA with ε-transition that except L(r)

* PROOF:-

Suppose L = L(r) for a regular expression 'r' we show that L = L(E) for some ε-NFA with

(i) Exactly only one accepting state.

(ii) No Edges into the initial state.

(iii) No Edges out of the accepting state.

* BASIS RULE:-

(i) ϕ , $\rightarrow$ ○ ◎

That means no path from initial state to final state.

(ii) ε, 

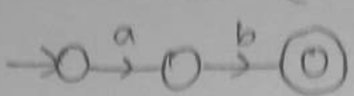
The language that has {ε} since only path from starting state to accepting state with label ε.

r = a

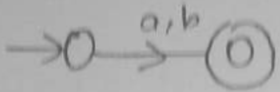
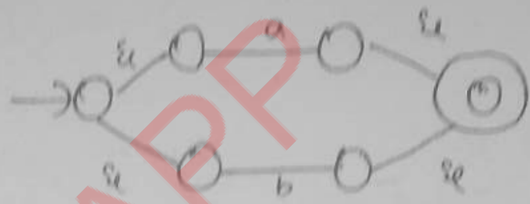
 - NFA ;  - ε-NFA

INDUCTION RULE

i) r = ab (CONCATENATION) :-

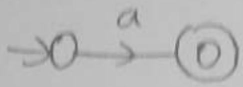
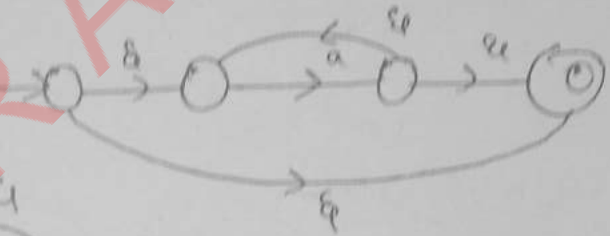
 - NFA ;  - ε-NFA

(ii) r = a+b (or) r = a/b (UNION) :-

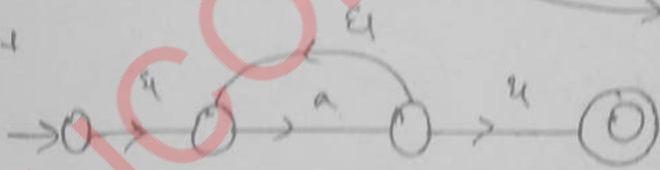
 - NFA ;  - ε-NFA

iii) (CLOSURE) r = a*

r = a*

 - NFA ;  - ε-NFA

r = a⁺

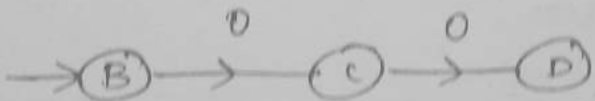
 - ε-NFA

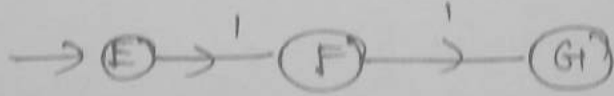
PROBLEMS

Convert Regular Expression into Finite automata

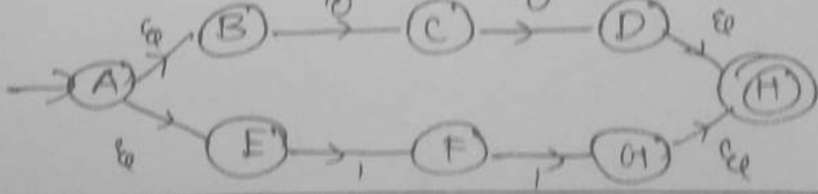
1) (00+11)

soln:-

$r_1 = 00$ 

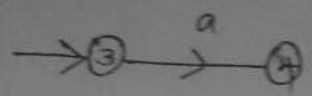
$r_2 = 11$ 

$r_3 = 00+11$

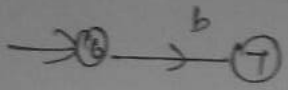


(2) $(a|b)^* abb$

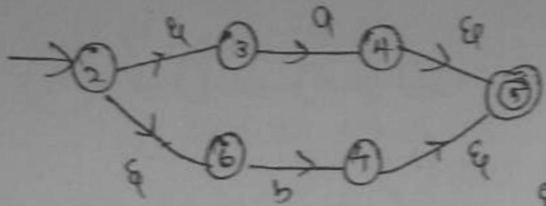
$r_1 = a$



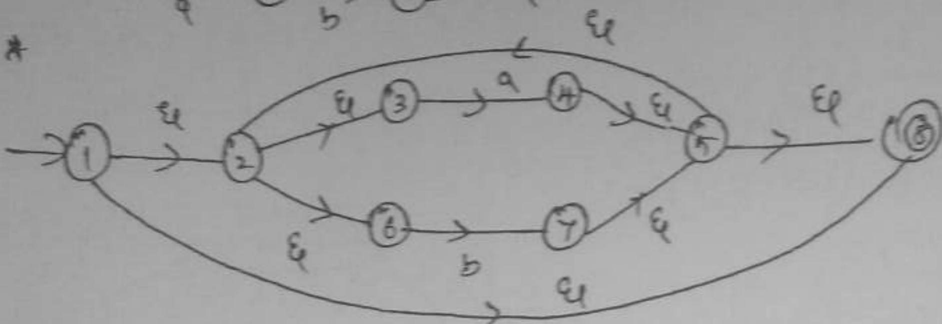
$r_2 = b$



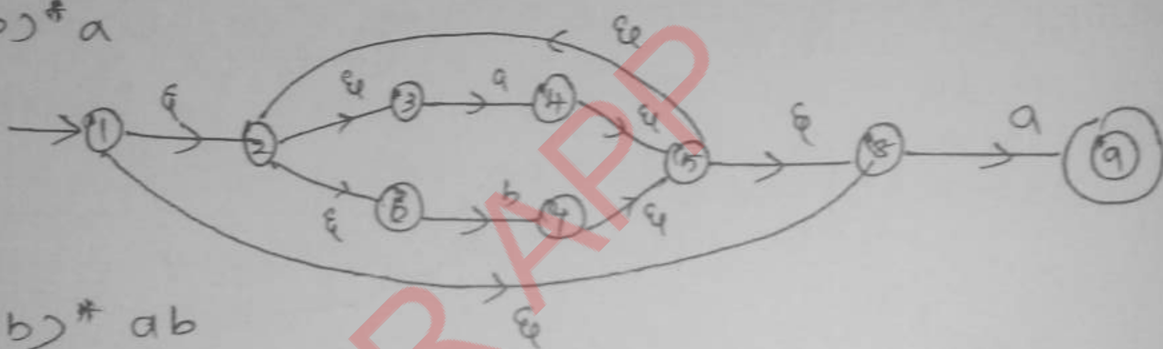
$r_3 = a|b$



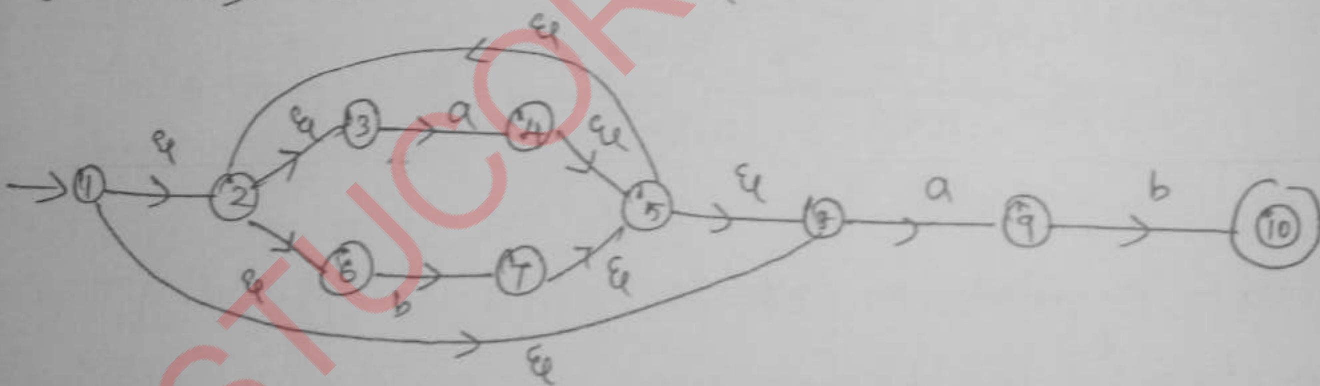
$r_4 = (a|b)^*$



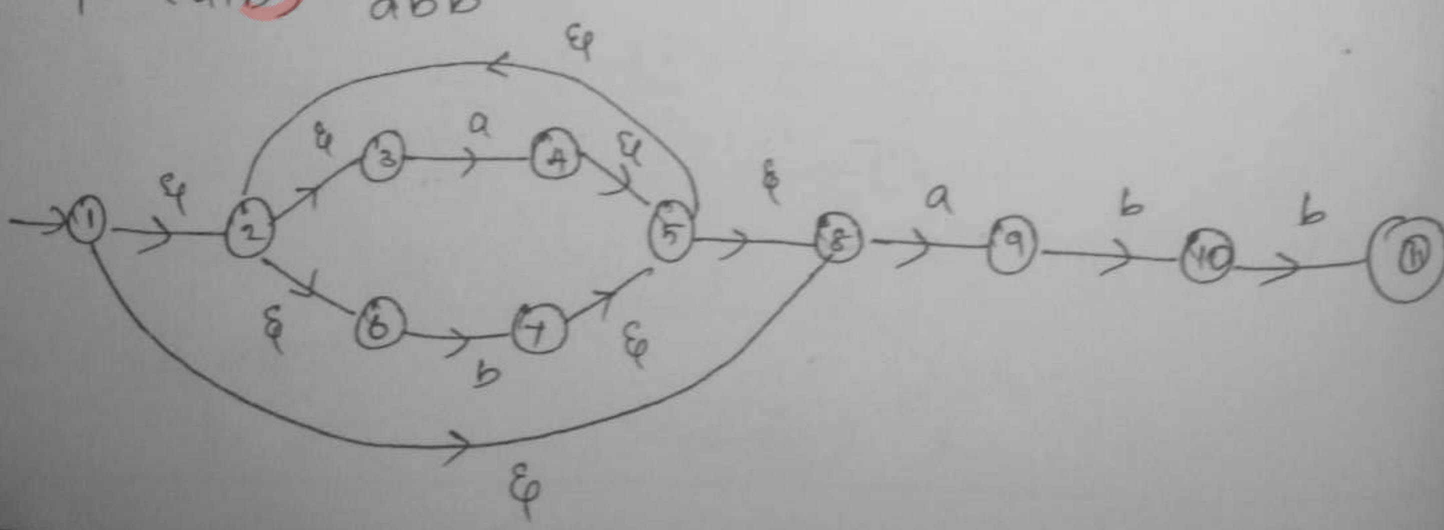
$r_5 = (a|b)^* a$



$r_6 = (a|b)^* ab$

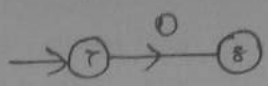


$r_7 = (a|b)^* abb$

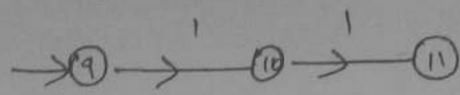


③ $10 + (0+11)^{*}01$

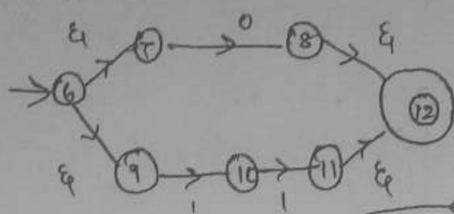
$r_1 = 0$



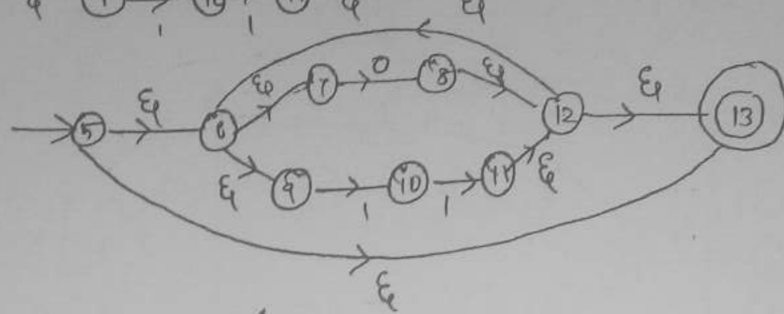
$r_2 = 11$



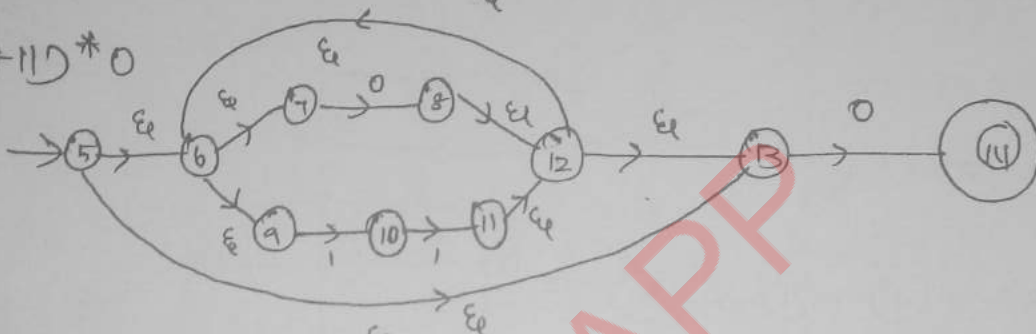
$r_3 = 0+11$



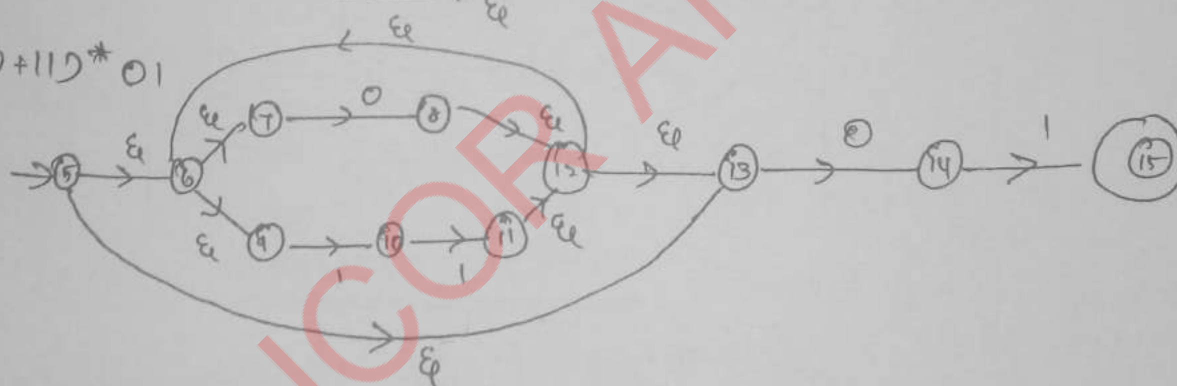
$r_4 = (0+11)^{*}$



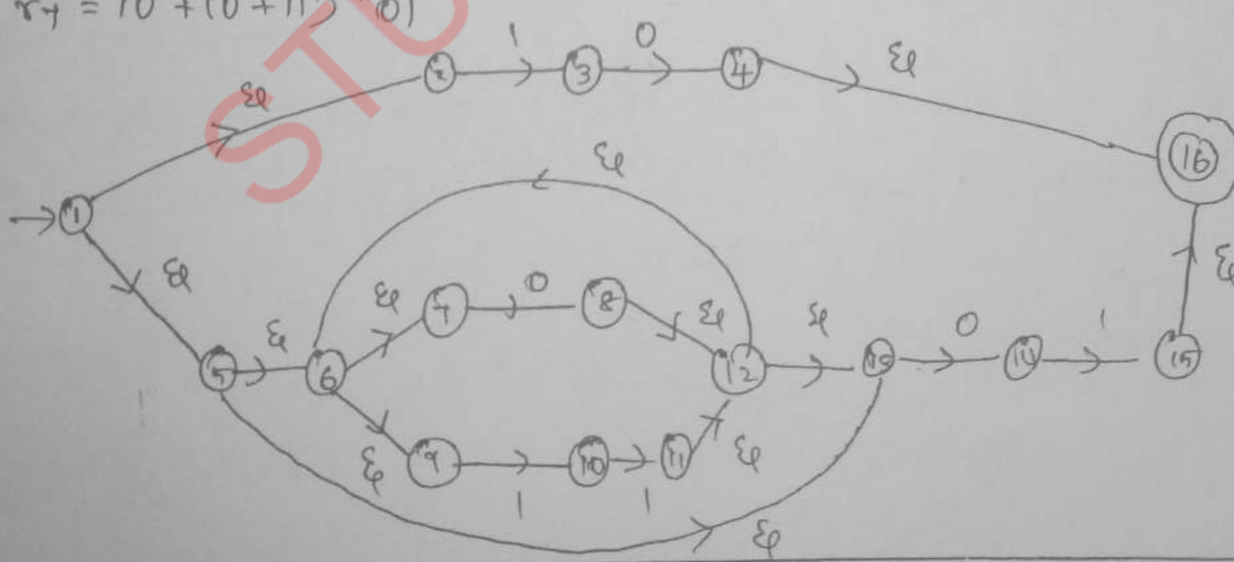
$r_5 = (0+11)^{*}0$



$r_6 = (0+11)^{*}01$

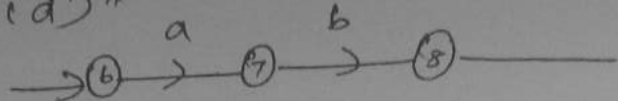


$r_7 = 10 + (0+11)^{*}01$

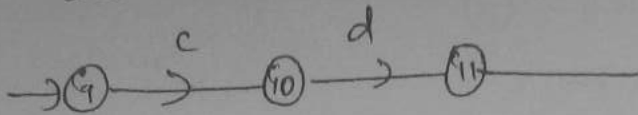


(4) abc(ab+cd)*

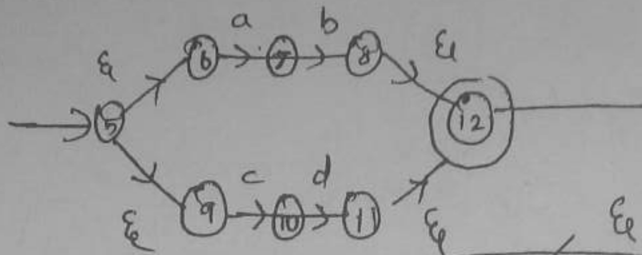
$r_1 = ab$



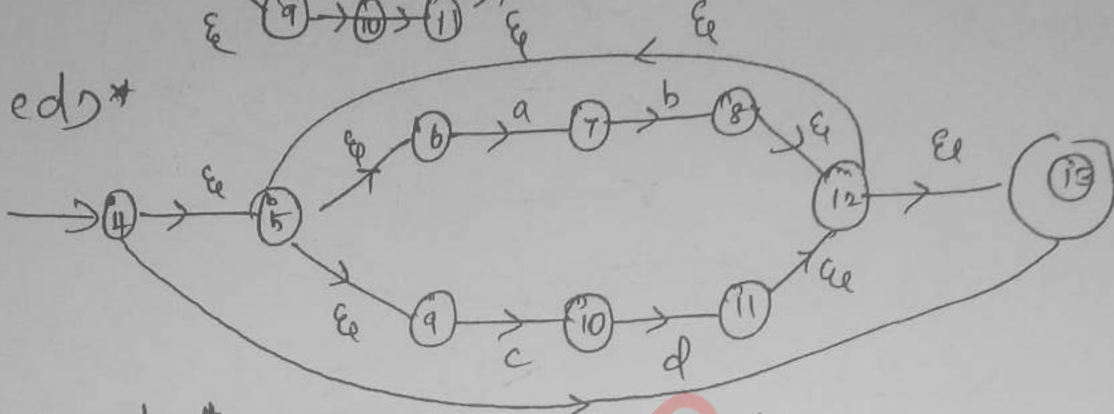
$r_2 = cd$



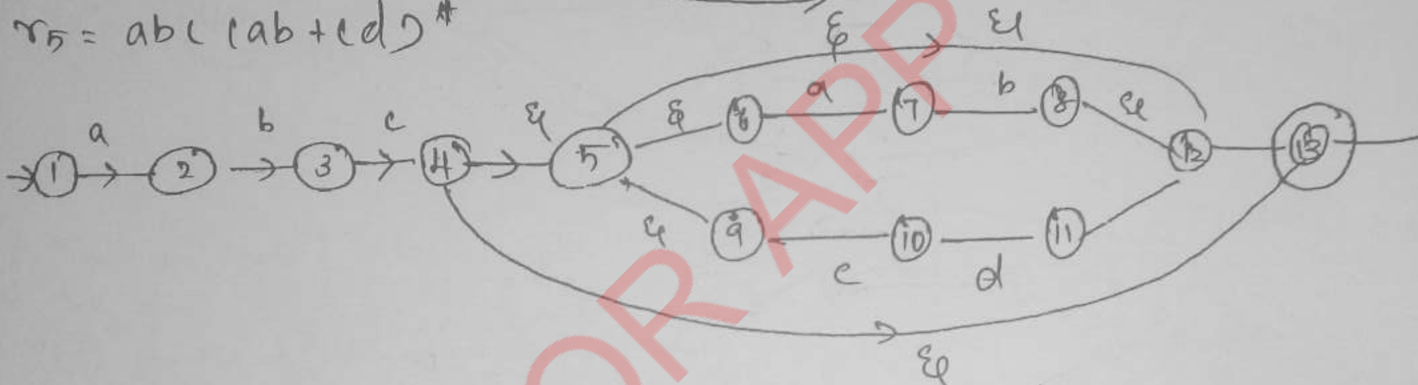
$r_3 = (ab+cd)$



$r_4 = (ab+cd)^*$

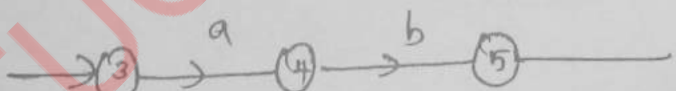


$r_5 = abc(ab+cd)^*$

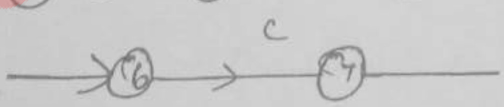


(5) $(ab+c)^*b$

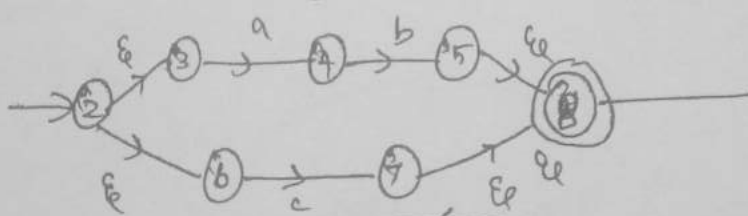
$r_1 = ab$



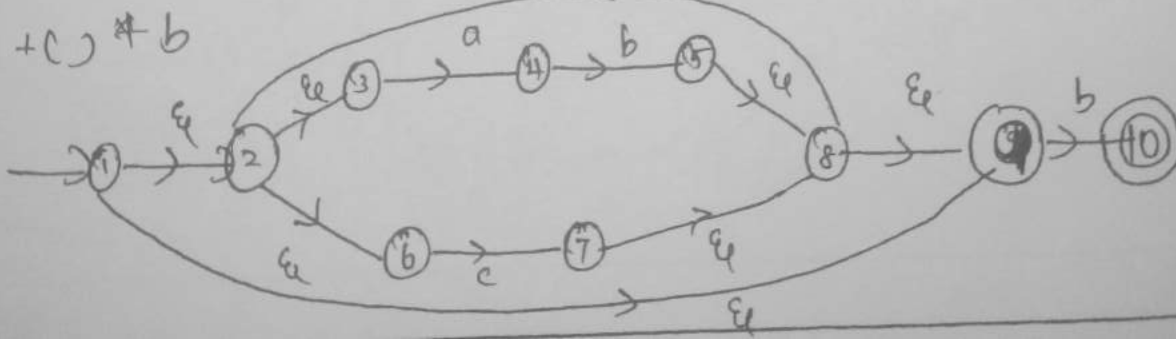
$r_2 = c$



$r_3 = (ab+c)$



$r_4 = (ab+c)^*b$



CONVERSION FROM FINITE AUTOMATA INTO REGULAR EXPRESSION :-

There are 3 methods.

1) Formula method (or) R_{ij} method.
2) Arden's Theorem.
3) State elimination Technique.

IDENTITY RULES

1) $(\varepsilon+R)^* = R^*$
2) $(\varepsilon+R)+R^* = R^*$
3) $R^*(\varepsilon+R) = R^*$
4) $R+RS^* = RS^*$
5) $\varepsilon + RR^* = R^*$
6) $\varepsilon+R = R$
7) $R.\varepsilon = R$
8) $R+R = R$
9) $RR^* = R^+$
10) $R+\phi = R$
11) $R.\phi = \phi$
12) $\phi^* = \varepsilon$
13) $\varepsilon^* = \varepsilon$
14) $(R^*)^* = R^*$
15) $RR^* = R^*R \neq R^*$
16) $(R+S)^* = R^*S^*$
17) $(R+S)^* = (R^*+S^*)^*$
18) $(P+Q)R = PR+QR$
19) $R(P+Q) = RP+RQ$
20) $(RS)^* \neq R^*S^*$

$R_{ij}^{(k)}$ METHOD / FORMULA METHOD

$R_{ij}^{(k)}$ stands for set of all strings that takes the DFA from state q_i to state q_j without going through any number higher than k. We can write the reverse of

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

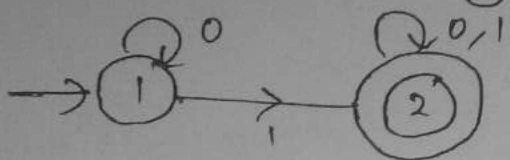
where k = 0 to n

Number of iteration → k+1 [0 to k]

Language accepted by FA, $L(A) = R_{ij}^{(k)}$ $k \rightarrow$ no. of states
i — initial state, j — final state

PROBLEMS

Convert the following Finite Automata into Regular Expression.

(1) 

Soln:-

S1: $L(A) = R_{12}^{(2)}$

S2: Using Formula,

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

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

case (i) : $k=0$

In $k=0$, if $i=j$ then add ε

		# Transition
$R_{11} = 0+\varepsilon$	$R_{21} = \phi$	$1 \to 1$
		$1 \to 2$
$R_{12} = 1$	$R_{22} = 0+1+\varepsilon$	$2 \to 1$
		$2 \to 2$

case (ii) : $k=1$

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

$= 1 + (0+\varepsilon)(0+\varepsilon)^*(1)$

$= 1 + (0+\varepsilon)0^*(1)$

$= 1 + 0^*(1)$

$= 0^*1$

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

$= (0+1+\varepsilon) + \phi\,(0+\varepsilon)^*(1)$

$= (0+1+\varepsilon) + \phi$

$= 0+1+\varepsilon$

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

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

$= 0+\varepsilon + 0^*$

$= 0^*$

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

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

$= \phi.$

case iii) : $k = 2$

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

$= 0^*1 + 0^*1 (0+1+\varepsilon)^* (0+1+\varepsilon)$

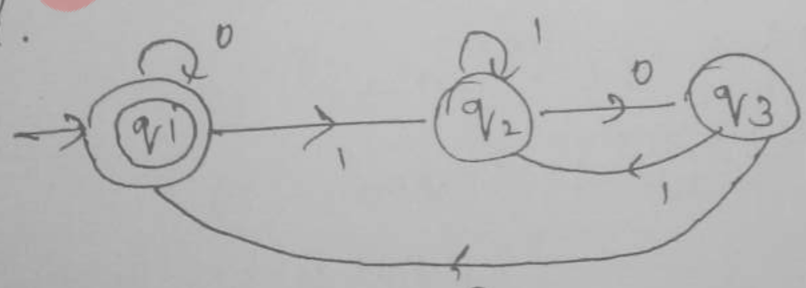
$= 0^*1 [\varepsilon + (0+1+\varepsilon)^* (0+1+\varepsilon)]$

$= 0^*1 [\varepsilon + (0+1+\varepsilon)^*]$

$= 0^*1 [\varepsilon + (0+1)^*]$

$= 0^*1 (0+1)^*$ //.

(2) Obtain the regular expression denoting the language accepted by the following DFA using R_{ij} method.



Soln:-

S1: $L(A) = R_{11}^{(3)}$

S2: Using the formula,

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

$$\Rightarrow \quad R_{11}^{(3)} = R_{11}^{(2)} + R_{13}^{(2)} \left(R_{33}^{(2)}\right)^* R_{31}^{(2)}$$

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

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

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

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

S3: case (i): $k=0$

$R_{11}^{(0)} = 0+\varepsilon$	$R_{21}^{(0)} = \phi$	$R_{31}^{(0)} = 0$
$R_{12}^{(0)} = 1$	$R_{22}^{(0)} = 1+\varepsilon$	$R_{32}^{(0)} = 1$
$R_{13}^{(0)} = \phi$	$R_{23}^{(0)} = 0$	$R_{33}^{(0)} = \varepsilon$

case (ii): $k=1$

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

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

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

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

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

$$= 0^*$$

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

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

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

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

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

$$R_{31}^{(1)} = R_{31}^{(0)} + R_{31}^{(0)} (R_{11}^{(0)})^* R_{11}^{(0)}$$
$$= 0+0 (0+\varepsilon)^* (0+\varepsilon)$$
$$= 0+0 (0+\varepsilon)^*$$
$$= 0 + 00^* = 00^*$$

case (ii) $k=2$

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

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

$$= 0^* + \phi$$

$$= 0^*$$

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

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

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

$$= 0^* 11^* 0$$

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

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

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

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

$$= 00^* + 1(1+\varepsilon)^* \phi$$

$$= 00^* + \phi$$

$$= 00^*$$

case (iii): $k=3$

$$R_{11}^{(3)} = R_{11}^{(2)} + R_{13}^{(2)} (R_{33}^{(2)})^* R_{31}^{(2)}$$

$$= 0^* + 0^* 11^* 0 (\varepsilon + (1+00^*1) 1^* 0)^* 00^*$$

ARDEN'S THEOREM:-

If P and Q are two regular expression over Σ and if P does not obtain ϵ from the following equation in R given by, $R = Q + RP$ as a Unique solution. (i.e) $R = QP^*$

PROOF:- (METHOD 1)

$$R = Q + RP$$

Replace R by QP^*

$$R = Q + QP^*P$$
$$= Q[\epsilon + P^*P]$$
$$\boxed{R = QP^*}$$

(METHOD 2) : We are checking $R = QP^*$ as a only unique solution for $R = RP + Q$.

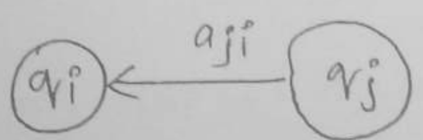
$$R = Q + RP$$

Replace R by $Q + RP$

$$R = Q + (Q + RP)P$$
$$= Q + QP + RP^2$$
$$= Q + QP + (Q + RP)P^2$$
$$= Q + QP + QP^2 + RP^3$$
$$\vdots$$
$$= Q + QP + QP^2 + \dots\dots + QP^n + RP^{(n+1)} \quad (R = QP^*)$$
$$= Q + QP + QP^2 + \dots\dots QP^n + QP^*P^{n+1}$$
$$= Q[\epsilon + P + P^2 + \dots + P^n + P^*P^{n+1}]$$
$$= QP^*$$
$$\boxed{R = QP^*}$$

RULES

(i) Let a_{ji} represent the transition from state q_j to q_i



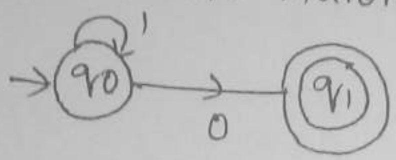
(iii) Then calculate q_i such that $q_i = q_j \cdot a_{ji}$

If q_i is the start state then the equation becomes $q_i = q_j \cdot a_{ji} + \epsilon$.

Similarly, computing final state which ultimately use regular expression R. If two states are final states, then add answer of two states

PROBLEMS

Convert Finite Automata to Regular Expression.

(1) 

S1 : $R = Q + RP$

$\Rightarrow R = QP^*$

S2 : Eliminate unreachable state.

$\Rightarrow$ Here no unreachable state so no need to eliminate.

S3 : Write the equation of each state,

$$q_0 = q_0 \cdot 1 + \epsilon \longrightarrow (1)$$

$$q_1 = q_0 \cdot 0 \longrightarrow (2)$$

From (1),

$$\underset{R}{q_0} = \underset{R}{q_0} \cdot \underset{P}{1} + \underset{Q}{\epsilon}$$

$$q_0 = \epsilon \cdot 1^*$$

$$\boxed{q_0 = 1^*} \longrightarrow (3)$$

Sub eq (3) in eq (2)

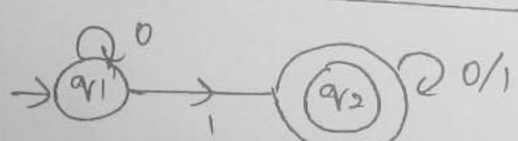
$$\boxed{q_1 = 1^* 0}$$

R.E $= 1^* 0$.

Ans :

R.E $= q_0 + q_1$

$$\boxed{R.E = 1^* + 1^* 0}$$

(2) 

S1 : $R = Q + RP$

$\Rightarrow R = QP^*$

S2: Eliminate unreachable state

⇒ Here no unreachable state. so no need to eliminate.

S3: write the transition equation of each state.

$$q_1 = q_1 \cdot 0 + \varepsilon \longrightarrow (1)$$

$$q_2 = q_1 \cdot 1 + q_2 (0+1) \longrightarrow (2)$$

From (1),

$$\underbrace{q_1}_{R} = \underbrace{q_1}_{R} \cdot \underbrace{0}_{P} + \underbrace{\varepsilon}_{Q}$$

$$q_1 = \varepsilon \cdot 0^*$$

$$\boxed{q_1 = 0^*} \longrightarrow (3)$$

sub eq (3) in eq (2),

$$\underbrace{q_2}_{R} = \underbrace{0^* 1}_{Q} + \underbrace{q_2}_{R} \underbrace{(0+1)}_{P}$$

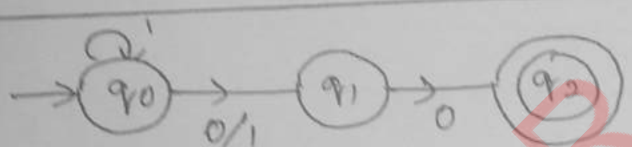
$$\boxed{q_2 = 0^* 1 (0+1)^*}$$

Ans:

$$R.E = q_1 + q_2$$

$$\boxed{R.E = 0^* + 0^* 1 (0+1)^*}$$

---

(3) 

(step 1 and step 2 are same as in sum number (1))

S3: $q_0 = q_0 \cdot 1 + \varepsilon \longrightarrow (1)$

$q_1 = q_0 (0+1) \longrightarrow (2)$

$q_2 = q_1 \cdot 0 \longrightarrow (3)$

From (1), $\underbrace{q_0}_{R} = \underbrace{q_0}_{R} \cdot \underbrace{1}_{P} + \underbrace{\varepsilon}_{Q}$

$$q_0 = \varepsilon \cdot 1^*$$

$$\boxed{q_0 = 1^*} \longrightarrow (4)$$

From (2), sub (4) in eq (2)

$q_1 = q_0 (0+1)$

$= 1^* (0+1)$

$\boxed{q_1 = 1^* (0+1)} \rightarrow ⑤$

sub eq ⑤ in eq ③

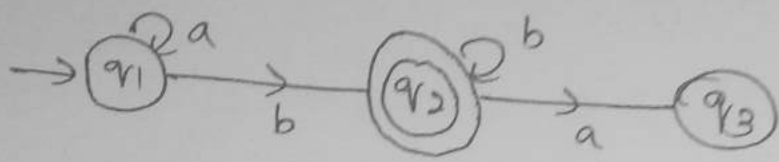
$\boxed{q_2 = 1^* (0+1) . 0}$

Ans:

$R.E = q_1 + q_2$

$\boxed{R.E = 1^* (0+1) + 1^* (0+1).0}$

④



(step 1 and step 2 are same as in sum number ①)

S3 : write the transition equation of each state.

$q_1 = q_1 . a + \epsilon \rightarrow ①$

$q_2 = q_2 . b + q_1 . b \rightarrow ②$

$q_3 = q_2 . a \rightarrow ③$

From ①, $\underset{R}{q_1} = \underset{R}{q_1} . \underset{P}{a} + \underset{Q}{\epsilon}$

$q_1 = \epsilon . a^*$

$\boxed{q_1 = a^*} \rightarrow ④$

sub ④ in eq ②,

$\underset{R}{q_2} = \underset{Q}{a^* b} + \underset{R}{q_2} . \underset{P}{b}$

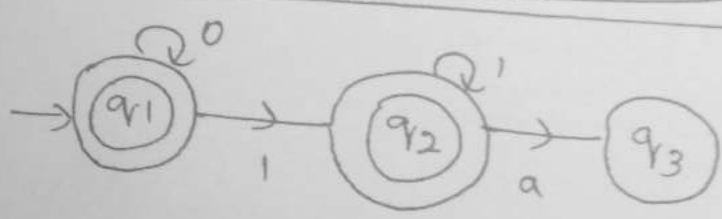
$\boxed{q_2 = a^* b b^*}$

Ans:

$R.E = q_1 + q_2$

$\boxed{R.E = a^* + a^* b b^*}$

⑤



soln:-

(step1 and step2 are same as in sum number ①)

S3: Find the transition equation of each state

$q_1 = q_1 \cdot 0 + \epsilon \rightarrow (1)$

$q_2 = q_1 \cdot 1 + q_2 \cdot 1 \rightarrow (2)$

$q_3 = q_2 \cdot 0 \rightarrow (3)$

From (1),

$$\underset{R}{\underline{q_1}} = \underset{R}{\underline{q_1}} \cdot \underset{P}{\underline{0}} + \underset{Q}{\underline{\epsilon}}$$

$$q_1 = \epsilon \cdot 0^*$$

$$\boxed{q_1 = 0^*} \rightarrow (4)$$

sub (4) in eq (2),

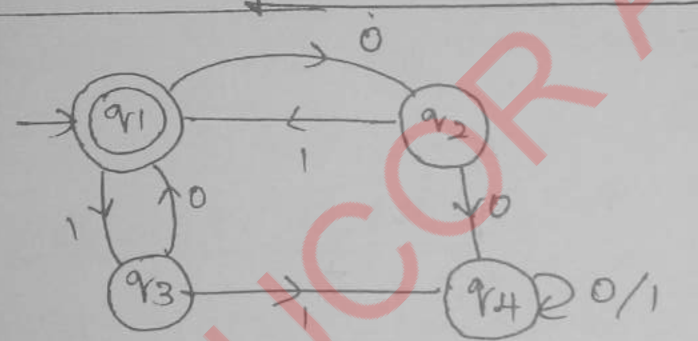
$$\underset{R}{\underline{q_2}} = \underset{Q}{\underline{0^* \cdot 1}} + \underset{R}{\underline{q_2}} \cdot \underset{P}{\underline{1}}$$

$$\boxed{q_2 = 0^* 1 1^*} \rightarrow (5)$$

$$R.E = q_1 + q_2$$

$$\boxed{R.E = 0^* + 0^* 1 1^*}$$

(6)



(step1 and step2 are same as in sum number (1))

S3: Find the transition equation of each state.

$q_1 = q_2 \cdot 1 + q_3 \cdot 0 + \epsilon \rightarrow (1)$

$q_2 = q_1 \cdot 0 \rightarrow (2)$

$q_3 = q_1 1 \rightarrow (3)$

$q_4 = q_2 \cdot 0 + q_3 \cdot 1 + q_4 \cdot (0+1) \rightarrow (4)$

sub (2) & (3) in eqn (1)

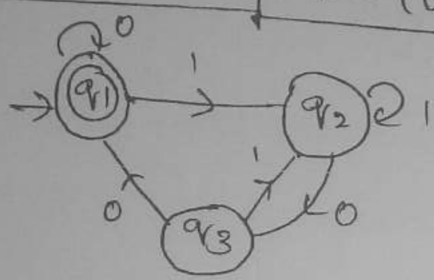
$$q_1 = q_1 \cdot 01 + q_1 \cdot 10 + \varepsilon$$

$$\underbrace{q_1}_{R} = \underbrace{q_1}_{R} \underbrace{[01+10]}_{P} + \underbrace{\varepsilon}_{Q}$$

$$q_1 = (01+10)^*$$

$$\boxed{R.E = (01+10)^*}$$

4)



(step 1 and step 2 are same as sum number (1))

S3: write the transition equation for each state.

$q_1 = q_1 \cdot 0 + q_3 \cdot 0 + \varepsilon \rightarrow (1)$

$q_2 = q_1 \cdot 1 + q_2 \cdot 1 + q_3 \cdot 1 \rightarrow (2)$

$q_3 = q_2 \cdot 0 \rightarrow (3)$

sub eq (3) in eq (2) $\Rightarrow q_2 = q_1 \cdot 1 + q_2 \cdot 1 + q_2 \cdot 01$

$$\underbrace{q_2}_{R} = \underbrace{q_1 \cdot 1}_{Q} + \underbrace{q_2}_{R} \underbrace{[1+01]}_{P}$$

$$\boxed{q_2 = q_1 \cdot 1 \, (0+01)^*} \rightarrow (4)$$

sub (4) in eq (3) $\Rightarrow \boxed{q_3 = q_1 \cdot 1 \, (1+01)^* \, 0} \rightarrow (5)$

sub (5) in eq (1) $\Rightarrow q_1 = q_1 \cdot 0 + q_1 \cdot 1 \, (0+01)^* \, 00 + \varepsilon$

$$\underbrace{q_1}_{R} = \underbrace{q_1}_{R} \underbrace{[0+1 \, (1+01)^* \, 00]}_{P} + \underbrace{\varepsilon}_{Q}$$

$$\boxed{R.E = (0+1 \, (1+01)^* \, 00)^*}$$

EQUIVALENCE AND MINIMIZATION OF FINITE AUTOMATA

* MINIMIZATION OF FINITE AUTOMATA :-

Process of constructing an equivalent DFA with minimum number of states.

It may be possible that for a given DFA, some of the states may be equivalent (or) they are not distinguishable.

Equivalent states from a equivalent class (or) root that is every states from an equivalent class as same transition behaviour.

Then state of a DFA can be divided into group of equivalent class.

* EQUIVALENT STATE :-

Two state q_i & $q_j \in Q$ are said to be equivalent

If q_i is an accepting state and q_j is a non-accepting state, then q_i and q_j are distinguishable.

If there is $\delta^*(q_i, x) \in F$ and $\delta^*(q_j, x) \notin F$ (or)

$\delta^*(q_i, x) \notin F$ and $\delta^*(q_j, x) \in F$

MINIMIZATION METHOD

1) Myhill - Nerode Theorem (or) Table Filling Algorithm.

2) π - method (or) State Equivalence method.

* π-METHOD / STATE EQUIVALENCE METHOD:-

Initially the states are divided into two groups. (i.e) final state & non-final state.

For each group repeat the following steps until no more groups can be splitted.

Transition on the input symbol is checked for every state.

If the transition state falls into two different groups, then group is splitted.

* ALGORITHM:-

S1 : we will divide states Q into two different set. One set contains all final states and other set contain all non-final states. This partition is called π_0.

S2 : Initialize $K=1$

S3 : Find π_K by partitioning the different set of π_{K-1}. In each set of π_{K-1} we will take all possible pair of states. If two states of a set are distinguishable, we will split the set into different set in π_K.

S4 : stop when $\boxed{\pi_K = \pi_{K-1}}$

S5 : All states of 1 set are merged into 1. Number of states in minimized DFA will be equal to number of set in π_K.

* MYHILL - NERODE THEOREM / TABLE FILLING METHOD:-

* PROCEDURE:-

S1: construct a table for all pair of states (P, Q). Initially all are unmarked.

S2: consider every state pair (P, Q) in the DFA where $P \in$ final & $Q \notin$ final and vice versa. 12

S3: Then mark the pair (P, Q)

S4: Repeat this step until no more mark can be made. If there is a unmarked pair (P, Q) mark it if the pair $\delta(P, a)$ & $\delta(Q, a)$ is marked same input alphabet.

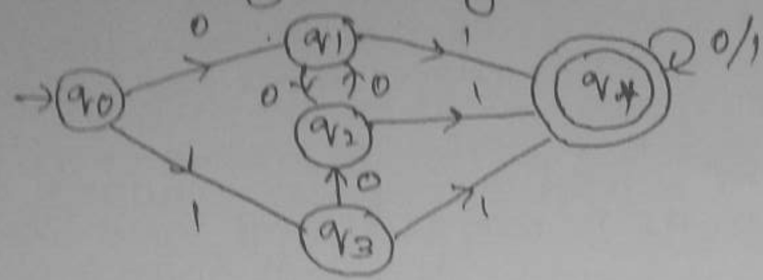
S5: combine all the unmarked pair and make them as a single state in a minimized DFA.

* <u>ALGORITHM:-</u>

begin
  for p in F & q in Q-F do mark (p, q);
  for each pair of distinct states (p, q) in F×F (or) (Q-F)×(Q-F) do
    if for some i/p symbol a, ($\delta(p, a)$, $\delta(q, a)$) is marked then
      begin
        mark (p, q);
        Recursively mark all unmarked pairs on the list for (p, q) and on the list of other pairs that are marked at this step.
      end
    else // no pair ($\delta(p, a)$ $\delta(q, a)$) is marked
    for all i/p symbol a do
      put (p, q) on the list for ($\delta(p, a)$, $\delta(q, a)$) unless $\delta(p, a) = \delta(q, a)$
end.

PROBLEMS

(1) Minimize the following DFA



Soln:-

(i) TABLE FILLING METHOD / MYHILL - NERODE THEOREM METHOD

S1:

q_1				
q_2				
q_3				
q_4^*	✓	✓	✓	✓
	q_0	q_1	q_2	q_3

S2: check (q_0, q_1)

	0	1
$\rightarrow q_0$	q_1	q_3
q_1	q_2	q_4

→ marked

$(q_0, q_1) \Rightarrow$ distinguishable

check (q_0, q_2)

	0	1
q_0	q_1	q_3
q_2	q_1	q_4

→ marked

$(q_0, q_2) \Rightarrow$ distinguishable

check (q_0, q_3)

	0	1
q_0	q_1	q_3
q_3	q_2	q_4

→ marked

$(q_0, q_3) \Rightarrow$ distinguishable

check (q_1, q_2)

	0	1
q_1	q_2	q_4
q_2	q_1	q_4

$(q_1, q_2) \Rightarrow$ equivalent state check (q_1, q_3)

	0	1
q_1	q_2	q_4
q_3	q_2	q_4

(q_1, q_3) are equivalent state

check (q_2, q_3)

	0	1
q_2	q_1	q_4
q_3	q_2	q_4

(q_2, q_3) are equivalent state

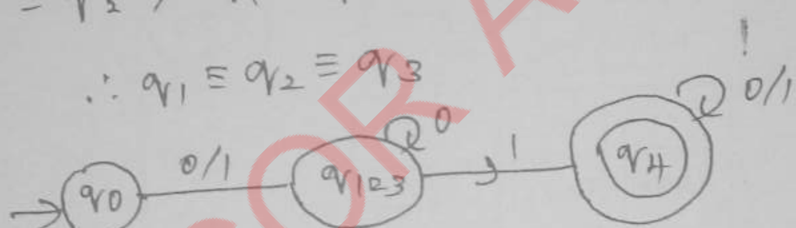
S3:

q_1	✓			
q_2	✓			
q_3				
$*q_4$	✓	✓	✓	✓
	q_0	q_1	q_2	q_3

S4: UNMARKED PAIRS,

$q_1 \equiv q_2$, $q_1 \equiv q_3$, $q_2 \equiv q_3$

$\therefore q_1 \equiv q_2 \equiv q_3$



	0	1
$\rightarrow q_0$	q_{123}	q_{123}
q_{123}	q_{123}	q_4
q_4	q_4	q_4

(ii) π-METHOD

S1: Find the π_0 state equivalence

$\pi_0 = [FS]\ [NFS]$

$= \underset{1}{\{q_4\}}\ \{\underbrace{q_0, q_1, q_2, q_3}_{2}\}$

2:

	q_0	q_1	q_2	q_3
0	q_1	q_2	q_1	q_2
1	q_3	q_4	q_4	q_4

	q_0	q_1	q_2	q_3
0	(2)	(2)	(2)	(2)
1	(2)	(1)	(1)	(1)

26

q_2: $\pi_1 = \{q_4\}_1 \{q_0\}_3 \underbrace{\{q_1, q_2, q_3\}}_{4}$

4:

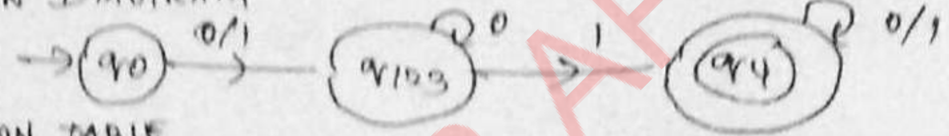
	q_1	q_2	q_3
0	q_2	q_1	q_2
1	q_4	q_4	q_4

	q_1	q_2	q_3
0	(4)	(4)	(4)
1	(1)	(1)	(1)

∴ The above table are of same. So, there is no partition.

$q_1 \equiv q_3 \equiv q_2$, q_4, q_0

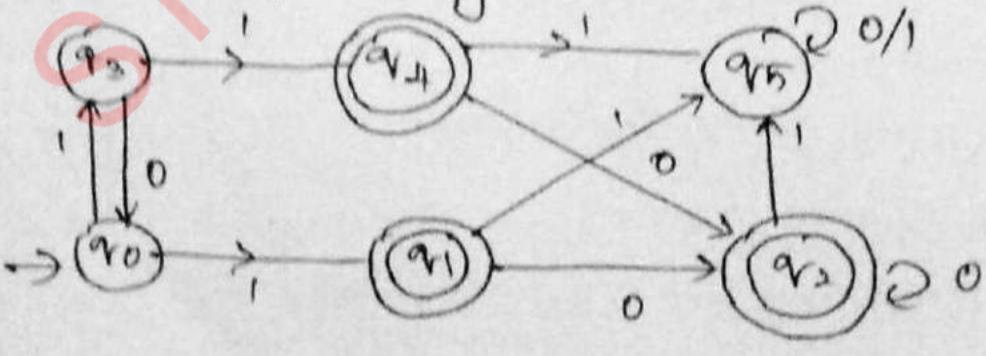
TRANSITION DIAGRAM



TRANSITION TABLE

	0	1
$\rightarrow q_0$	q_{123}	q_{123}
q_{123}	q_{123}	q_4
$*q_4$	q_4	q_4

(2) Minimize the following DFA



Soln:-

For convenience,

3:

	q_0	q_1	q_2	q_3
0	q_1	q_2	q_1	q_2
1	q_3	q_4	q_4	q_4

	q_0	q_1	q_2	q_3
0	(2)	(2)	(2)	(2)
1	(2)	(1)	(1)	(1)

S2: $\pi_1 = \underset{1}{\{q_4\}} \; \underset{3}{\{q_0\}} \; \{\underbrace{q_1, q_2, q_3}_{4}\}$

4:

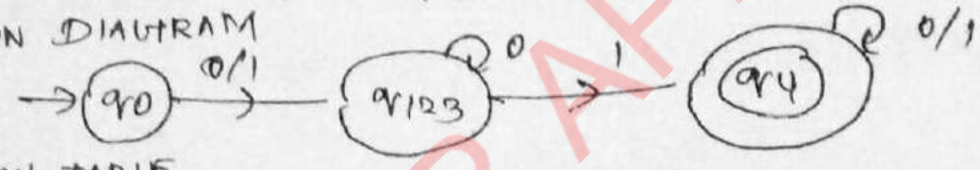
	q_1	q_2	q_3
0	q_2	q_1	q_2
1	q_4	q_4	q_4

	q_1	q_2	q_3
0	(4)	(4)	(4)
1	(1)	(1)	(1)

∴ The above table are of same. So, there is no partition.

$q_1 \equiv q_2 \equiv q_3$, q_4 , q_0

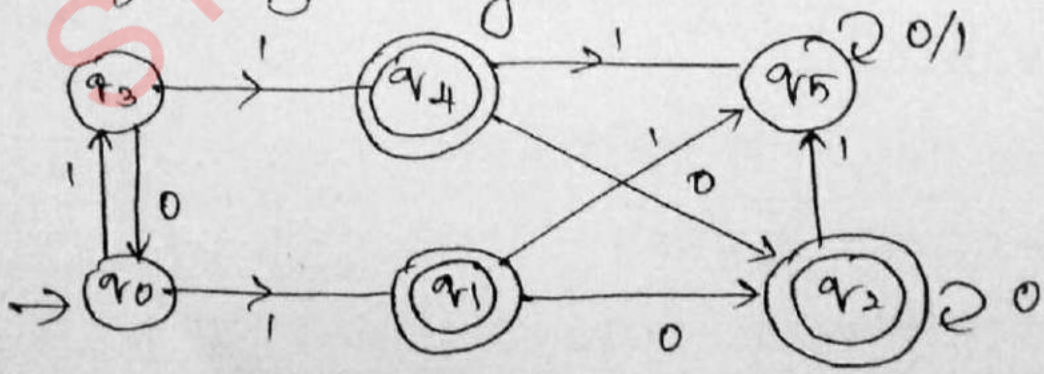
TRANSITION DIAGRAM



TRANSITION TABLE

	0	1
$\rightarrow q_0$	q_{123}	q_{123}
q_{123}	q_{123}	q_4
$*q_4$	q_4	q_4

(2) Minimize the following DFA



Soln:-

For convenience,

14

δ	0	1
$\rightarrow q_0$	q_3	q_1
$*q_1$	q_2	q_5
$*q_2$	q_2	q_5
q_3	q_0	q_4
$*q_4$	q_2	q_5
q_5	q_5	q_5

S1: Find the π_0 state equivalent

$$\pi_0 = \{FS\}\{NFS\}$$

$$= \underbrace{\{q_1, q_2, q_4\}}_{(1)} \underbrace{\{q_0, q_3, q_5\}}_{(2)}$$

check (1):

	q_1	q_2	q_4
0	q_2	q_2	q_2
1	q_5	q_5	q_5

	q_1	q_2	q_4
0	(1)	(1)	(1)
1	(2)	(2)	(2)

No need to split $q_1 \equiv q_2 \equiv q_4$. All are equivalent

check (2):

	q_0	q_3	q_5
0	q_3	q_0	q_5
1	q_1	q_4	q_5

	q_0	q_3	q_5
0	(2)	(2)	(2)
1	(1)	(1)	(2)

S2: $\pi_1 = \underbrace{\{q_1, q_2, q_4\}}_{(1)} \underbrace{\{q_0, q_3\}}_{(3)} \underbrace{\{q_5\}}_{(4)}$

check (3):

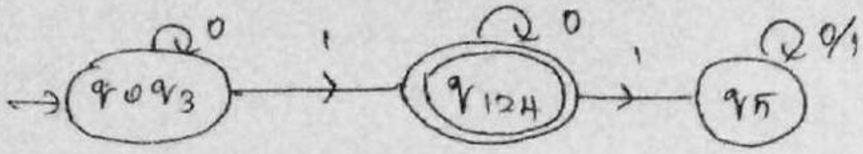
	q_0	q_3
0	q_3	q_0
1	q_1	q_4

	q_0	q_3
0	3	3
1	1	1

$\{q_0, q_3\}$ are equivalent states.

$$\boxed{\pi_k = \pi_{k-1}}$$

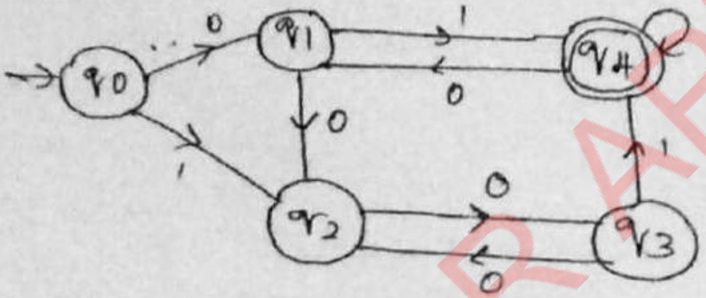
TRANSITION DIAGRAM



TRANSITION TABLE

δ	0	1
$\rightarrow q_{03}$	q_{03}	q_{124}
$* q_{124}$	q_{124}	q_5
q_5	q_5	q_5

(3) MINIMIZE THE FOLLOWING DFA



Soln:-

(using Table filling method).

S1:

q_1				
q_2				
q_3				
$*q_4$	✓	✓	✓	✓
	q_0	q_1	q_2	q_3

S2: check (q_0, q_1)

	0	1
q_0	q_1	q_2
q_1	q_2	q_4

→ marked

(q_0, q_1) = distinguishable

check (q_0, q_2)

	0	1
q_0	q_1	q_2
q_2	q_3	q_2

(q_0, q_2) are equivalent state

check (q_0, q_3)

	0	1
q_0	q_1	q_2
q_3	q_2	q_4

(q_0, q_3) = distinguishable

check (q_1, q_2)

	0	1
q_1	q_2	q_4
q_2	q_3	q_2

→ marked.

(q_1, q_2) = distinguishable

check (q_1, q_3)

	0	1
q_1	q_2	q_4
q_3	q_2	q_4

(q_1, q_3) is equivalent state

check (q_2, q_3)

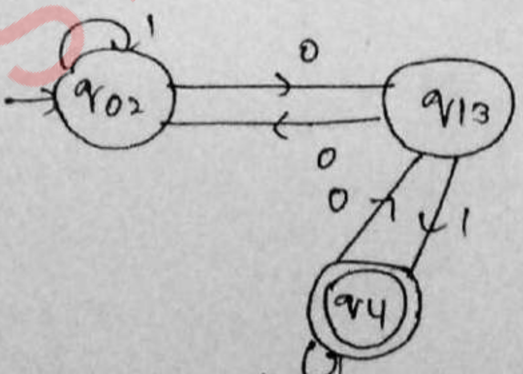
	0	1
q_2	q_3	q_2
q_3	q_2	q_4

→ marked

(q_2, q_3) = distinguishable

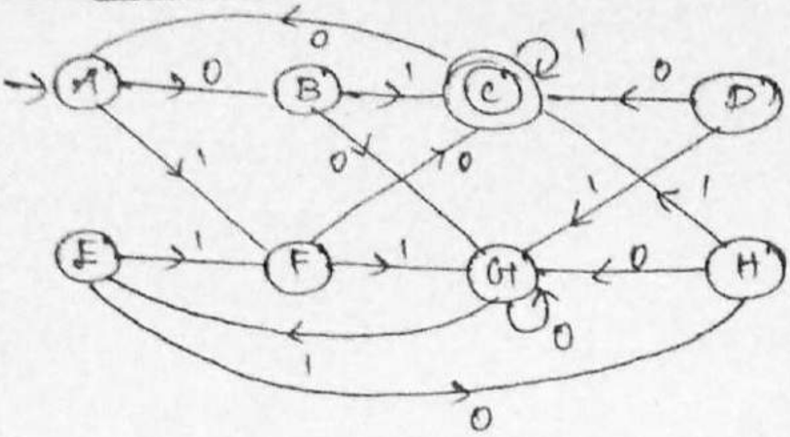
q_1	✓			
q_2				
q_3	✓	✓	✓	
* q_4	✓	✓	✓	✓
	q_0	q_1	q_2	q_3

S3: UNMARKED STATE : $q_0 \equiv q_2$, $q_1 \equiv q_3$, q_4.



	0	1
$\rightarrow q_{02}$	q_{13}	q_{02}
q_{13}	q_{02}	q_4
$* q_4$	q_{13}	q_4

(4)



Soln:-

For convenience,

δ	0	1
→ A	B	F
B	G	C
* C	A	C
D	C	G
E	H	F
F	C	G
G	G	E
H	G	C

By π - method.

S1: $\pi_0 = \{FS\} \{NFS\}$

$\pi_0 = \underbrace{\{C\}}_{1} \underbrace{\{A, B, D, E, F, G, H\}}_{2}$

check (2)

	A	B	D	E	F	G	H
0	B	G	C	H	C	G	G
1	F	C	G	F	G	E	C

	A	B	C	D	E	F	G	H
0	(2)	(2)	(1)	(1)	(2)	(1)	(2)	(2)
1	(2)	(1)		(2)	(2)	(2)	(2)	(1)

S2: $\pi_1 = \underbrace{\{C\}}_{1} \underbrace{\{A, E, G\}}_{3} \underbrace{\{B, H\}}_{4} \underbrace{\{D, F\}}_{5}$

check (3)

	A	E	G
0	B	H	G
1	F	F	E

	A	E	G
0	(4)	(4)	(3)
1	(5)	(5)	(3)

check (4)

	B	H
0	G	G
1	C	C

	B	H
0	(3)	(3)
1	(1)	(1)

check (5)

	D	F
0	C	C
1	G	G

	D	F
0	(1)	(1)
1	(3)	(3)

$\Pi_2 = \{C\}\ \{B,H\}\ \{D,F\}\ \{A,E\}\ \{G\}$

1 6 5 8 9

check (6):

	B	H
0	G	G
1	C	C

	B	H
0	(9)	(9)
1	(1)	(1)

check (7):

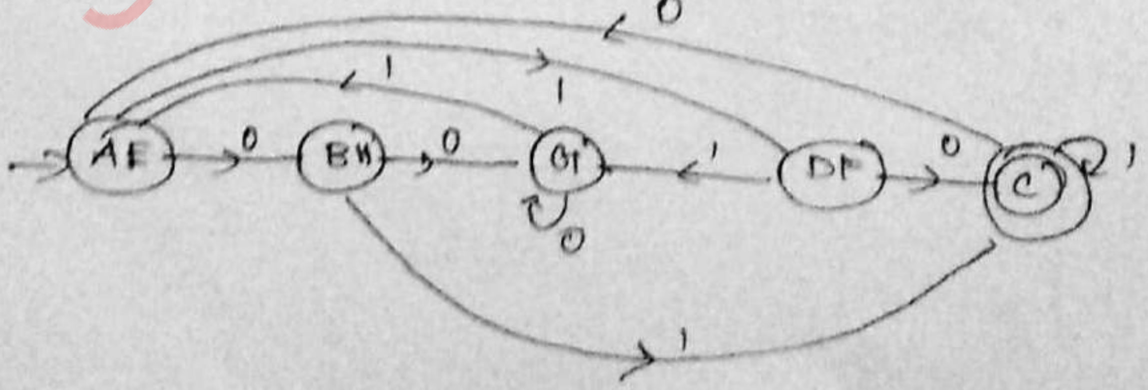
	A	E
0	B	H
1	F	F

	A	E
0	(6)	(6)
1	(7)	(7)

TRANSITION TABLE

δ	0	1
*C	AE	C
DF	C	G
G	G	AE
BH	G	C
→AE	BH	DF

TRANSITION DIAGRAM



REGULAR EXPRESSION INTO MINIMIZED DFA - 33

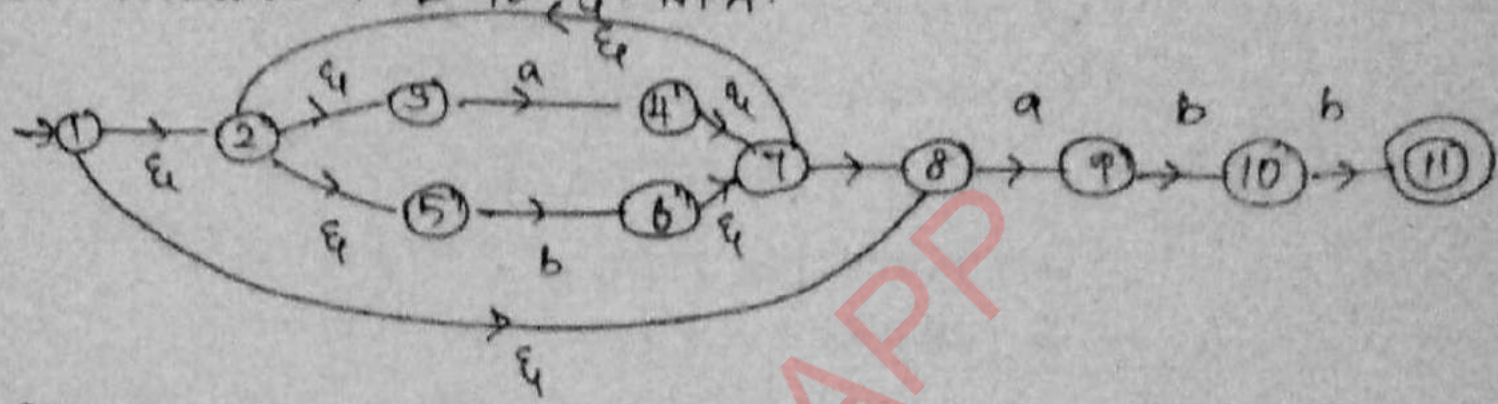
S1: Convert Regular Expression into ε-NFA using Thomson's Rule.

S2: Convert ε-NFA to DFA directly

S3: Compute minimized DFA.

① Convert the Regular Expression into minimized DFA $(a+b)^* abb$.

S1: Convert R.E to ε-NFA.



S2: ε-NFA to DFA

S2.1: Compute ε-closure of each state.

ε-closure (1) = {1, 2, 3, 5, 8}

ε-closure (2) = {2, 3, 5}

ε-closure (3) = {3}

ε-closure (4) = {2, 3, 4, 5, 7, 8}

ε-closure (5) = {5}

ε-closure (6) = {2, 3, 5, 6, 7, 8}

ε-closure (7) = {2, 3, 5, 7, 8}

ε-closure (8) = {8}

ε-closure (9) = {9}

ε-closure (10) = {10}

ε-closure (11) = {11}

S2.2: Take the ε-closure of initial state

ε-closure (1) = {1, 2, 3, 5, 8}

Step 3: Write the transition of input alphabet a, b.

a	b
$3 \to 4$	$5 \to 6$
$8 \to 9$	$9 \to 10$
	$10 \to 11$

Step 4: Find the extended transition.

$\hat{\delta}(A, a) = \epsilon$- closure $(\delta(\hat{\delta}(A, \epsilon), a))$
$= \epsilon$- closure $(\delta(\{1, 2, 3, 5, 8\}, a))$
$= \epsilon$- closure $(\{4, 9\})$
$= \epsilon$- closure $(4) \cup \epsilon$- closure (9)
$= \{2, 3, 4, 5, 7, 8, 9\} \to B$

$\hat{\delta}(A, b) = \epsilon$- closure $(\delta(\hat{\delta}(A, \epsilon), b))$
$= \epsilon$- closure $(\delta(\{1, 2, 3, 5, 8\}, b))$
$= \epsilon$- closure $(\{6\})$
$= \{2, 3, 5, 6, 7, 8\} \to C$

$\hat{\delta}(B, a) = \epsilon$- closure $(\delta(\hat{\delta}(B, \epsilon), a))$
$= \epsilon$- closure $(\delta(\{2, 3, 4, 7, 8, 9\}, a))$
$= \epsilon$- closure $(\{4, 9\})$
$= \epsilon$- closure $(4) \cup \epsilon$- closure (9)
$= \{2, 3, 4, 5, 7, 8, 9\} \to B$.

$\hat{\delta}(B, b) = \epsilon$- closure $(\delta(\hat{\delta}(B, \epsilon), b))$
$= \epsilon$- closure $(\delta(\{2, 3, 4, 5, 7, 8, 9\}, b))$
$= \epsilon$- closure $(\{6, 10\})$
$= \epsilon$- closure $(6) \cup \epsilon$- closure (10)
$= \{2, 3, 5, 6, 7, 8, 10\} \to D$

$\hat{\delta}(C, a) = \epsilon$- closure $(\delta(\hat{\delta}(C, \epsilon), a))$
$= \epsilon$- closure $(\delta(\{2, 3, 5, 6, 7, 8\}, a))$
$= \epsilon$- closure $(\{4, 9\})$

$= \epsilon\text{-closure}(4) \cup \epsilon\text{-closure}(9)$

$= \{2, 3, 4, 5, 7, 8, 9\} \rightarrow B$

$\hat{\delta}(C, b) = \epsilon\text{-closure}(\delta(\hat{\delta}(C, \epsilon), b))$

$= \epsilon\text{-closure}(\delta(\{2, 3, 5, 6, 7, 8\}, b))$

$= \epsilon\text{-closure}(\{6\})$

$= \{2, 3, 5, 6, 7, 8\} \rightarrow C$

$\hat{\delta}(D, a) = \epsilon\text{-closure}(\delta(\hat{\delta}(D, \epsilon), a))$

$= \epsilon\text{-closure}(\delta(\{2, 3, 5, 6, 7, 8, 10\}, a))$

$= \epsilon\text{-closure}(\{4, 9\})$

$= \epsilon\text{-closure}(4) \cup \epsilon\text{-closure}(9)$

$= \{2, 3, 4, 5, 7, 8, 9\} \rightarrow B$

$\hat{\delta}(D, b) = \epsilon\text{-closure}(\delta(\hat{\delta}(D, \epsilon), b))$

$= \epsilon\text{-closure}(\delta(\{2, 3, 5, 6, 7, 8, 10\}, b))$

$= \epsilon\text{-closure}(\{6, 11\})$

$= \epsilon\text{-closure}(6) \cup \epsilon\text{-closure}(11)$

$= \{2, 3, 5, 6, 7, 8, 11\} \rightarrow E$

$\hat{\delta}(E, a) = \epsilon\text{-closure}(\delta(\hat{\delta}(E, \epsilon), a))$

$= \epsilon\text{-closure}(\delta(\{2, 3, 5, 6, 7, 8, 11\}, a))$

$= \epsilon\text{-closure}(\{4, 9\})$

$= \{2, 3, 4, 5, 7, 8, 9\} \rightarrow B$

$\hat{\delta}(E, b) = \epsilon\text{-closure}(\delta(\hat{\delta}(E, \epsilon), b))$

$= \epsilon\text{-closure}(\delta(\{2, 3, 5, 6, 7, 8, 11\}, b))$

$= \epsilon\text{-closure}(\{6\})$

$= \{2, 3, 5, 6, 7, 8\} \rightarrow C$

S2.5 : Transition Table

δ	a	b
→A	B	C
B	B	D
C	B	C
D	B	E
*E	B	C

S3: compute minimized DFA (Myhill)

B				
C				
D				
*E	✓	✓	✓	✓
	A	B	C	D

i) check (A, B)

	0	1
A	B	C
B	B	D

(A, B) are distinguishable

ii) check (A, C)

	0	1
A	B	C
C	B	C

(A, C) are equivalent

iii) check (A, D)

	0	1
A	B	C
D	B	E

(A, D) are distinguishable

iv) check (B, C)

	B	C
0	B	D
1	B	C

(B, C) are distinguishable

v) check (B, D)

	B	D
0	B	D
1	B	E

(B, D) are distinguishable

vi) check (C, D)

	C	D
0	B	C
1	B	E

(C, D) are distinguishable

	A	B	C	D
B	✓			
C	≡	✓		
D	✓	✓	✓	
*E	✓	✓	✓	✓

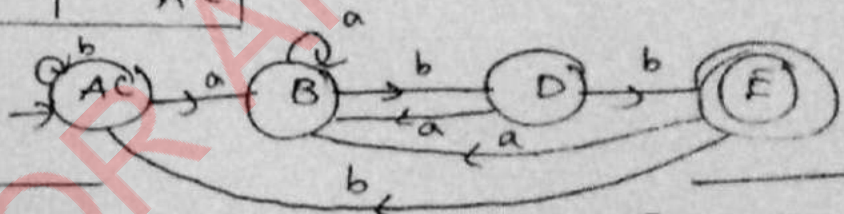
UNMARKED PAIR:-

A ≡ C

TRANSITION TABLE:-

	a	b
→AC	B	AC
B	B	D
D	B	E
*E	B	AC

TRANSITION DIAGRAM :

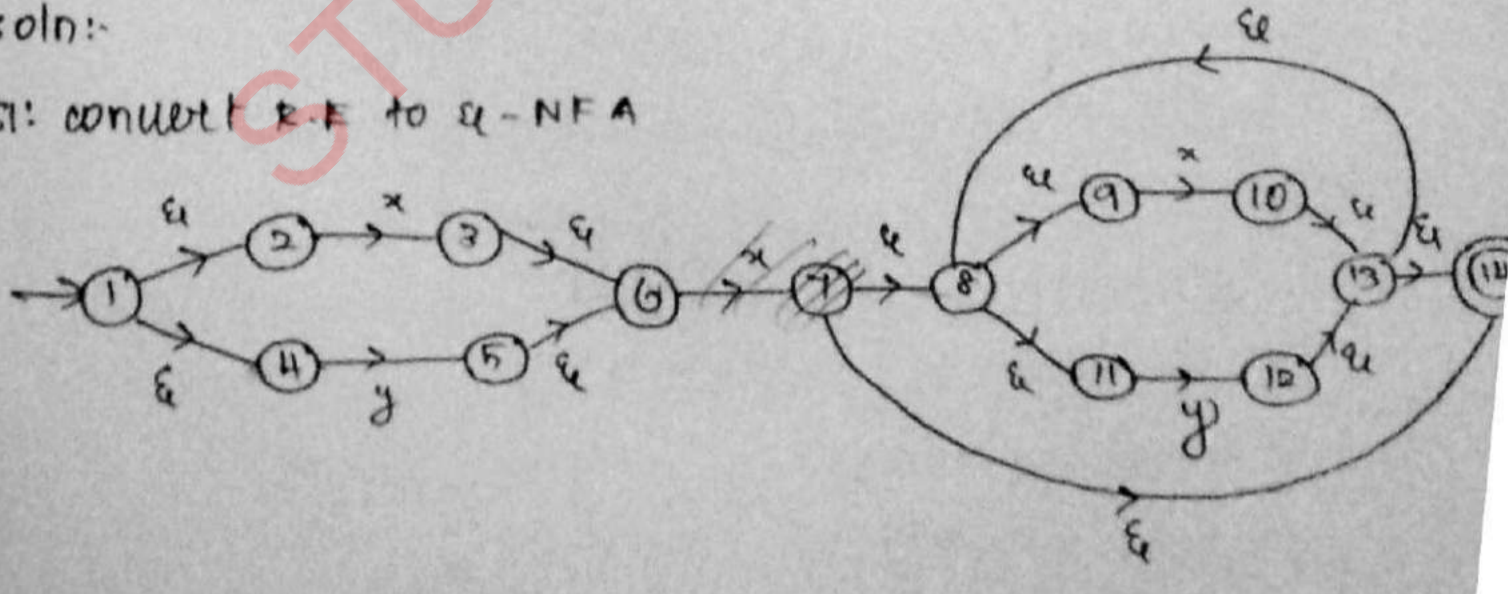


② convert Regular Expression into Minimize DFA

$(x+y)^* x (x+y)^*$

soln:-

S1: convert R.E to ε-NFA



S2: ε-NFA to DFA

S2.1: compute ε-closure of each state

ε-closure (1) = {1, 2, 4}

ε-closure (2) = {2}

ε-closure (3) = {3, 6}

ε-closure (4) = {4}

ε-closure (5) = {5, 6}

ε-closure (6) = {6}

ε-closure (7) = {7, 8, 9, 11, 14}

ε-closure (8) = {8, 9, 11}

ε-closure (9) = {9}

ε-closure (10) = {10, 13, 8, 9, 11, 14}

ε-closure (11) = {11}

ε-closure (12) = {12, 13, 8, 14, 9, 11}

ε-closure (13) = {13, 14, 8, 9, 11}

ε-closure (14) = {14}

S2.2: Take the ε-closure of initial state

ε-closure (1) = {1, 2, 4} → (A)

S2.3: write the transition of input alphabets x, y.

x	y
2 → 3	4 → 5
6 → 7	11 → 12
9 → 10	

S2.4: Find the extended transition

$\hat{\delta}(A, x) = \epsilon\text{-closure}(\delta(\hat{\delta}(A, \epsilon), x))$

$= \epsilon\text{-closure}(\delta(\{1, 2, 4\}, x))$

$= \epsilon\text{-closure}(\{3\})$

$= \{3, 6\} \rightarrow (B)$

$\hat{\delta}(A, y) = \epsilon\text{-closure}(\delta(\hat{\delta}(A, \epsilon), y))$

$= \epsilon\text{-closure}(\delta(\{1, 2, 4\}, y))$

$= \epsilon\text{-closure}(\{5\})$

$= \{5, 6\} \rightarrow (C)$

$$\hat{\delta}(B,x) = \epsilon\text{-closure}(\delta(\hat{\delta}(B,\epsilon),x))$$
$$= \epsilon\text{-closure}(\delta(\{3,6\},x))$$
$$= \epsilon\text{-closure}(\{7\})$$
$$= \{7,8,9,11,14\} \rightarrow D$$

$$\hat{\delta}(B,y) = \epsilon\text{-closure}(\delta(\hat{\delta}(B,\epsilon),y))$$
$$= \epsilon\text{-closure}(\delta(\{3,6\},y))$$
$$= \phi$$

$$\hat{\delta}(C,x) = \epsilon\text{-closure}(\delta(\hat{\delta}(C,\epsilon),x))$$
$$= \epsilon\text{-closure}(\delta(\{5,6\},x))$$
$$= \epsilon\text{-closure}(\{7\})$$
$$= \{7,8,9,11,14\} \rightarrow D$$

$$\hat{\delta}(C,y) = \epsilon\text{-closure}(\delta(\hat{\delta}(C,\epsilon),y))$$
$$= \epsilon\text{-closure}(\delta(\{5,6\},y))$$
$$\cancel{= \epsilon\text{-closure}(\{7\})}$$
$$\cancel{= \{7,8,9,11,14\}} \rightarrow$$
$$= \phi$$

$$\hat{\delta}(D,x) = \epsilon\text{-closure}(\delta(\hat{\delta}(D,\epsilon),x))$$
$$= \epsilon\text{-closure}(\delta(\{7,8,9,11,14\},x))$$
$$= \epsilon\text{-closure}(\{10\})$$
$$= \{8,9,10,11,13,14\} \rightarrow E$$

$\hat{\delta}(D, y) = \varepsilon\text{-closure}(\delta(\hat{\delta}(D, \varepsilon), y))$

$= \varepsilon\text{-closure}(\delta(\{7, 8, 9, 11, 14\}, y))$

$= \varepsilon\text{-closure}(12)$

$= \{8, 9, 11, 12, 13, 14\} \rightarrow F$

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

$= \varepsilon\text{-closure}(\delta(\{8, 9, 10, 11, 13, 14\}, x))$

$= \varepsilon\text{-closure}(\{10\})$

$= \{8, 9, 10, 11, 13, 14\} \rightarrow E$

$\hat{\delta}(E, y) = \varepsilon\text{-closure}(\delta(\hat{\delta}(E, \varepsilon), y))$

$= \varepsilon\text{-closure}(\delta(\{8, 9, 10, 11, 13, 14\}, y))$

$= \varepsilon\text{-closure}\{12\}$

$= \{8, 9, 11, 12, 13, 14\} \rightarrow F$

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

$= \varepsilon\text{-closure}(\delta(\{8, 9, 11, 12, 13, 14\}, x))$

$= \varepsilon\text{-closure}(\{10\})$

$= \{8, 9, 10, 11, 13, 14\} \rightarrow E$

$\hat{\delta}(F, y) = \varepsilon\text{-closure}(\delta(\hat{\delta}(F, \varepsilon), y))$

$= \varepsilon\text{-closure}(\delta(\{8, 9, 11, 12, 13, 14\}, y))$

$= \varepsilon\text{-closure}(\{12\})$

$= \{8, 9, 11, 12, 13, 14\} \rightarrow F$

S2.5 : Transition Table

q	x	y
→A	B	C
B	D	ϕ
C	D	ϕ
*D	E	F
*E	E	F
*F	E	F

S3 : compute minimized DFA (π-method)

π_0 = [FS] [NFS]

$= \underbrace{\{D, E, F\}}_{1} \quad \underbrace{\{A, B, C\}}_{2}$

check {D, E, F}

	D	E	F
x	E	E	E
y	F	F	F

	D	E	F
x	(1)	(1)	(1)
y	(1)	(1)	(1)

check {A, B, C}

	A	B	C
x	B	D	D
y	C	ϕ	ϕ

	A	B	C
x	(2)	(1)	(1)
y	(2)		

π_1 = $\underset{1}{\{DEF\}}$ $\underset{3}{\{A\}}$ $\underset{4}{\{BC\}}$

	B	C
x	D	∅
y	D	∅

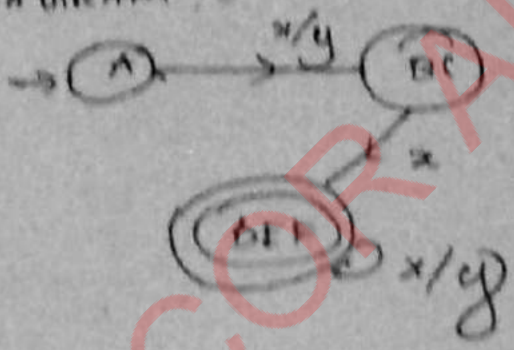
	B	C
x	(1)	
y	(1)	

$\pi_3 = \{DEF\}\ \{A\}\ \{BC\}$

TRANSITION TABLE :

	x	y
→A	BC	BC
BC	DEF	∅
*DEF	DEF	DEF

TRANSITION DIAGRAM :-



PROVING LANGUAGE NOT TO BE REGULAR

PUMPING LEMMA

Pumping Lemma is used to check whether the given string is accepted by regular set (or) not.

APPLICATION: It is used to check whether the language is regular (or) not.

THEOREM:-

Let L be a regular language (or) set then there is a constant 'n' such that if 'z' is any word (or) any string in L and $|z| \geq n$, then we can write $z = uvw$ such that

(i) $|uv| \leq n$

(ii) $|v| \geq 1$

(iii) for all $i \geq 0$, $uv^iw \in L$

PROBLEMS

Show that the given language is regular (or) not

soln:- ① $L = \{a^n b^n \,/\, n \geq 1\}$

S1: Assume L is regular language / set

S2: Identify the language.

$L = \{ab, a^2b^2, a^3b^3, \ldots\}$

S3: Take any one above string as z

$z = a^2b^2$

S4: Write Pumping Lemma Theorem,

if $|z| \geq n$ then $z = uvw$,

To prove (i) $|uv| \leq n$

(ii) $|v| \geq 1$

(iii) for all $i \geq 0$, then $z = uv^i w \in L$

S5: $|z| \geq n$

$|aabb| \geq 2$

$4 \geq 2$ (True)

$z = a|a|bb$ (u, v, w)

(i) $|uv| = |aa| = 2 \leq 2$ (True)

(ii) $|v| = |a| = 1 \geq 1$ (True)

(iii) for all $i=0$, $uv^i w \in L$

$a(a)^0 bb$

$abb \notin L$ (false)

for all $i=1$, $uv^i w \in L$

$a(a)^1 bb \in L$ (True)

for all $i=2$, $uv^i w \in L$

$a(a)^2 bb \notin L$

$aaabb \notin L$ (false)

∴ The given language is not regular.

---

(2) $L = \{ww / w \in (0,1)^*\}$

soln:-

S1: Assume L is regular language (or) set

S2: Identify the language

$L = \{00, 11, 1010, 10011001 \ldots\}$

S3: Take $z = 1010$.

S4: If $|z| \geq n$, then $z = uvw$

To prove, (i) $|uv| \leq n$

(ii) $|v| \geq 1$

(iii) for all $i \geq 0$, $z = uv^iw \in L$

S5: $|z| \geq n$

$|1010| \geq 2$

$4 \geq 2$ (True)

$z = \underset{u}{1} | \underset{v}{0} | \underset{w}{10}$

(i) $|uv| = |10| = 2 \leq 2$ (True)

(ii) $|v| = |0| = 1 \geq 1$ (True)

(iii) for all $i = 0$, $uv^iw \in L$

$1(0)^0 10 \notin L$

$110 \notin L$ (false)

for all $i = 2$, $uv^iw \in L$

$1(0)^2 10 \notin L$

$10010 \notin L$ (false)

$\therefore$ The given language is not regular.

③ $L = \{a^{i^2} / i \geq 1\}$

soln:-

S1: Assume L is a regular language (or) set

S2: Identify the language

$L = \{a^1, a^4, a^9, \dots\}$

S3: Take $z = a^4$

S4: Write pumping Lemma Theorem.

If $|z| \geq n$ then $z = uvw$

To prove (i) $|uv| \leq n$

(ii) $|v| \geq 1$

(iii) for all $i \geq 0$, then $z = uv^iw \in L$

S5: $|z| \geq n$

$|aaaa| \geq 2$

$4 \geq 2$ (True)

$$z = \underset{u}{a} | \underset{v}{a} | \underset{w}{aa}$$

(i) $|uv| = |aa| = 2 \leq 2$ (True)

(ii) $|v| = |a| = 1 \geq 1$ (True)

(iii) for all $i = 0$, $uv^iw \in L$

$a(a^0)aa \notin L$

$aaa \notin L$ (False)

for all $i = 1$, $uv^iw \in L$

$a(a^1)aa \in L$

$aaaa \in L$ (True)

for all $i = 2$, $uv^iw \in L$

$a(a^2)aa \notin L$

$aaaaa \notin L$ (False)

$\therefore$ The given language is not regular.

---

(4) $L = \{a^p / p \text{ is a prime}\}$

soln:-

S1: Assume L is a regular language / set

S2: Identify the language.

$L = \{a^2, a^3, a^5, a^7 \ldots\ldots\}$

S3: Take $z = a^3$

S4: Write the Pumping Lemma Theorem

If $|z| \geq n$ then $z = uvw$

to prove (i) $|uv| \leq n$

(ii) $|v| \geq 1$

(iii) for all $i \geq 0$, $z = uv^iw \in L$

S5: $|z| \geq n$

$|aaa| \geq 3$

$3 \geq 3$ (True)

$$z = \underset{u}{a} | \underset{v}{a} | \underset{w}{a}$$

(i) $|uv| = |aa| = 2 \leq 3$ (True)

(ii) $|v| = |a| = 1 \geq 1$ (True)

(iii) for all $i = 0$, $uv^iw \in L$

$a(a)^0 a \in L$

$aa \in L$ (True)

for all $i = 1$, $uv^iw \in L$

$a(a)^1 a \in L$

$aaa \in L$ (True)

for all $i = 2$, $uv^iw \in L$

$a(a)^2 a \in L$

$aaaa \notin L$ (False)

$\therefore$ The given language is not regular

---

(5) $L = \{ww^R / w \in (0,1)^*\}$

soln:-

S1: Assume L is a regular language / set.

S2: Identify the language

$L = \{0110, 1001, \ldots\}$

S3: Take $z = 1001$

S4: Write the pumping lemma Theorem,

If $|z| \geq n$, then $z = uv^iw$

to prove: (i) $|uv| \leq n$

(ii) $|v| \geq 1$

(iii) for all $i \geq 0$, $z = uv^iw \in L$

SR: $|z| \geq n$

$|1001| \geq 2$ (True)

$4 \geq 2$ (True)

$$z = \underset{u}{1} \,|\, \underset{v}{0} \,|\, \underset{w}{01}$$

(i) $|uv| = |10| = 2 \leq 2$ (True)

(ii) $|v| = |0| = 1 \geq 1$ (True)

(iii) for all $i = 0$, $uv^i w \in L$

$1(0)^0 01 \in L$

$101 \notin L$ (False)

for all $i = 1$, $uv^i w \in L$

$1(0)^1 01 \in L$

$1001 \in L$ (True)

for all $i = 2$, $uv^i w \in L$

$1(0)^2 01 \in L$

$10001 \notin L$ (False)

$\therefore$ The given language is not regular

CLOSURE PROPERTIES OF REGULAR LANGUAGE.

If certain languages are Regular and a Language 'L' is formed from them by the certain operations. (Eg: L is the Union of two Regular Language) then L is also Regular) These theorems are often called Closure Property of Regular Language.

Let L and M be a Regular Language then the following languages are all regular.

1) The Union of two Regular Language is Regular: $L \cup M$
2) The Intersection of two Regular Language is Regular: $L \cap M$
3) The Complement of a Regular Language is Regular: $\overline{N}$
4) The Difference of two Regular Language is Regular: $L - N$
5) The Reversal of Regular Language is Regular: $L^R = w^R ;$ $w \in L$
6) The Closure star of Regular Language is Regular: L^*
7) The Concatenation of Regular Language is Regular: LM
8) The Homomorphism [substitution of a String for symbols] of a Regular Language is Regular.

$$h(L) = \{ h(w) / w \in L, \text{ h is homomorphism} \}$$

9) The Inverse Homomorphism of a Regular Language is Regular.

$$h^{-1}(L) = \{ w \in \Sigma / h(w) L, \ h : \Sigma \to \Delta \text{ is a homomorphism} \}$$

I. CLOSURE UNDER UNION:

THEOREM - 1 :

If L and M be Regular Language then $L \cup M$ is also a Regular.

PROOF :

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ be an NFA such that $L = L(N_1)$

$||^{ly}$ $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ be an NFA such that : $M = L(N_2)$

We may assume that, $Q_1 \cap Q_2 = \phi$

From these two NFA's, we will construct an

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

such that : $L(N) = L \cup M$.

Then NFA N is defined as follows :

1) $Q = \{q_0\} \cup Q_1 \cup Q_2$

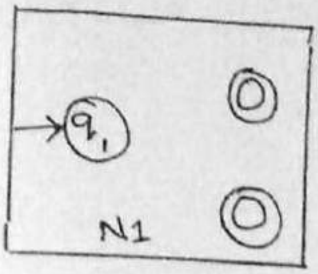
2) Start State $= q_0$

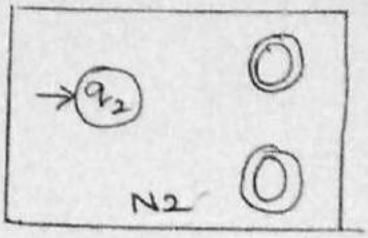
3) $F = F_1 \cup F_2$

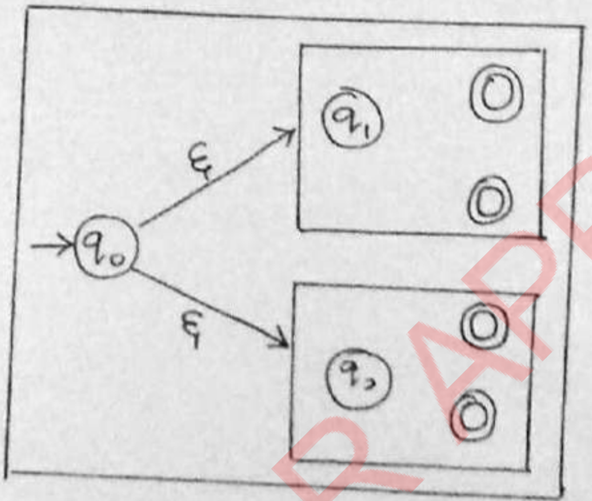
4) $\delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow 2^Q$ is defined as follows:

For any $r \in Q$ and for any $a \in \Sigma \cup \{\epsilon\}$

$$\delta(r,a) = \begin{cases} \delta_1(r,a) & , \text{ if } r \in Q_1 \\ \delta_2(r,a) & , \text{ if } r \in Q_2 \\ \{q_1, q_2\} & , \text{ if } r = q_0 \text{ and } a = \epsilon \\ \phi & , \text{ if } r = q_0 \text{ and } a \neq \epsilon \end{cases}$$







$\therefore$ NFA N accepts $L(N_1) \cup L(N_2)$

II. CLOSURE UNDER CONCATENATION:

THEOREM - 2:

If L and M are Regular Language over the some alphabet Σ, then LM is also a Regular Language.

PROOF:

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ be an NFA

such that $L = L(N_1)$

$||^{ly}$ $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ be an NFA

such that $M = L(N_2)$

We assume that, $Q_1 \cap Q_2 = \phi$

We will construct a new NFA $N = (Q, \Sigma, \delta, q_0, F)$

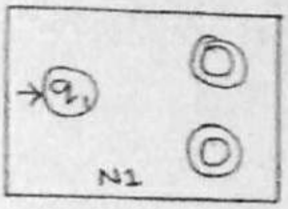
Such that $L(N) = LM$

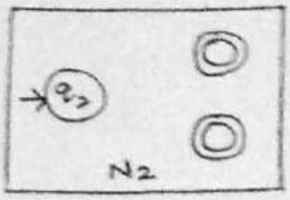
NFA N is defined as follows:

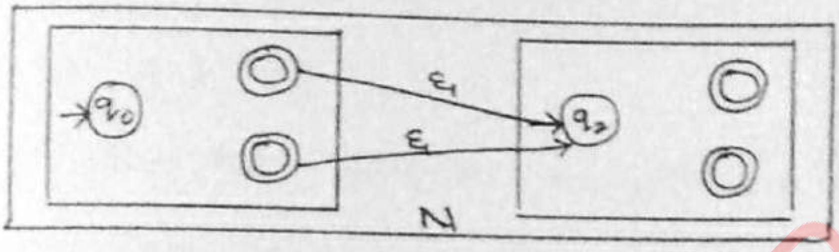
1) $Q = Q_1 \cup Q_2$
2) $q_0 = q_1$
3) $F = F_2$
4) $\delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow 2^Q$ is defined as follows.

For any $r \in Q$ and for any $a \in \Sigma \cup \{\epsilon\}$

$$\delta(r,a) = \begin{cases} \delta_1(r,a) & , \text{ if } r \in Q_1 \text{ and } r \notin F_1 \\ \delta_1(r,a) & , \text{ if } r \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(r,a) \cup \{q_2\} & , \text{ if } r \in F_1 \text{ and } a = \varepsilon \\ \delta_2(r,a) & , \text{ if } r \in Q_2 \end{cases}$$







$\therefore$ The NFA N accepts $L(N_1) \cdot L(N_2)$

III. CLOSURE UNDER COMPLEMENTATION:

THEOREM-3:

If L is a Regular Language over alphabet Σ then $\overline{L} = \Sigma^* - L$ is also Regular.

PROOF:

Let L be recognized by a DFA

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

then $\overline{L} = L(B)$ where B is the DFA.

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

(i.e) B is exactly like A but the accepting states of A have become the non-accepting states of B and viceversa.

Then w is in $L(B)$ iff ~~δ~~ $\hat{\delta}(q_0, w)$ is in $Q-F$, which occurs iff w is not in $L(A)$.

IV. CLOSURE UNDER INTERSECTION:

THEOREM:

If L and M are Regular Language then so is $L \cap M$.

PROOF:

Let L be recognized by the DFA

$$A_L = (Q_L, \Sigma, \delta_L, q_L, F_L) \text{ and}$$

M by the DFA, $A_M = (Q_M, \Sigma, \delta_M, q_M, F_M)$.

We assume that the alphabets of both automata are the same; Σ is the union of alphabet of L and M if they are different.

We also assume w.l.o.g that both Automata are Deterministics.

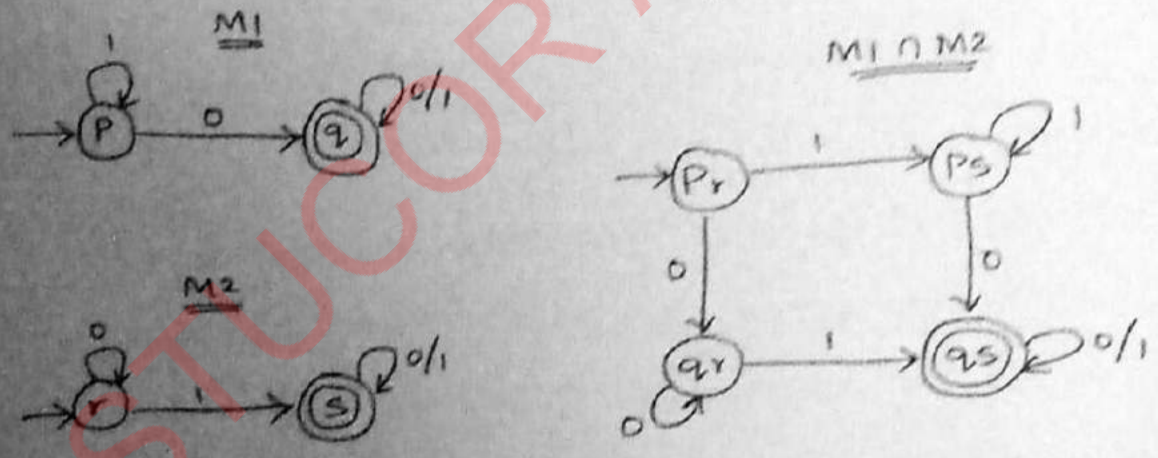
We shall Construct an automaton A that simulates both A_L and A_M.

States of A are pairs of states, the 1st from A_L and 2nd from A_M.

If A_L goes from state p to state s on the reading "a" and A_M goes from state q to state t on reading a then $A_{L \cap M}$ will go from state (p,q) to state (s,t) on reading 'a'.

Start State of A is the pair of start states of A_L and A_M.

Since we want to accept iff both automata accept we select as Accepty states of A all pairs (p,q) such that p is an Accepty states of A_L and q is Accepty States of A_M.



V. CLOSURE UNDER DIFFERENCE :

L-M the difference of L and M is the set of strings that are in language L but not in language M.

THEOREM :

If L and M are regular language then so is L–M.

PROOF :

Observe that $L - M = L \cap \overline{M}$

By the theorem-3 : If M is Regular Language then $\overline{M}$ is also Regular Language.

then we can say that $\overline{M}$ is Regular.

By the Theorem-4 : If L and M are Regular Language then $L \cap M$ is also Regular.

$\therefore L \cap \overline{M}$ is also Regular.

$\therefore L - M$ is Regular.

VI CLOSURE UNDER REVERSAL :

The reversal of a String $a_1 a_2 \ldots a_n$ is the String written backwards.

(i.e) $a_n a_{n-1} a_{n-2} \ldots a_1$

$$w = w^R$$

$$0100 = 0010^R$$

$$\epsilon^R = \epsilon$$

$L = L^R$ (consist of Reversals of all its strings)

For eg: if $L = \{001, 10, 10111\}$

$$L^R = \{100, 01, 11101\}$$

THEOREM - 6 :

If L is Regular Language then L^R is also Regular.

PROOF - I (USING FINITE AUTOMATA)

$$L = L(A)$$

(i) Reverse all the arcs in Transition Diagram for A.

(ii) Make the start state of A be the only Accepty state for the new automaton.

(iii) Create a new start state P_0 with Transition ϵ to all the accepting state of A.

(i.e) "A in reverse"

∴ Accept a String w iff A accept w^R

PROOF - II (USING REGULAR EXPRESSION):

Assume L is defined by Regular Expression E.

Structural Induction on size of E. We show that there is another Regular Expression E^R such that $L(E^R) = (L(E))^R$

(i.e) Language of E^R is the reversal of the Language of E.

* **BASICS:**

If Regular Expression E is ε, ϕ, a, then E^R is the same as E.

(i.e) $\{\varepsilon\}^R = \{\varepsilon\}$, $\phi^R = \phi$, $\{a\}^R = \{a\}$

* **INDUCTION:** 3-cases

Case 1: $E = E_1 + E_2$

$E^R = E_1^R + E_2^R$

Reversal of the union of two language is obtained by computing the reversals of the two languages and taking the union of those language.

Case 2 : $E = E_1 E_2$ then $E^R = E_1^R E_2^R$

We reverse the order of the two language as well as a reversing the language themselves.

For eg: if $L(E_1) = \{01, 110\}$

$L(E_2) = \{00, 10\}$

then $L(E_1, E_2) = \{0100, 0110, 11000, 11010\}$

$L(E_1, E_2)^R = \{0010, 0110, 00011, 01011\}$

If we concatenate the reversal of $L(E_2)$ & $L(E_1)$

$\therefore \{00, 01\}, \{10, 011\} = \{0010, 00011, 0110, 01011\}$

Same as $L(E_1 E_2)^R$.

In general, If a word w in $L(E)$ is the concatenate of w_1 from $L(E_1)$ and w_2 from $L(E_2)$ then,

$$w^R = w_2^R \cdot w_1^R$$

Case 3 : $E = E_1^*$ then $E^R = (E_1^R)^*$

(i.e) any string w in $L(E)$ can be written as $w_1 w_2 \ldots w_n$ where each w_i is in $L(E)$

But $w^R = w_n^R \cdot w_{n-1}^R \ldots w_1^R$

Each w_i^R is in $L(E^R)$, so w^R is in $(E_1^R)^*$

Conversely, any String in $L(E_1^R)^*$ is the form of $w_1 w_2 \ldots w_n$, where each w_i is the reversal of a String in $L(E_1)$.

The reversal of this String $w_n^R w_{n-1}^R \ldots w_1^R$ is

$\therefore$ A String in $L(E_1^*)$, which is $L(E)$.

We have thus shown that a String is in $L(E)$ if and only if its reversal is in $L(E_1^R)^*$.

VII. CLOSURE UNDER KLEENE CLOSURE:

THEOREM:

If L is a Regular Language then L^* is also a Regular.

PROOF:

Let L be a Regular Language and

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ be a NFA that accepts L.

To show that L^* is also a Regular.

$\therefore$ New NFA, $L^* = (Q \cup \{q_s\}, \Sigma, \delta_s, q_s, F \cup \{q_s\})$

where,

$$\delta_s(q,c) = \begin{cases} \delta(q,c) \cup \{q_0\}, & \text{if } q \in F,\ c = \epsilon \\ \delta(q,c), & \text{if } q \in F,\ c \neq \epsilon \\ \delta(q,c), & \text{if } q \in Q - F \\ \{q_0\}, & \text{if } q = q_s,\ c = \epsilon \\ \phi, & \text{if } q = q_s,\ c \neq \epsilon \end{cases}$$

VIII. CLOSURE UNDER HOMOMORPHISM :

A homomorphism is just substitution of strings for letters.

Formally a Homomorphism is a function $h : \Sigma \to \Gamma^*$

Homomorphism can be extended from letters to strings $h : \Sigma^* \to \Gamma^*$ in a straightforward manner:

$$\hat{h}(w) = \begin{cases} \epsilon & , \text{ if } w = \epsilon \\ \hat{h}(w).h(a) & , \text{ if } w = xa \end{cases}$$

We can apply homomorphism to languages as well, for a homomorphism h and a language $L \subseteq \Sigma^*$ we define $h(L) \subseteq \Gamma^*$ as

$$h(L) = \{h(w) \in \Gamma^* : w \in L \subseteq \Sigma^*\}.$$

We define inverse - homomorphism of a language $L \in \Gamma^*$ as,

$$h^{-1}(L) = \{w \in \Sigma^* : h(w) \in L \subseteq \Gamma^*\}.$$

LEMMA :

The class of regular language is closed under homomorphism.

PROOF :

Prove for arbitary regular language L and homomorphism h that $h(L)$ is a Regular Languages. Let E be REGEX accepting L.

REGEX Construction : We claim the REGEX E_h defined inductively as

$$
\begin{aligned}
E_h &= \varepsilon, && \text{if } E = \varepsilon \\
E_h &= \phi, && \text{if } E = \phi \\
E_h &= h(a), && \text{if } E = a \\
E_h &= F_h + G_h, && \text{if } E = F+G \\
E_h &= F_h . G_h, && \text{if } E = F.G \\
E_h &= (F_h)^*, && \text{if } E = F^*
\end{aligned}
$$

Accepts $h(L)$. (i.e) $L(E_h) = h(L(E))$.

Proof of Correctness: Prove that $L(E_h) = h(L(E))$

if $E = \epsilon$ then

$$LHS = L(E_h) = L(h(\epsilon)) = L(\epsilon) = \{\epsilon\}$$
$$RHS = h(L(E)) = h(L(\epsilon)) = h(\{\epsilon\}) = \{\epsilon\}$$

Similarly for $E = \phi$.

if $E = a$, then

$$LHS = L(E_h) = L(h(a)) = \{h(a)\}$$
$$RHS = h(L(E)) = h(L(a)) = h(\{a\}) = \{h\{a\}\}$$

if $E = F+G$, then

$$L(h(E)) = L(h(F+G)) = L(h(F)+h(G))$$
$$= L(h(F)) \cup L(h(G))$$
$$L(L(E)) = h(L(F+G)) = h(L(F)) \cup h(L(G))$$

From induction hypothesis, both of these expression are equal. Other inductive cases are similar, and hence omitted.

IX. CLOSURE UNDER INVERSE - HOMOMORPHISM :

LEMMA :

The class of Regular Languages is closed under homomorphism.

PROOF :

Let $A = (S, \Gamma, \delta, s_0, F)$ be a DFA accepting L and $h : \Sigma \to \Gamma^*$ be an arbitrary homomorphism. We show that the DFA $h^{-1}(A) = (S', \Sigma, \delta', s_0', F')$ defined below accepts $h^{-1}(L)$.

$$S' = S, \quad s_0' = s_0, \quad F' = F$$

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

It is an easy induction over w that $\delta'(s,w) = \hat{\delta}(s, h(w))$. Now since accepting states of A and $h^{-1}(A)$ are the same, $h^{-1}(A)$ accepts w iff A accepts $h(w)$.

UNIT-3

CONTEXT FREE GRAMMAR AND LANGUAGE

CFG - Parse trees - Ambiguity in Grammar & Languages - Definition of Pushdown Automata - Languages of PDA - Equivalence of PDA & CFG, Deterministic Pushdown Automata.

CONTEXT FREE GRAMMAR:-

Grammar describes the programming language constructs. It consist of set of terminals, set of Non-terminals, set of production Rules and a start symbol.

It is denoted by $G = (V, T, P, S)$

where

V → Variables/Non Terminals

a finite non empty set of non terminal/variables that represent a language (i.e) a set of strings. These are generally represent by capital letters A, B, C, D . . . . X, Y, Z.

T → Terminals

The finite set of symbols from which the string for the language are formed. These are generally represented by lowercase letters, digits, operators, special characters etc.

P → Production Rule

Set of rules that describe the recursive definition of a language. All productions are of the form $\alpha \to \beta$

where α : A non Terminals (i.e) $\alpha \in V$

β : A combination of T and V

(i.e) $\beta \in (V \cup T)$

S → Start Symbol

Initial non-terminal symbol in the grammar. It represent the language being defined by the grammar. It is denoted by S.

USE OF CFG:-

1) Defining the syntax of programming language
2) Used to help generating Parse trees
3) Used for defining the syntactic structure of natural language
4) Used in designing and implementation of compilers
5) Used as specification and implementation of programming language

Eg: 1

Find CFG for the given procedure $S \to AB$, $B \to b$, $A \to aB$

Soln:-

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

$V = \{S, A, B\}$ $P = \{S \to AB, A \to aB, B \to b\}$

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

Eg: 2

Find CFG for the given production $E \to E+E / E*E / (E) / id$.

soln

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

$V = \{E\}$

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

$T = \{+, *, (,), id\}$

$S = \{E\}$

DERIVATION AND SENTENTIAL FORM

DERIVATION :-

Derivation is a method of replacing the NT present on RHS of a production rule with a terminal symbol. This is the process of deriving an input string from a set of production rule starts with start symbol.

$$S \xRightarrow[G]{*} w \ / \ w \in T^*$$

SENTENTIAL FORM :-

If $A \xRightarrow{*} \beta$, then β is said to be in sentential form if β contain terminal or Non-terminal.

REPRESENTATION OF DERIVATION :-

1) Derivation form / Sentential form → LMD, → RMD

2) Parse Tree → LMPT, → RMPT } having only one Parse tree.

LEFT MOST DERIVATION (LMD):-

If at each step in a derivation a production rule is applied to the left most variable it is called as left most derivation.

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

RIGHT MOST DERIVATION (RMD):-

If at each step in a derivation a production rule is applied to the right most variable it is called as right most derivation.

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

PROBLEMS

1) Find LMD & RMD of string "00101" for the given grammar $G = (V, T, P, S)$ where

$G = (\{S, A, B\}, \{0, 1\}, P, S)$

P: $S \rightarrow \cancel{A/B}\ A1B$

$A \rightarrow 0A / \epsilon$

$B \rightarrow 0B / 1B / \epsilon$

LMD

$$S \underset{lm}{\Rightarrow} \cancel{A/B}\ A1B \quad [S \rightarrow A1B]$$
$$\underset{lm}{\Rightarrow} 0A1B \quad [A \rightarrow 0A]$$
$$\underset{lm}{\Rightarrow} 00A1B \quad [A \rightarrow 0A]$$
$$\underset{lm}{\Rightarrow} 001B \quad [A \rightarrow \epsilon]$$
$$\underset{lm}{\Rightarrow} 0010B \quad [B \rightarrow 0B]$$
$$\underset{lm}{\Rightarrow} 00101B \quad [B \rightarrow 1B]$$
$$\underset{lm}{\Rightarrow} 00101 \quad [B \rightarrow \epsilon]$$

$$S \overset{*}{\underset{lm}{\Rightarrow}} 00101$$

RMD

$$S \underset{rm}{\Rightarrow} A1\underline{B} \quad [S \to A1B]$$
$$\underset{rm}{\Rightarrow} A10\underline{B} \quad [B \to 0B]$$
$$\underset{rm}{\Rightarrow} A101\underline{B} \quad [B \to 1B]$$
$$\underset{rm}{\Rightarrow} \underline{A}101 \quad [B \to \epsilon]$$
$$\underset{rm}{\Rightarrow} 0\underline{A}101 \quad [A \to 0A]$$
$$\underset{rm}{\Rightarrow} 00A101 \quad [A \to 0A]$$
$$\underset{rm}{\Rightarrow} 00101 \quad [A \to \epsilon]$$

$$S \overset{*}{\underset{rm}{\Rightarrow}} 00101$$

2) Find LMD & RMD of $S \to aSx / b$, $x \to xb / a$ for string "aababa".

soln

LMD

$$S \underset{lm}{\Rightarrow} a\underline{S}x \quad [S \to aSx]$$
$$\underset{lm}{\Rightarrow} aa\underline{S}xx \quad [S \to aSx]$$
$$\underset{lm}{\Rightarrow} aabx\underline{x} \quad [S \to b]$$
$$\underset{lm}{\Rightarrow} aabxbx \quad [x \to xb]$$
$$\underset{lm}{\Rightarrow} aab\underline{a}bx \quad [x \to a]$$
$$\underset{lm}{\Rightarrow} aababa \quad [x \to a]$$

$$S \overset{*}{\underset{lm}{\Rightarrow}} aababa$$

RMD

$$S \underset{rm}{\Rightarrow} aS\underline{x} \quad [S \to aSx]$$
$$\underset{rm}{\Rightarrow} a\underline{S}a \quad [x \to a]$$
$$\underset{rm}{\Rightarrow} aaS\underline{x}a \quad [S \to aSx]$$
$$\underset{rm}{\Rightarrow} aaSxba \quad [x \to xb]$$

$\underset{rm}{\Rightarrow}$ aaSaba [$x \to a$]

$\underset{rm}{\Rightarrow}$ aababa [$S \to b$]

$S \underset{rm}{\overset{*}{\Rightarrow}}$ aababa

3) Find LMD & RMD of $S \to bS / aT / \epsilon$, $T \to aT / bU / \epsilon$ and $U \to aT / \epsilon$. For the string "bbaa" and also draw parse tree

soln

LMD

$S \underset{lm}{\Rightarrow} b\underline{S}$ [$S \to bS$]

$\underset{lm}{\Rightarrow} bb\underline{S}$ [$S \to bS$]

$\underset{lm}{\Rightarrow} bba\underline{T}$ [$S \to aT$]

$\underset{lm}{\Rightarrow} bbaa\underline{T}$ [$T \to aT$]

$\underset{lm}{\Rightarrow}$ bbaa [$T \to \epsilon$]

$S \underset{lm}{\overset{*}{\Rightarrow}}$ bbaa

RMD

$S \underset{rm}{\Rightarrow} b\underline{S}$ [$S \to bS$]

$\underset{rm}{\Rightarrow} bb\underline{S}$ [$S \to bS$]

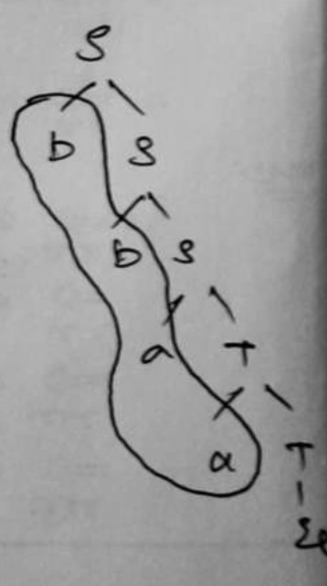
$\underset{rm}{\Rightarrow} bba\underline{T}$ [$S \to aT$]

$\underset{rm}{\Rightarrow} bbaa\underline{T}$ [$T \to aT$]

$\underset{rm}{\Rightarrow}$ bbaa [$T \to \epsilon$]

$S \underset{rm}{\overset{*}{\Rightarrow}}$ bbaa

PARSE TREE



PARSE TREE / DERIVATION TREE:-

A tree representation for the derivation of the given production rules for a CFG $G = (V, T, P, S)$ is called Derivation tree / parse tree.

PROPERTIES TO CONSTRUCT PARSE TREE:-

(i) The root node is always a start symbol of G (i.e) S

(ii) The leaf nodes are always terminal 'a' (i.e) $a \in (T \cup \{\varepsilon\})$.

(iii) The interior nodes are always non-terminals A (i.e) $A \in V$

(iv) If the children of a node A, $A \in V$ are $x_1, x_2, x_3 \ldots x_n$ then the production $A \to x_1, x_2 \ldots x_n$ is in P.

(v) When moved from left to right form a string that is derived from Root node.

Example - 1

Construct the derivation tree for a given string a^5 for the given grammar CFG $G = (V, T, P, S)$.

$S \to aSa / a$

soln:-

LMD

$$S \underset{lm}{\Rightarrow} aSa \quad [S \to aSa]$$
$$\underset{lm}{\Rightarrow} aaSaa \quad [S \to aSa]$$
$$\underset{lm}{\Rightarrow} aaaaa \quad [S \to a]$$

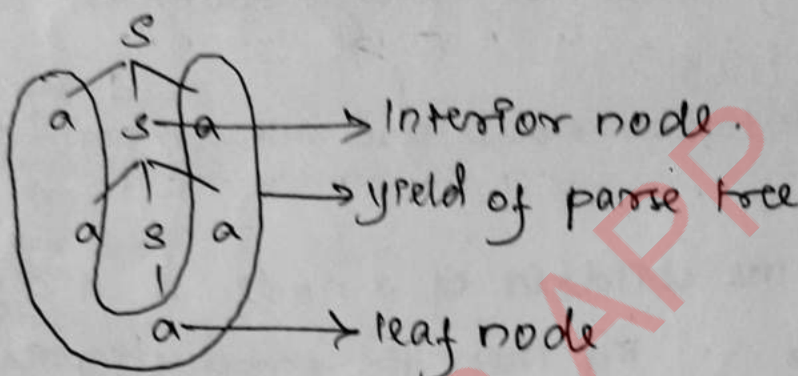
$$S \overset{*}{\underset{lm}{\Rightarrow}} aaaaa.$$

RMD

$$S \underset{rm}{\Rightarrow} a\underline{S}a \quad [S \to aSa]$$
$$\underset{rm}{\Rightarrow} aa\underline{S}aa \quad [S \to aSa]$$
$$\underset{rm}{\Rightarrow} aaaaa \quad [S \to a]$$

$$S \overset{*}{\underset{rm}{\Rightarrow}} aaaaa.$$

PARSE TREE



Example 2

Generate the parse tree for the string $w = -(id + id)$ of the grammar $E \to E+E / E*E / (E) / -E / id$.

soln:-

LMD

$$E \underset{lm}{\Rightarrow} -E \quad [E \to -E]$$
$$\underset{lm}{\Rightarrow} -(E) \quad [E \to (E)]$$
$$\underset{lm}{\Rightarrow} -(E+E) \quad [E \to E+E]$$
$$\underset{lm}{\Rightarrow} -(id+E) \quad [E \to id]$$
$$\underset{lm}{\Rightarrow} -(id+id) \quad [E \to id]$$

$$E \overset{*}{\underset{lm}{\Rightarrow}} -(id+id)$$

RMD

$$E \underset{rm}{\Longrightarrow} -E \quad [E \to -E]$$
$$\underset{rm}{\Longrightarrow} -(E) \quad [E \to (E)]$$
$$\underset{rm}{\Longrightarrow} -(E+E) \quad [E \to E+E]$$
$$\underset{rm}{\Longrightarrow} -(E+id) \quad [E \to id]$$
$$\underset{rm}{\Longrightarrow} -(id+id) \quad [E \to id]$$
$$E \overset{*}{\underset{rm}{\Longrightarrow}} -(id+id)$$

PARSE TREE

EXAMPLE -3

Find LMD, RMD and parse tree of $S \to aB / bA$, $A \to a / aS / bAA$, $B \to b / bS / aBB$ for the string aaabbabbba.

LMD

$$S \underset{lm}{\Longrightarrow} a\underline{B} \quad [S \to aB]$$
$$\underset{lm}{\Longrightarrow} aa\underline{B}B \quad [B \to aBB]$$
$$\underset{lm}{\Longrightarrow} aaa\underline{B}BB \quad [B \to aBB]$$
$$\underset{lm}{\Longrightarrow} aaab\underline{B}B \quad [B \to b]$$
$$\underset{lm}{\Longrightarrow} aaabb\underline{B} \quad [B \to b]$$
$$\underset{lm}{\Longrightarrow} aaabbaBB \quad [B \to aBB]$$

$\underset{lm}{\Rightarrow}$ aaabbabB [B → b]

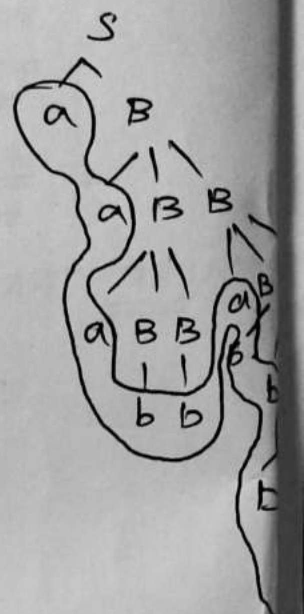
$\underset{lm}{\Rightarrow}$ aaabbabbS [B → bS]

$\underset{lm}{\Rightarrow}$ aaabbabbbA [S → bA]

$\underset{lm}{\Rightarrow}$ aaabbabbba [A → a]

S $\overset{*}{\underset{lm}{\Rightarrow}}$ aaabbabba

PARSE TREE



RMD:

S $\underset{rm}{\Rightarrow}$ aB [S → aB]

$\underset{rm}{\Rightarrow}$ aaBB [S → aBB]

$\underset{rm}{\Rightarrow}$ aaBaBB [S → aBB]

$\underset{rm}{\Rightarrow}$ aaBaBbS [B → bS]

$\underset{rm}{\Rightarrow}$ aaBaBbbA [S → bA]

$\underset{rm}{\Rightarrow}$ aaBaBbba [A → a]

$\underset{rm}{\Rightarrow}$ aaaBBaBbba [B → aBB]

$\underset{rm}{\Rightarrow}$ aaabbabbba [B → b]

S $\overset{*}{\underset{rm}{\Rightarrow}}$ aaabbabbba .

EXAMPLE-4.

Find LMD, RMD & parse tree of S → A/B, A → 0A/ε,
B → 0B/1B/ε

(i) 1001.

LMD

S $\underset{lm}{\Rightarrow}$ B [S → B]

$\underset{lm}{\Rightarrow}$ 1B [B → 1B]

$\underset{lm}{\Rightarrow}$ 10B [B → 0B]

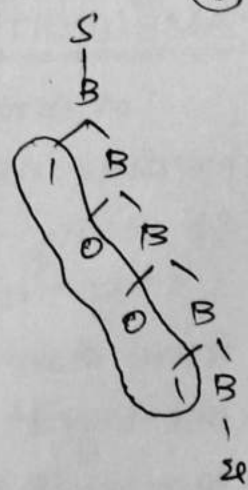
$\underset{lm}{\Rightarrow}$ 100B [B → 0B]

$\underset{lm}{\Rightarrow}$ 1001B [B → 1B]

S $\overset{*}{\underset{lm}{\Rightarrow}}$ 1001 [B → ε]

(6)

$S \xRightarrow[lm]{*} 1001$



RMD.

$S \xRightarrow[rm]{} \underline{B}$ [S → B]

$\xRightarrow[rm]{} 1\underline{B}$ [S → 1B]

$\xRightarrow[rm]{} 10\underline{B}$ [S → 0B]

$\xRightarrow[rm]{} 100\underline{B}$ [B → 0B]

$\xRightarrow[rm]{} 1001\underline{B}$ [B → 1B]

$\xRightarrow[rm]{} 1001$ [B → ε]

$S \xRightarrow[rm]{*} 1001.$

ii) 00011

LMD.

$S \xRightarrow[lm]{} \underline{B}$ [S → B]

$\xRightarrow[lm]{} 0\underline{B}$ [B → 0B]

$\xRightarrow[lm]{} 00\underline{B}$ [B → 0B]

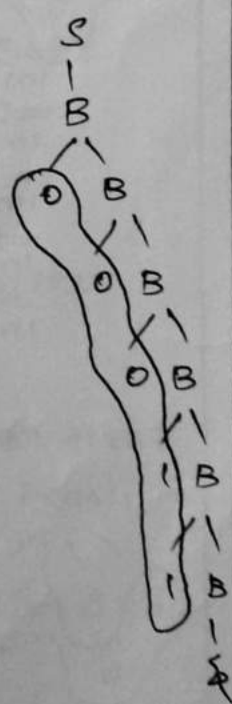
$\xRightarrow[lm]{} 000\underline{B}$ [B → 0B]

$\xRightarrow[lm]{} 0001\underline{B}$ [B → 1B]

$\xRightarrow[lm]{} 00011\underline{B}$ [B → 1B]

$\xRightarrow[lm]{} 00011$ [B → ε]

$S \xRightarrow[lm]{*} 00011$



RMD.

$S \xRightarrow[rm]{} B$ [S → B]

$\xRightarrow[rm]{} 0B$ [B → 0B]

$\xRightarrow[rm]{} 00B$ [B → 0B]

$\xRightarrow[rm]{} 000B$ [B → 0B]

$\xRightarrow[rm]{} 0001B$ [B → 1B]

$\xRightarrow[rm]{} 00011B$ [B → 1B]

$S \xRightarrow[rm]{*} 00011$

AMBIGUITY IN GENERATING LANGUAGE :-

A Grammar is said to be ambiguous if it produce more than one derivation tree for some i/p string generated by it. (OR)

If there are two distinct derivation (i.e) more than one left most derivation (or) more than one right most derivation for the given string then such a grammar is said to be ambiguous.

PROBLEM

1) Show that the following grammar is ambiguous

$S \to SbS / a$.

soln:-

LMD - 1

$S \xRightarrow[lm]{} SbS \quad [S \to SbS]$

$\xRightarrow[lm]{} abS \quad [S \to a]$

$\xRightarrow[lm]{} abSbS \quad [S \to SbS]$

$\xRightarrow[lm]{} ababS \quad [S \to a]$

$\xRightarrow[lm]{} ababa \quad [S \to a]$

LMD - 2

$S \xRightarrow[lm]{} SbS \quad [S \to SbS]$

$\xRightarrow[lm]{} SbSbS \quad [S \to SbS]$

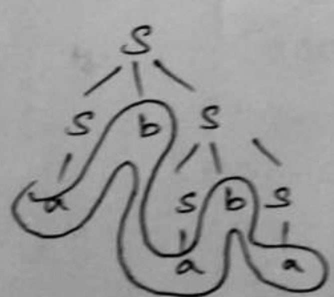
$\xRightarrow[lm]{} abSbS \quad [S \to a]$

$\xRightarrow[lm]{} ababS \quad [S \to a]$

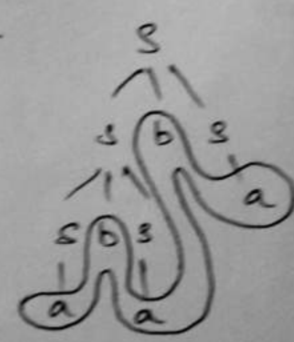
$\xRightarrow[lm]{} ababa \quad [S \to a]$.

PARSE TREE

LMD - 1



LMD - 2



Thus the grammar is ambiguous.

2) $S \to AA$, $A \to AAA / Ab / a / bA$ check whether it is ambiguous

consider string abab

LMD-1

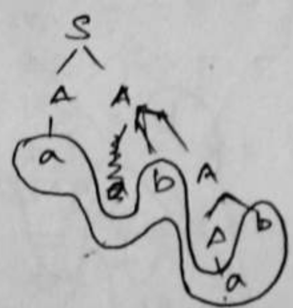
$$S \underset{lm}{\Rightarrow} AA \; [S \to AA]$$
$$\underset{lm}{\Rightarrow} aA \; [A \to a]$$
$$\underset{lm}{\Rightarrow} abA \; [A \to bA]$$
$$\underset{lm}{\Rightarrow} abAb \; [A \to Ab]$$
$$\underset{lm}{\Rightarrow} abab \; [A \to a]$$
$$S \overset{*}{\underset{lm}{\Rightarrow}} abab$$

LMD-2

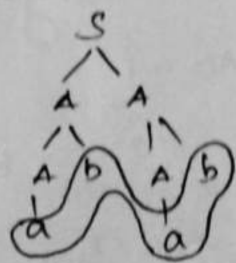
$$S \underset{lm}{\Rightarrow} AA \; [S \to AA]$$
$$\underset{lm}{\Rightarrow} AbA \; [A \to Ab]$$
$$\underset{lm}{\Rightarrow} abA \; [A \to a]$$
$$\underset{lm}{\Rightarrow} abAb \; [A \to Ab]$$
$$\underset{lm}{\Rightarrow} abab \; [A \to a]$$
$$S \overset{*}{\underset{lm}{\Rightarrow}} abab$$

PARSE TREE

LMD-1



LMD-2



$\therefore$ The grammar is ambiguous.

3) $S \to a / absb / aAb$, $A \to bs / aAAb$. check whether it is ambiguous.

consider string abab

LMD-1

$$S \underset{lm}{\Rightarrow} absb \; [S \to absb]$$
$$\underset{lm}{\Rightarrow} abab \; [S \to a]$$
$$S \overset{*}{\underset{lm}{\Rightarrow}} abab$$

PARSE TREE

LMD-2

$S \underset{lm}{\Rightarrow} aAb \; [S \rightarrow aAb]$

$\underset{lm}{\Rightarrow} absb \; [A \rightarrow bS]$

$\underset{lm}{\Rightarrow} abab \; [S \rightarrow a]$

$S \overset{*}{\underset{lm}{\Rightarrow}} abab$

∴ The grammar is ambiguous.

4) $S \rightarrow a / Sa / bSS / SSb / Sbs$. check whether it is ambiguous. consider a baaabaa.

LMD-1

$S \underset{lm}{\Rightarrow} bSS \; [S \rightarrow bSS]$

$\underset{lm}{\Rightarrow} bSbSS \; [S \rightarrow Sbs]$

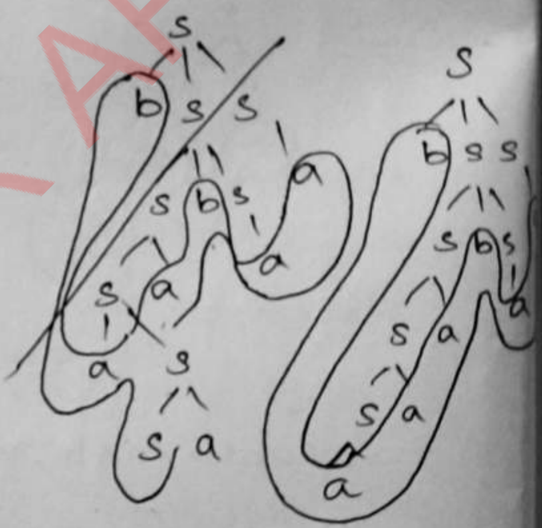
$\underset{lm}{\Rightarrow} bSabSS \; [S \rightarrow Sa]$

$\underset{lm}{\Rightarrow} bSaabSS \; [S \rightarrow Sa]$

$\underset{lm}{\Rightarrow} baaabSS \; [S \rightarrow a]$

$\underset{lm}{\Rightarrow} baaabaS \; [S \rightarrow a]$

$S \overset{*}{\underset{lm}{\Rightarrow}} baaabaa$



LMD-2

$S \underset{lm}{\Rightarrow} bSS \; [S \rightarrow bSS]$

$\underset{lm}{\Rightarrow} bSaS \; [S \rightarrow Sa]$

$\underset{lm}{\Rightarrow} bSaaS \; [S \rightarrow Sa]$

$\underset{lm}{\Rightarrow} baaaS \; [S \rightarrow Sa]$

$\underset{lm}{\Rightarrow} baaabSS \; [S \rightarrow bSS]$

$\underset{lm}{\Rightarrow}$ baaabaS [S → a]

$\underset{lm}{\Rightarrow}$ baaabaa [S → a]

$S \overset{*}{\underset{lm}{\Rightarrow}}$ baaabaa [S → a]

∴ The grammar is ambiguous.

5) E → E+T / T, T → T*F / F, F → (E) / id for string w = id*(id+id)*id. Find RMD, LMD, Parse tree and check whether it is ambiguous or not.

LMD:

$E \underset{lm}{\Rightarrow}$ T [E → T]

$\underset{lm}{\Rightarrow}$ T*F [T → T*F]

$\underset{lm}{\Rightarrow}$ T*F*F [T → T*F]

$\underset{lm}{\Rightarrow}$ F*F*F [T → F]

$\underset{lm}{\Rightarrow}$ id*F*F [F → id]

$\underset{lm}{\Rightarrow}$ id*(E)*F [F → (E)]

$\underset{lm}{\Rightarrow}$ id*(E+T)*F [E → E+T]

$\underset{lm}{\Rightarrow}$ id*(T+T)*F [E → T]

$\underset{lm}{\Rightarrow}$ id*(F+T)*F [T → F]

$\underset{lm}{\Rightarrow}$ id*(id+T)*F [F → id]

$\underset{lm}{\Rightarrow}$ id*(id+F)*F [T → F]

$\underset{lm}{\Rightarrow}$ id*(id+id)*F [F → id]

$\underset{lm}{\Rightarrow}$ id*(id+id)*id [F → id]

$S \overset{*}{\underset{lm}{\Rightarrow}}$ id*(id+id)*id.

RMD

$E \underset{rm}{\Rightarrow} T$ [$E \to T$]

$\underset{rm}{\Rightarrow} T*F$ [$T \to T*F$]

$\underset{rm}{\Rightarrow} T*id$ [$F \to id$]

$\underset{rm}{\Rightarrow} T*F*id$ [$T \to T*F$]

$\underset{rm}{\Rightarrow} T*(E)*id$ [$F \to (E)$]

$\underset{rm}{\Rightarrow} T*(E+T)*id$ [$E \to E+T$]

$\underset{rm}{\Rightarrow} T*(E+F)*id$ [$T \to F$]

$\underset{rm}{\Rightarrow} T*(E+id)*id$ [$F \to id$]

$\underset{rm}{\Rightarrow} T*(T+id)*id$ [$E \to T$]

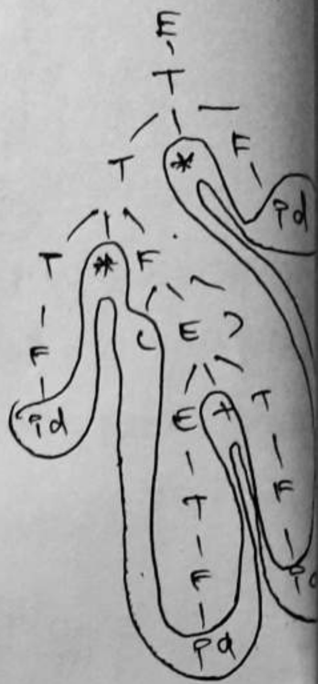
$\underset{rm}{\Rightarrow} T*(F+id)*id$ [$T \to F$]

$\underset{rm}{\Rightarrow} T*(id+id)*id$ [$F \to id$]

$\underset{rm}{\Rightarrow} F*(id+id)*id$ [$T \to F$]

$E \underset{rm}{\Rightarrow} id*(id+id)+id$ [$F \to id$]

$E \underset{rm}{\overset{*}{\Rightarrow}} id*(id+id)+id$



AMBIGUOUS TREE

consider: id+id

LMD1: $E \underset{lm}{\Rightarrow} E+T$ [$E \to E+T$]

$\underset{lm}{\Rightarrow} T+T$ [$E \to T$]

$\underset{lm}{\Rightarrow} F+T$ [$T \to F$]

$\underset{lm}{\Rightarrow} id+T$ [$F \to id$]

$\underset{lm}{\Rightarrow} id+F$ [$T \to F$]

$\underset{lm}{\Rightarrow} id+id$

LMD2: $E \underset{lm}{\Rightarrow} T$ [$E \to T$]

$\underset{lm}{\Rightarrow} F$ [$T \to F$]

$\underset{lm}{\Rightarrow} id$ [$F \to id$]

LMD1 ≠ LMD2

∴ The grammar is not ambiguous

(i) CFL into CFG

(ii) CFG into CFL

CONTEXT FREE LANGUAGE (CFL):-

The language generated by CFG is called as context free language

$L(G) = \{ w / w \in T^*$ and it can be derived from start symbol $S \}$

CFG into CFL:-

1) Find CFL for the given grammar $S \to aSb / ab$

soln:-

Grammar $G: S \to aSb / ab$

i) $S \underset{lm}{\Rightarrow} ab \quad [S \to ab]$

ii) $S \underset{lm}{\Rightarrow} a\underline{S}b \quad [S \to aSb]$

$\underset{lm}{\Rightarrow} aabb \quad [S \to ab]$

iii) $S \underset{lm}{\Rightarrow} a\underline{S}b \quad [S \to aSb]$

$\underset{lm}{\Rightarrow} aa\underline{S}bb \quad [S \to aSb]$

$\underset{lm}{\Rightarrow} aaabbb \quad [S \to ab]$

$\therefore L = \{ ab, aabb, aaabbb \ldots \}$

$L(G) = \{ a^n b^n / n \geq 1 \}$

2) Find CFL for grammar $S \to aB / bA$, $A \to a / aS / bAA$, $B \to b / bS / aBB$

soln:-

i) $S \underset{lm}{\Rightarrow} a\underline{B}$ [$S \to aB$]

$\underset{lm}{\Rightarrow} ab$ [$B \to b$]

ii) $S \underset{lm}{\Rightarrow} b\underline{A}$ [$S \to bA$]

$\underset{lm}{\Rightarrow} ba$ [$A \to a$]

iii) $S \underset{lm}{\Rightarrow} a\underline{B}$ [$S \to aB$]

$\underset{lm}{\Rightarrow} ab\underline{S}$ [$B \to bS$]

$\underset{lm}{\Rightarrow} aba\underline{B}$ [$S \to aB$]

$\underset{lm}{\Rightarrow} abab$ [$B \to b$]

iv) $S \underset{lm}{\Rightarrow} b\underline{A}$ [$S \to bA$]

$\underset{lm}{\Rightarrow} ba\underline{S}$ [$A \to aS$]

$\underset{lm}{\Rightarrow} bab\underline{A}$ [$S \to bA$]

$\underset{lm}{\Rightarrow} babb\underline{A}A$ [$A \to bAA$]

$\underset{lm}{\Rightarrow} babba\underline{A}$ [$A \to a$]

$\underset{lm}{\Rightarrow} babbaa$ [$A \to a$]

$\therefore L = \{ab, ba, abab, babbaa \ldots\}$

$L(G) = \{w / w \in (a, b)^*$, where w is equal number of a's and equal number of b's $\}$.

3) Find CFL for grammar for $S \rightarrow aSa / bSb / \epsilon$. (10)

Soln:-

$G: S \rightarrow aSa / bSb / \epsilon$

i) $S \underset{lm}{\Rightarrow} \epsilon \quad [S \rightarrow \epsilon]$

ii) $S \underset{lm}{\Rightarrow} a\underline{S}a \quad [S \rightarrow aSa]$

$\underset{lm}{\Rightarrow} aa \quad [S \rightarrow \epsilon]$

iii) $S \underset{lm}{\Rightarrow} b\underline{S}b \quad [S \rightarrow bSb]$

$\underset{lm}{\Rightarrow} bb \quad [S \rightarrow \epsilon]$

iv) $S \underset{lm}{\Rightarrow} a\underline{S}a \quad [S \rightarrow aSa]$

$\underset{lm}{\Rightarrow} ab\underline{S}ba \quad [S \rightarrow bSb]$

$\underset{lm}{\Rightarrow} abba \quad [S \rightarrow \epsilon]$

v) $S \underset{lm}{\Rightarrow} bSb \quad [S \rightarrow bSb]$

$\underset{lm}{\Rightarrow} ba\underline{S}ab \quad [S \rightarrow aSa]$

$\underset{lm}{\Rightarrow} baab \quad [S \rightarrow \epsilon]$

$L = \{\epsilon, aa, bb, abba, baab, abbbba \ldots\}$

$\therefore L(G) = \{ww^R / w \in (a, b)^*\}$

4) Find CFL for the given grammar $S \to aS/bS/a$

Soln:-

G: $S \to aS / bS / a$

i) $S \underset{lm}{\Rightarrow} a$ [$S \to a$]

ii) $S \underset{lm}{\Rightarrow} a\underline{S}$ [$S \to aS$]

$\underset{lm}{\Rightarrow} aa$ [$S \to a$]

iii) $S \underset{lm}{\Rightarrow} b\underline{S}$ [$S \to bS$]

$\underset{lm}{\Rightarrow} ba$ [$S \to a$]

iv) $S \underset{lm}{\Rightarrow} a\underline{S}$ [$S \to aS$]

$\underset{lm}{\Rightarrow} ab\underline{S}$ [$S \to bS$]

$\underset{lm}{\Rightarrow} aba$ [$S \to a$]

v) $S \underset{lm}{\Rightarrow} b\underline{S}$ [$S \to bS$]

$\underset{lm}{\Rightarrow} ba\underline{S}$ [$S \to aS$]

$\underset{lm}{\Rightarrow} baa$ [$S \to a$]

$L = \{a, aa, ba, aaba, baa \ldots\}$

$\therefore L(G) = \{w / w \in (a,b)^*$ where w ends with $a\}$

CONTEXT FREE LANGUAGE TO CONTEXT FREE GRAMMAR

(11)

1) Generate CFG for the language $L = \{a^n b^n / n \geq 0\}$

soln:-

$L = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$

$P: S \to aSb / \varepsilon$

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

$V = \{S\}$ $\quad T = \{a, b\}$

$P = \{S \to aSb, S \to \varepsilon\}$

$S = S$

$S \underset{lm}{\Rightarrow} aSb \quad [S \to aSb]$

$\underset{lm}{\Rightarrow} aaSbb \quad [S \to aSb]$

$\underset{lm}{\Rightarrow} aabb \quad [S \to \varepsilon]$

$\boxed{S \overset{*}{\underset{lm}{\Rightarrow}} aabb} \in L.$

2) $L = \{a^n b^n / n \geq 1\}$

soln:-

$L = \{ab, aabb, aaabbb \ldots\}$

$P: S \to aSb / ab$

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

$V = \{S\}$ $\quad P = \{S \to aSb, S \to ab\}$

$T = \{a, b\}$ $\quad S = S$

$S \underset{lm}{\Rightarrow} aSb \quad [S \to aSb]$

$\underset{lm}{\Rightarrow} aabb \quad [S \to ab]$

$\boxed{S \overset{*}{\underset{lm}{\Rightarrow}} aabb} \in L$

3) $L = \{a^n b a^n / n \geq 1\}$

soln:-

$L = \{aba, aabaa, aaabaaa, \ldots\}$

$P: S \to aSa / aba$

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

$V = \{S\}$ $\quad P = \{S \to aSa, S \to aba\}$

$T = \{a, b\}$ $\quad S = S$

$S \underset{lm}{\Rightarrow} aSa \quad [S \to aSa]$

$\underset{lm}{\Rightarrow} aabaa \quad [S \to aba]$

$\boxed{S \overset{*}{\underset{lm}{\Rightarrow}} aabaa} \in L$

4) Construct CFG representing the set of palindrome over $(0+1)^*$

soln:-

$L = \{\varepsilon, 0, 1, 00, 11, 101, 1001, \ldots\}$

$P: S \to 0S0 / 1S1$

$S \to \epsilon / 0 / 1$

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

$V = \{S\}$

$T = \{0, 1\}$

$P = \{S \to \epsilon, S \to 0, S \to 1, S \to 0S0, S \to 1S1\}$

$S = \{S\}$

$S \xrightarrow{} 0S0 \underset{lm}{\Rightarrow} 01S10 \underset{lm}{\Rightarrow} \underline{0110} \in L$

---

5) construct a CFG for a set of string that contain equal no of a's and b's

soln:-

$L = \{\epsilon, ab, ba, aabb, aaabbb, \dots\}$

$P: S \to \epsilon / ab / ba / aSb / bSa$

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

$V = \{S\}$ $T = \{a, b\}$

$P = \{S \to \epsilon, S \to ab, S \to ba, S \to aSb, S \to bSa\}$

$S = \{S\}$

---

6) construct a CFG for a regular expression $(011 + 1)(01)$

soln:-

$S \to AB$

$A \to 011 / 1$

$B \to 01$

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

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

$P = \{S \to AB, A \to 011, A \to 1, B \to 01\}$

$S = \{S\}$

PUSH-DOWN AUTOMATA :

→ The context free language is defined by the special type of automata namely Push Down Automata.

→ Push Down Automata is an extension of NFA with ϵ-transitions with the addition of stack.

→ Stack (Last in First Out) is used to store the string of stack symbols and read the symbols, push and pop only at the top of stack.

→ Push-down automata can remember an infinite amount of string information.

→ Push-down can access the information on its stack in LIFO way.

→ Push down automata can recognize only CFL.

→ The PDA is more powerful than finite automata which has finite memory.

DEFINITION OF PUSH-DOWN AUTOMATA :

PDA involves seven tuples or components.

$$\boxed{\text{PDA } P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)}$$

Where,

Q → Finite non-empty set of states.

Σ → Finite set of input symbols.

Γ → Finite stack alphabet [This is the set of symbols that are pushed to the stack]

δ → Transition function.

$\delta : Q \times (\Sigma \cup \epsilon) \times \Gamma \longrightarrow Q \times \Gamma^*$ defined as

$\delta(q, a, x) = (q', ax')$

where, q, q' → states in Q

a → Input symbol in Σ or ϵ

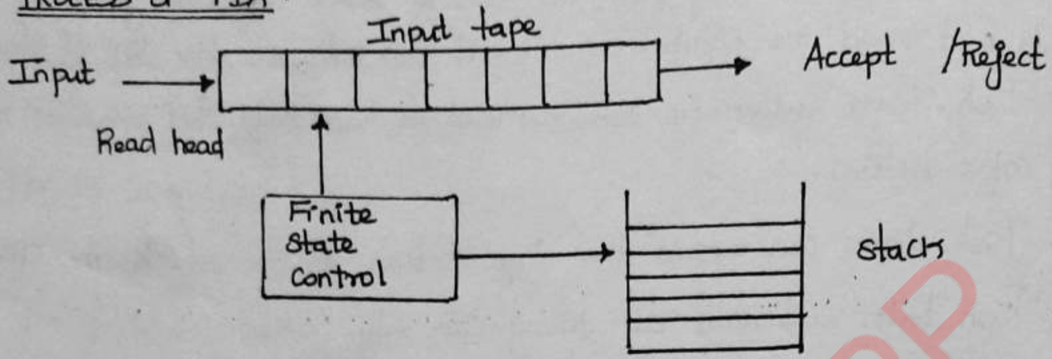
$x, x' \rightarrow$ stack symbol in Γ

$q_0 \longrightarrow$ initial / start symbol ($q_0 \in Q$)

$Z_0 \longrightarrow$ initial / start / Bottom of stack symbol ($Z_0 \in \Gamma$)

$F \longrightarrow$ Final set of accepting states / Final state ($F \subseteq Q$)

PROCESS OF PDA :



The PDA Consists of the following,

→ Finite state control unit

→ Input tape

→ Stack

→ Reading Head.

- The input string is present in input tape.
- The finite state control read the input, one symbol at a time from the input tape.
- The Finite state control is the NFA with ϵ-transition.
- After reading the input symbol, it recognizes the current state of the control and the symbol present at top of the stack.
- So, the Finite control consists of set of transitions, input state and set of Final states.
- So the transition of PDA depends on the current state, input symbol from the input tape and the symbol at top of stack.

- The PDA can also make a <u>spontaneous transition</u> using ϵ its input instead of an input symbol.
- The activities done by PDA is as follows,

1. Read the input symbol from input tape. If ϵ is the input, then no input symbol is consumed.

2. It makes the transition with the current state, input symbol, symbol at the top of stack. So after transition, the control may enter in either a new state or previous state.

3. Replace the symbol at top of the stack by any other symbol. The string could be also ϵ.

i) If <u>ϵ is used</u> to replace the stack symbol, it corresponds to a <u>pop of the stack</u>

ii) If the same symbol is again used to replace, then no change to the stack is made.

iii) The finite state control can <u>push one or more symbols</u> to the stack.

<u>The two different versions of PDA is as follows,</u>

1. One that accept the string by entering an accepting state.
2. Accepts by emptying the stack.

<u>TYPES OF PDA:</u>

i) Non Deterministic PDA [NPDA]

2) Deterministic PDA [DPDA]

Moves / Transitions of PDA :

A transition is given by,

$$Q \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow Q \times \Gamma^{*} \quad [\text{DPDA}]$$

$$Q \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow 2^{(Q \times \Gamma^{*})} \quad (\text{NPDA})$$

$Q \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow$ implies that a transition based on

- * Current state $q \in Q$
- * Next input $\Sigma \cup \epsilon$
- * Stack symbol [Top most element of stack

$(Q \times \Gamma^{*}) \rightarrow$ (i) implies that next state reached after transition.

(ii) $q' \rightarrow$ Current / Next state defined in Q, $q' \in Q$

(iii) stack symbol remaining after Γ^{*}.

Operations on Stack :

1) **Push** :

$$\boxed{\delta(q_1, a, b) = (q_2, ab)}$$

Next state

Top of stack

Input symbol

Current state.

$ab \Rightarrow$ Here the top most symbols on stack is 'b' is replaced by 'a' followed by 'b'. Since the input 'a' is pushed / inserted on to the stack.

2) **Pop** :

$$\boxed{\delta(q_1, a, b) = (q_2, \epsilon)}$$

Next state

Top of stack

Input Symbol

Current state.

$\epsilon \Rightarrow$ Here 'a' cancels 'b' from the stack. The above transition cancels the top most stack symbols by the input processing.

3) READ INPUT WITH NO OPERATION ON STACK :

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

No operation is performed on stack, the stack symbols are neither added up/ deleted off.

THE LANGUAGE OF PDA :

It can be accepted by two ways :

1. Acceptance by Final state.
2. Acceptance by Empty stack.

* These two methods are equivalent that the language is accepted by PDA by final state if only if the language is accepted by empty stack.
* However the PDA, P generated for a language is different that is accepted by final state is different with that of PDA generated by empty stack.

ACCEPTANCE BY FINAL STATE:

Let $\boxed{P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)}$ be a PDA.

Then $L(P)$, the language accepted by P by final state is,

$$L(P) = \{ w \mid (q_0, w, Z_0) \overset{*}{\underset{P}{\vdash}} (q_F, \epsilon, \alpha) \}$$

for some stack q_F in F and any stack string α, ie., starting in the initial Instantaneous Descriptor with w waiting on the input, P consumes w from the input and enters an accepting state. $\therefore q_F \in F, \alpha \in \Gamma^*$

INSTANTANEOUS DESCRIPTION OF PDA (ID):-

* The exeution status of the PDA is represented by the ID of PDA.
* It is a pictorial / diagrammatic representation of a string processed by a PDA.
* The ID records the state, stack contents and the input symbol. The ID is defined as 3 tuples (q, a, γ) where,

$q \rightarrow$ state of PDA

$a \rightarrow$ Remaining input

$\gamma \rightarrow$ Stack contents.

If a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ has the transition $\delta(q, a, x) = (p, \alpha)$ then for all the strings w in Σ^* and β in γ^*, the ID is given by

$$\boxed{(q_1, aw, x\beta) \overset{*}{\underset{P}{\vdash}} (q_2, w, \alpha\beta)}$$

This means that by reading the input symbol 'a' at the state 'q' with x as top stack symbols replaces α for x and reaches the state 'p'.

REPRESENTATION OF PDA:

1. Transition Diagram.
2. Transition Function / Moves

Acceptance by empty stack:

A PDA defined by $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ accept a given language by generating the set of strings $w \in \Sigma^*$ by making its stack empty such that,

$$L(P) = \{ w / (q_0, w, z_0) = (q, \varepsilon, \varepsilon) \}$$

CONSTRUCTION OF PDA / DESIGN OF PDA.

PROBLEMS

1. Design the PDA to accept the language $L = \{a^n b^n / n \geq 1\}$ accepting by final state / Empty stack (or) $L = \{0^n 1^n / n \geq 1\}$

Solution:

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

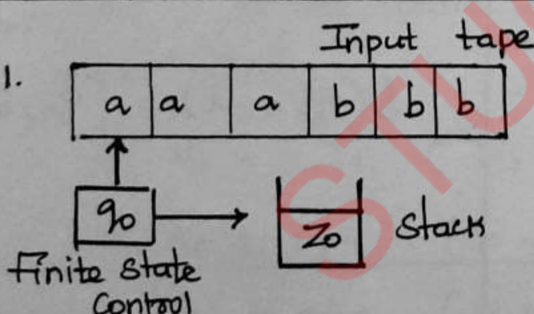
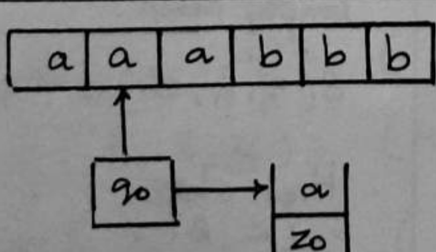
$$L = \{ab, aabb, aaabbb, \cdots\}$$

Algorithm:

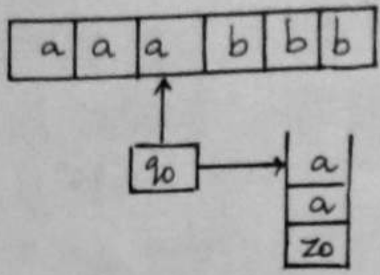
Consider $n=3$, $w = aaabbb$

1. Push 'n' number of 'a's into the stack.
2. For every 'b' pop out an 'a' from the stack.
3. At the end of the string, the machine stops as it reaches the final state.

EXAMPLE: $n=3$ $w = aaabbb$

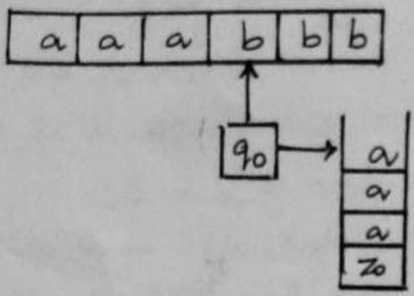
Graphical representation	Transition moves
1. 	input $\delta(q_0, a, Z_0) = (q_0, aZ_0)$ state top of Stack at initial Push top of stack
2. 	$\delta(q_0, a, a) = (q_0, aa)$ top of stack

3.



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

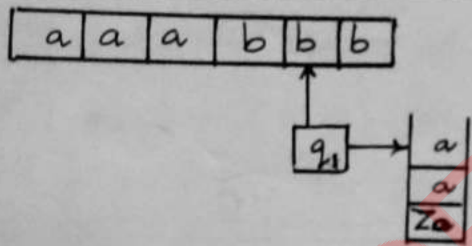
4.



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

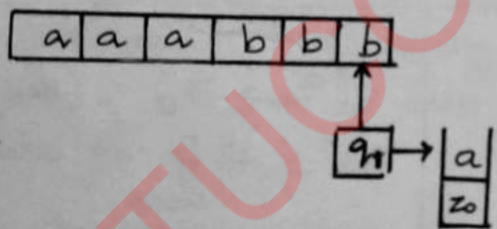
Current state ↓ input ↓ top of stack ↓ Next state ↓ Pop

5.



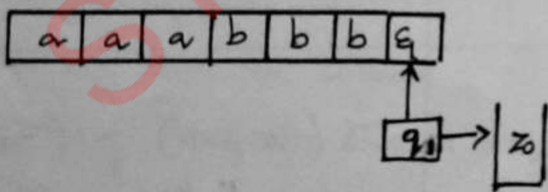
$\delta(q_1, b, a) = (q_1, \epsilon)$

6.



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

7.



Empty stack:

$\delta(q_1, \epsilon, Z_0) = (q_2, \epsilon)$

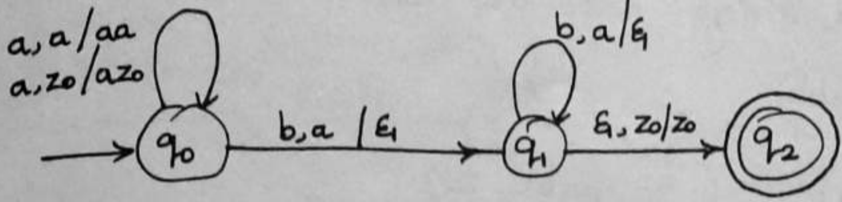
Final state:

$\delta(q_1, \epsilon, Z_0) = (q_2, Z_0)$

Transition diagram:

$a^n \; b^n$
push pop
| └ $b, a/\epsilon$
$a, z_0/az_0$
$a, a/aa$

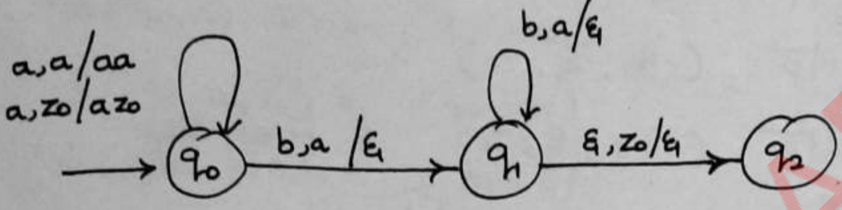
FINAL STATE:



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

PDA $P_F = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \{q_2\})$

EMPTY STACK:



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

PDA $P_N = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \phi)$

Transition Function / Moves defined as:

Let q_0 be the initial state and z_0 be the top symbol of the stack initially.

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

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

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

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

$\delta(q_1, \epsilon, z_0) = (q_2, z_0)$ // Final state

(or) $\delta(q_1, \epsilon, z_0) = (q_2, \epsilon)$ // Empty state.

INSTANTANEOUS DESCRIPTION:

Let us consider strings aabb and aab.

Input 1: $w = aabb$

$$
\begin{aligned}
(q_0, w, z_0) &\vdash_P (q_0, aabb, z_0) \\
&\vdash_P (q_0, abb, az_0) \\
&\vdash_P (q_0, bb, aaz_0) \\
&\vdash_P (q_1, b, az_0) \\
&\vdash_P (q_1, \epsilon, z_0) \\
&\vdash_P (q_2, \epsilon, z_0) \quad // \text{ Final state.}
\end{aligned}
$$

So, the string "aabb" is accepted.

Input 2: $w = aab$

$$
\begin{aligned}
(q_0, w, z_0) &\vdash_P (q_0, aab, z_0) \\
&\vdash_P (q_0, ab, az_0) \\
&\vdash_P (q_0, b, aaz_0) \\
&\vdash_P (q_1, \epsilon, az_0)
\end{aligned}
$$

Hence there is no transition for (q_1, ϵ, az_0) in above PDA.

So the string aab is not accepted.

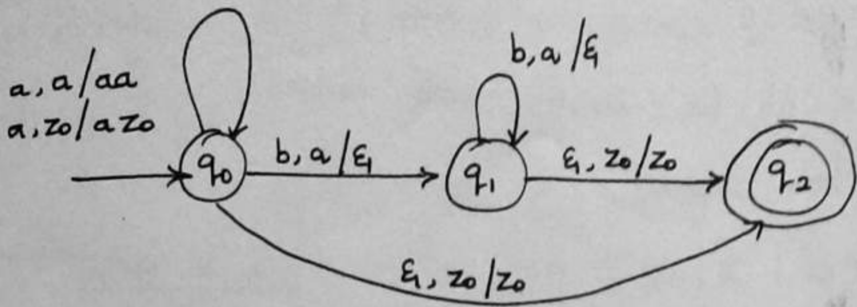
---

2) $L = \{a^n b^n / n \geq 0\}$ accept by final state.

Step 1: $L = \{a^n b^n / n \geq 0\}$

$\therefore L = \{\epsilon, ab, aabb, aaabbb, \ldots\}$

step 2 : Transition Diagram



step 3 : Transition function / Moves.

$\delta(q_0, \varepsilon, z_0) = (q_2, z_0)$

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

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

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

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

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

PDA $P_F = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \{q_2\})$

step 4 : Instantaneous description.

$w = aabb$

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

$\vdash_P (q_0, bb, aaz_0)$

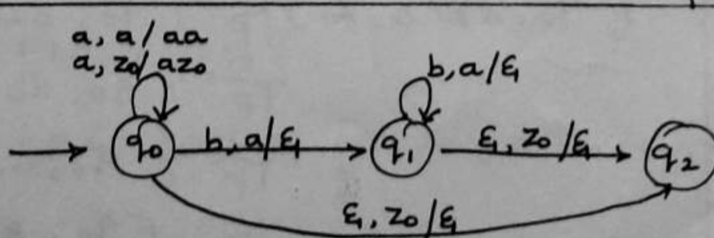
$\vdash_P (q_1, b, az_0)$

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

$\vdash_P (q_2, \varepsilon, z_0)$ ∴ $q_2 \in F$. So the string is accepted.

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

accept by empty stack :

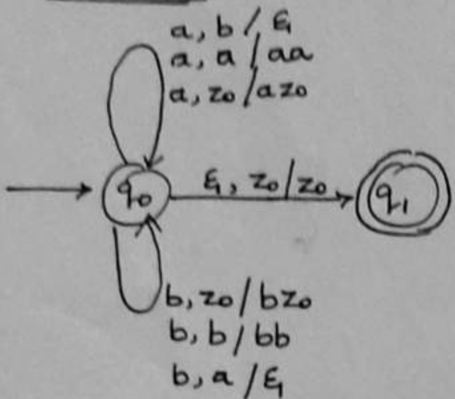


3). $L = \{$equal No. of a's and b's$\}$ over $\{a, b\}$ accepted by final state / Empty stack

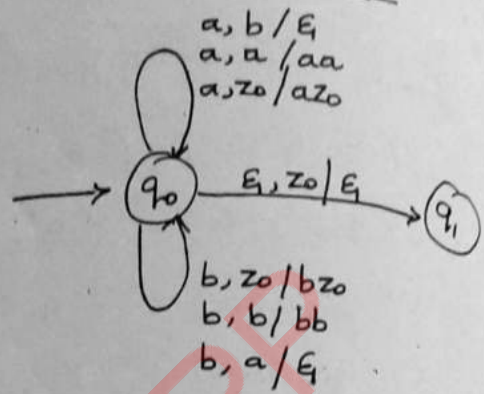
step 1: $L = \{\epsilon, ab, ba, aabb, abab, abba, \dots\}$

step 2: Transition diagram.

Final state.



Empty stack.



Step 3: Transition Function

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

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

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

$\delta(q_0, b, z_0) = (q_0, bz_0)$

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

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

$\delta(q_0, \epsilon, z_0) = (q_1, z_0)$ // Final state

$\delta(q_0, \epsilon, z_0) = (q_1, \epsilon)$ // Empty stack.

PDA definition:

Final state:

PDA $P_F = (\{q_0, q_1\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \{q_1\})$

empty stack:

PDA $P = (\{q_0, q_1\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \phi)$.

Step 4: Instantaneous Description for final state.

Input: $w_1 = abab$

$(q_0, abab, z_0) \vdash_P (q_0, bab, az_0)$

$\vdash_P (q_0, ab, z_0)$

$\vdash_P (q_0, b, az_0)$

$\vdash_P (q_0, \epsilon, z_0)$

$\vdash_P (q_1, \epsilon, z_0)$ // String is accepted for final state

w_2 - abb

$(q_0, abb, z_0) \vdash_P (q_0, bb, az_0)$

$\vdash_P (q_0, b, z_0)$

$\vdash_P (q_0, \epsilon, bz_0)$

There is no transition for (q_0, ϵ, bz_0)

$\therefore$ The string is not accepted.

$L = \{ 0^n 1^{2n} / n \geq 0 \}$

4) $L = \{ a^n b^{2n} / n \geq 0 \}$ (or) Design a PDA with set of strings with twice as many b's than a's with a as the starting string (or) 2 occurences of b's for each a's. accept by final state

Step1 : $L = \{ \epsilon, abb, aabbbb, \ldots \}$

Idea: To Design this PDA, is that when we read single 'a' we insert / push 2 a's on stack.

Then when we read 'b' we pop each 'a' on the top of stack and when reading z_0 on stack, we reach final state.

Step 3: Transition function:

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

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

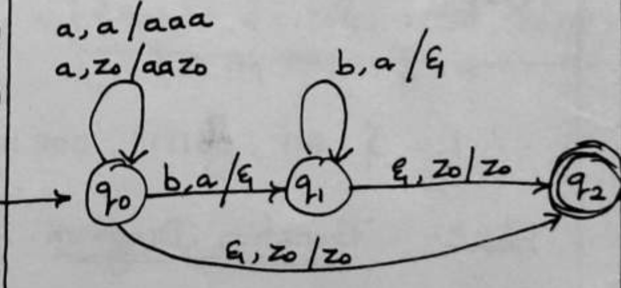
$\delta(q_0, b, a) = (q_1, \epsilon)$

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

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

$\delta(q_0, \epsilon, z_0) = (q_2, z_0)$

Step 2: Transition diagram.



PDA for $L = \{ a^n b^{2n} / n \geq 0 \}$

PDA $P_F = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \{q_2\})$

Step 4: Instantaneous description

w = aabbbb

$(q_0, aabbbb, z_0)$

$\vdash_P (q_0, abbbb, aaz_0)$

$\vdash_P (q_0, bbbb, aaaaz_0)$

$\vdash_P (q_1, bbb, aaaz_0)$

$\vdash_P (q_1, bb, aaz_0)$

$\vdash_P (q_1, b, az_0)$

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

$\vdash_P (q_2, \epsilon, z_0)$

$\therefore$ The string aabbbb is accepted.

w = aab

$(q_0, aab, z_0) \vdash_P (q_0, ab, aaz_0)$

$\vdash_P (q_0, b, aaaaz_0)$

$\vdash_P (q_1, \epsilon, aaaz_0)$

Here aab is not accepted as there is no transition for $\delta(q_1, \epsilon, a)$.

5) $L = \{0^n 1^{2n} / n \geq 1\}$ accept by final state/Empty stack

(or) $L = \{a^n b^{2n} / n \geq 1\}$

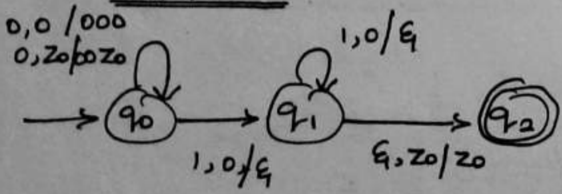
Step 1:

$L = \{0^n 1^{2n} / n \geq 1\}$

$\therefore L = \{011, 001111, 000111111, \ldots\}$

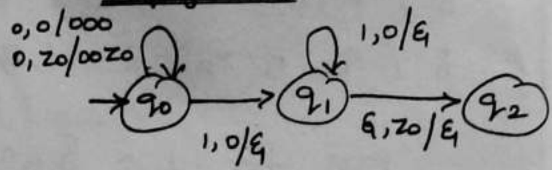
Step 2: Transition Diagram

Final state



PDA $P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, z_0\}, \delta, q_0, z_0, \{q_2\})$

empty stack



PDA $P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, z_0\}, \delta, q_0, z_0, \phi)$

step 3: Transition moves - Finalstate

$\delta(q_0, 0, z_0) = (q_0, 00z_0)$

$\delta(q_0, 0, 0) = (q_0, 000)$

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

$\delta(q_1, 1, 0) = (q_1, \epsilon)$

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

step 4: Instantaneous description

$w = 011$

$\delta(q_0, 011, z_0) \vdash_P (q_0, 11, 00z_0)$

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

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

$\vdash_P (q_2, \epsilon, z_0)$

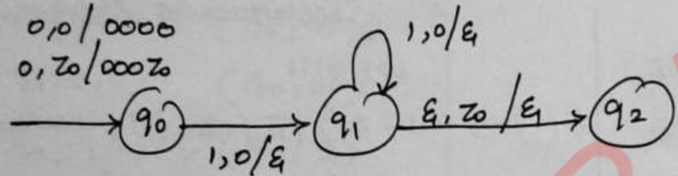
$\therefore$ String is accepted.

b) $L = \{0^n 1^{3n} \mid n \geq 1\}$ using final state or / empty stack.

(or) $L = \{a^n b^{3n} \mid n \geq 1\}$

step 1: $L = \{ 0111, 0011111, \ldots \}$

step 2:



step 3: Transition Moves. For empty stack.

$\delta(q_0, 0, z_0) = (q_0, 000z_0)$

$\delta(q_0, 0, 0) = (q_0, 0000)$

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

$\delta(q_1, 1, 0) = (q_1, \epsilon)$

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

PDA $P_N = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, z_0\}, \delta, q_0, z_0, \phi)$

step 4: Instantaneous description $w = 0111$

$(q_0, 0111, z_0) \vdash_P (q_0, 111, 000z_0)$

$\vdash_P (q_1, 11, 00z_0)$

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

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

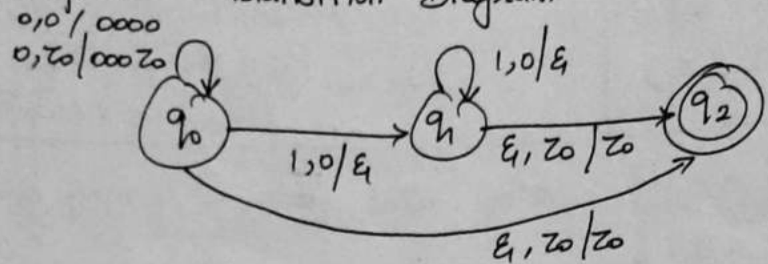
$\vdash_P (q_2, \epsilon, \epsilon)$

$\therefore$ The string is accepted

7) $L = \{0^n 1^{3n} / n \geq 0\}$ using final state

Step 1: $L = \{\varepsilon, 0111, 00111111, \ldots\}$

Step 2: Transition Diagram



PDA $P_F = (\{q_0, q_1, q_2\}, \{0,1\}, \{0, Z_0\}, \delta, q_0, Z_0, \{q_2\})$

3: Transition moves

$\delta(q_0, 0, Z_0) = (q_0, 000Z_0)$

$\delta(q_0, 0, 0) = (q_0, 0000)$

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

$\delta(q_1, 1, 0) = (q_1, \varepsilon)$

$\delta(q_1, \varepsilon, Z_0) = (q_2, Z_0)$

$\delta(q_0, \varepsilon, Z_0) = (q_2, Z_0)$

4. Instantaneous description

$w = 0111$

$\delta(q_0, 0111, Z_0) \vdash_P (q_0, 111, 000Z_0)$

$\vdash_P (q_1, 11, 00Z_0)$

$\vdash_P (q_1, 1, 0Z_0)$

$\vdash_P (q_1, \varepsilon, Z_0)$

$\vdash_P (q_2, \varepsilon, Z_0)$

$\therefore$ The string is accepted.

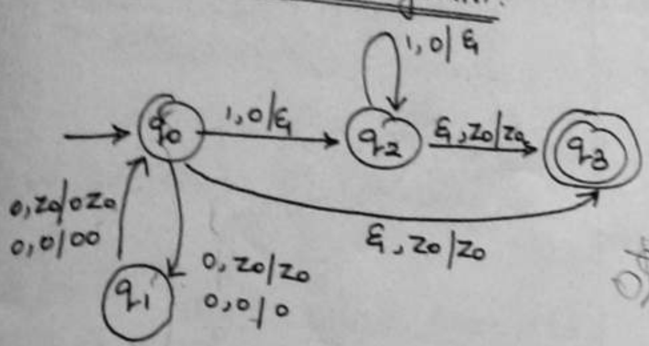
000011

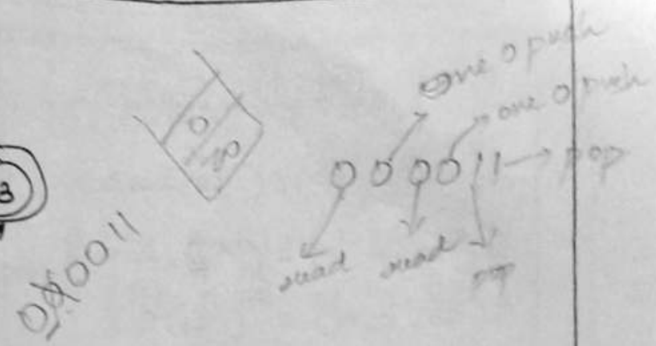
8) $L = \{0^{2n} 1^n / n \geq 0\}$ accept by final state.

Step 1: $L = \{\varepsilon, 001, 000011, \ldots\}$

Idea: When reading first '0' we goto some state and when we read the second '0' we push the single '0' to the stack and move to the initial state itself. Then while reading '1' we try to pop '0' and when reading empty string and Z_0 at the top stack symbol, we enter the final state.

step 2: Transition diagram.





PDA $P_F = (\{q_0, q_1, q_2, q_3\}, \{1, 0\}, \{0, Z_0\}, \delta, q_0, Z_0, \{q_3\})$

step 3: Transition function

$\delta(q_0, 0, Z_0) = (q_1, Z_0)$

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

$\delta(q_1, 0, Z_0) = (q_0, 0Z_0)$

$\delta(q_1, 0, 0) = (q_0, 00)$

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

$\delta(q_2, 1, 0) = (q_2, \epsilon)$

$\delta(q_2, \epsilon, Z_0) = (q_3, Z_0)$

$\delta(q_0, \epsilon, Z_0) = (q_3, Z_0)$

step 4: Instantaneous description

$w_1 = 001$

$(q_0, 001, Z_0) \vdash_P (q_1, 01, Z_0)$

$\vdash_P (q_0, 1, 0Z_0)$

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

$\vdash_P (q_3, \epsilon, Z_0)$

$\therefore$ The string is accepted.

$w_2 = 011$

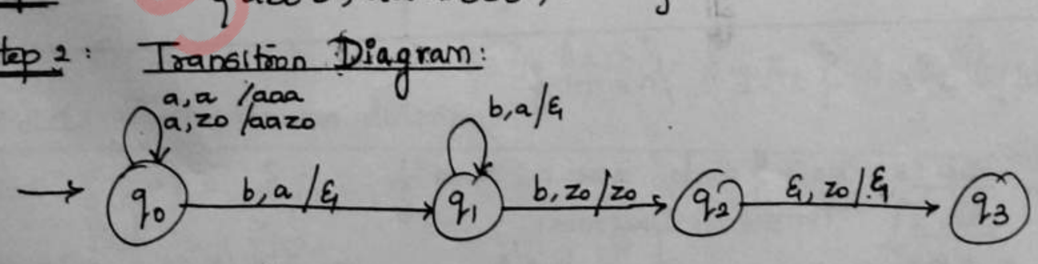
$(q_0, 011, Z_0) \vdash_P (q_1, 11, Z_0)$

Since there is no transition for $\delta(q_1, 1, Z_0)$ the string is not accepted.

9) $L = \{a^n b^{2n+1} / n \geq 1\}$ by empty stack.

Step 1: $L = \{abbb, aabbbbb, \dots\}$

Step 2: Transition Diagram:



PDA $P_N = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$

Step 3: Transition Function

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

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

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

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

$\delta(q_2, \epsilon, z_0) = (q_3, \epsilon)$.

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

Step 4: Instantaneous description:

$w_1 = abbb$

$(q_0, abbb, z_0) \vdash_P (q_0, bbb, aaz_0)$

$\vdash_P (q_1, bb, az_0)$

$\vdash_P (q_1, b, z_0)$

$\vdash_P (q_2, \epsilon, z_0)$

$\vdash_P (q_3, \epsilon, \epsilon)$

$\therefore$ string is accepted.

$w_2 = abb$

$(q_0, abb, z_0) \vdash_P (q_0, bb, aaz_0)$

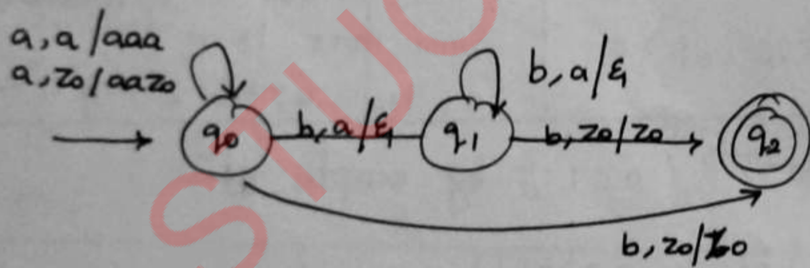
$\vdash_P (q_1, b, az_0)$

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

There is no transition $\therefore$ string is not accepted.

10) $L = \{a^n b^{2n+1} / n \geq 0\}$ accepted by Final state.

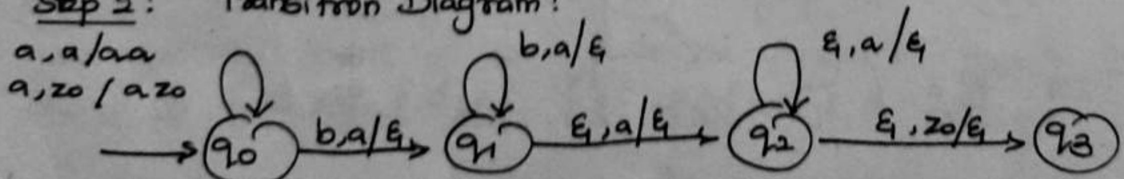
$L = \{b, abbb, aabbbbb, \ldots\}$



11) $L = \{a^n b^m / n > m\}$

Step 1: $L = \{aab, aaab, aaabb, \ldots\}$

Step 2: Transition Diagram:



PDA $P_N = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \phi)$

Step 3: Transition function.

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

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

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

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

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

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

$\delta(q_2, \epsilon, z_0) = (q_3, \epsilon)$

Step 4: Instantaneous description

$w_1 = aab$

$\delta(q_0, aab, z_0) \vdash_P (q_0, ab, az_0)$

$\vdash_P (q_0, b, aaz_0)$

$\vdash_P (q_1, \epsilon, az_0)$

$\vdash_P (q_2, \epsilon, z_0)$

$\vdash_P (q_3, \epsilon, \epsilon)$

$\therefore$ String is accepted.

$w_2 = ab$

$(q_0, ab, z_0) \vdash_P (q_0, b, az_0)$

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

There is no transition for

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

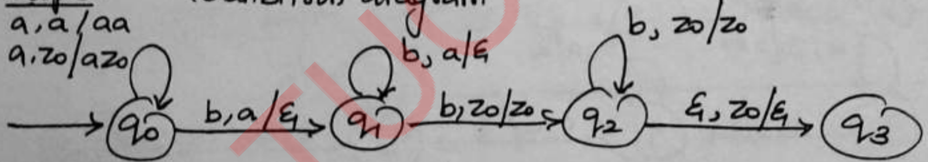
$\Rightarrow$ The string is not accepted.

12) $L = \{a^n b^m \mid n < m\}$

Step 1: $L = \{abb, aabbb, abbb, \cdots\}$

Step 2: Write transition function.

Step 3: Transition diagram



PDA for $L = \{a^n b^m \mid n < m\}$

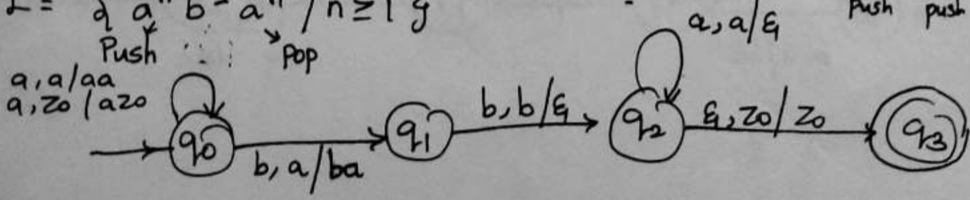
PDA $P_N = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \phi)$

Step 4: Instantaneous description

13) $L = \{a^n b^2 a^n \mid n \geq 1\}$

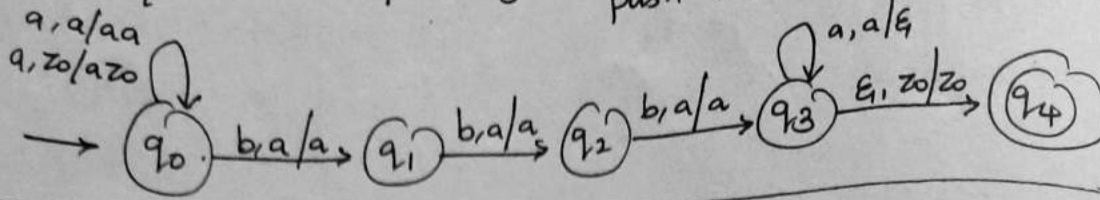
Push ... pop

a^n push, b push, b pop, a^n pop



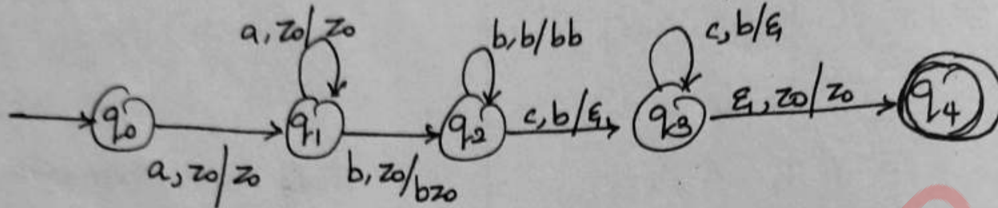
14) $L = \{a^n b^3 a^n \mid n \geq 1\}$ a^n b^3 a^n

push read pop



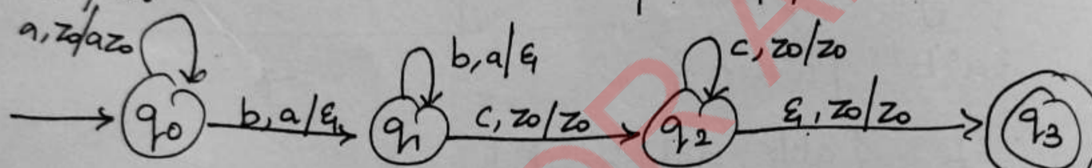
15) $L = \{a^n b^m c^m \mid n, m \geq 1\}$ a^n b^m c^m

read push pop



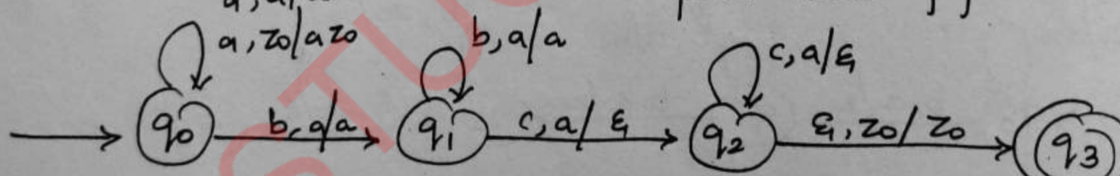
16) $L = \{a^n b^n c^m \mid n, m \geq 1\}$ a^n b^n c^m

push pop read



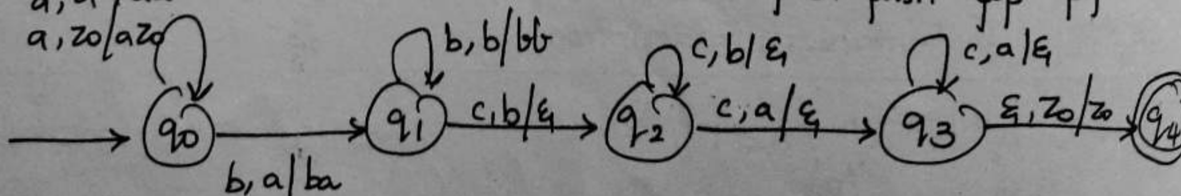
17) $L = \{a^n b^m c^n \mid n, m \geq 1\}$ a^n b^m c^n

push read pop

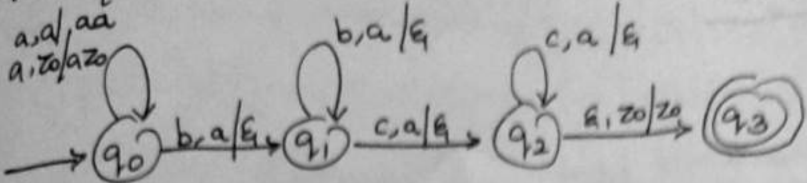


18) $L = \{a^n b^m c^{n+m} \mid n, m \geq 1\}$ a^n b^m c^n c^m

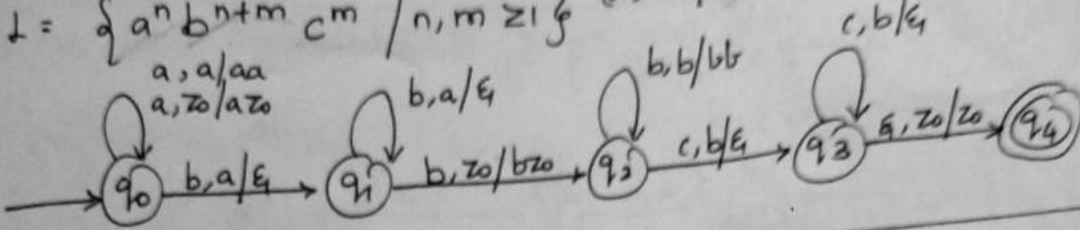
push push pop pop



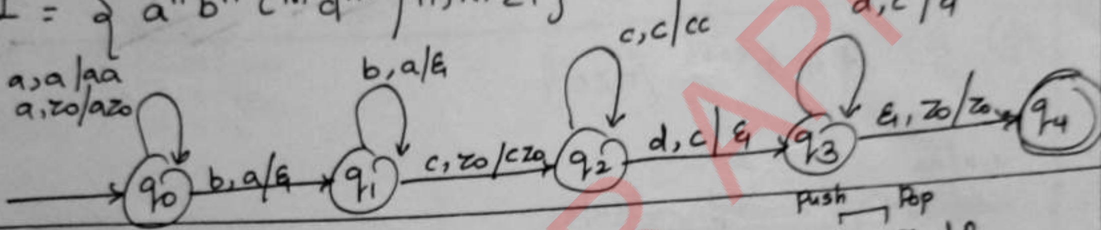
19) $L = \{ a^{n+m} b^m c^n \mid n, m \geq 1 \}$ (or) $\{ a^n a^m b^m c^n \}$



20) $L = \{ a^n b^{n+m} c^m \mid n, m \geq 1 \}$ (or) $\{ a^n b^n b^m c^m \}$



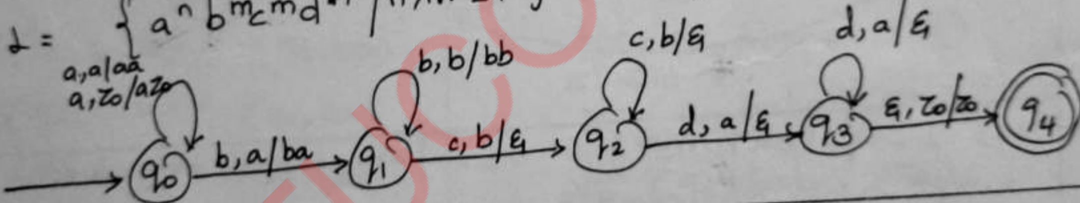
21) $L = \{ \underset{\text{push}}{a^n} \underset{\text{pop}}{b^n} \underset{\text{push}}{c^m} \underset{\text{pop}}{d^m} \mid n, m \geq 1 \}$



push a^n b^m c^m d^n pop

push pop

22) $L = \{ a^n b^m c^m d^n \mid n, m \geq 1 \}$



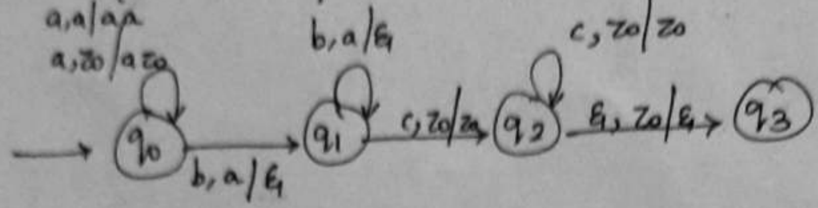
23) No PDA :-

$L = \{ \underset{\text{push}}{a^n} \underset{\text{push}}{b^m} \underset{\text{pop}}{c^n} \underset{\text{pop}}{d^m} \mid n, m \geq 1 \}$

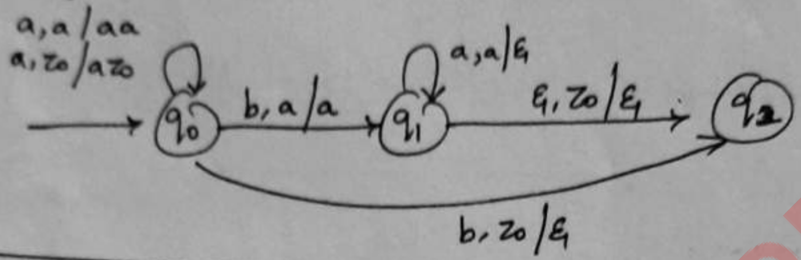
$L = \{ a^n b^n c^n \mid n \geq 1 \}$

push pop

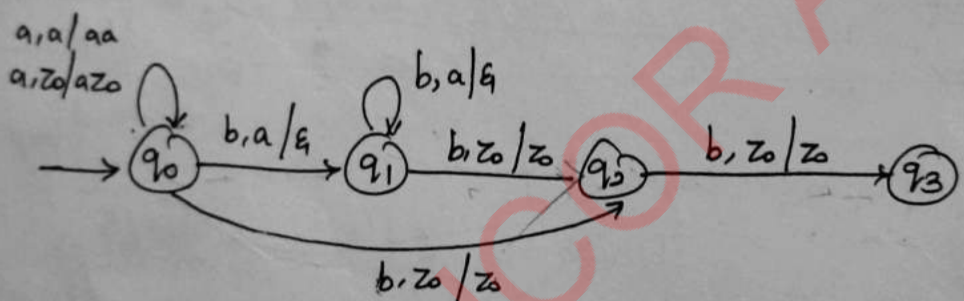
24) $L = \{a^m b^m c^n / n, m \geq 1\}$ by Empty stack



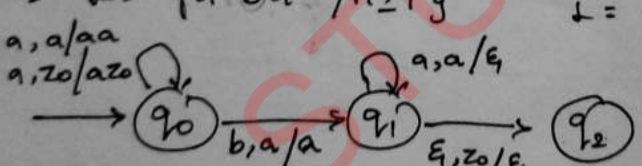
25) $L = \{a^n b a^n / n \geq 0\}$ by Empty stack



26) $L = \{a^n b^{2n+2} / n \geq 0\}$



27) $L = \{a^n b a^n / n \geq 1\}$ $L = \{aba, aabaa, aaabaaa, \ldots\}$

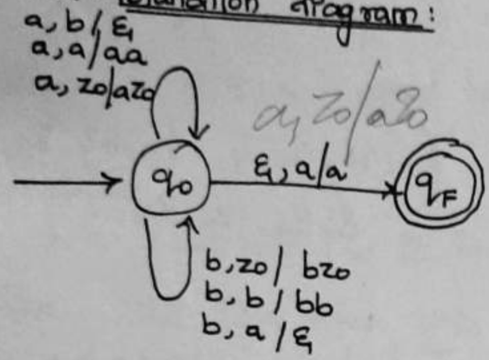


Design a PDA for the language

1. $L = \{ w / w \in (a+b)^* \text{ and } n_a(w) > n_b(w) \}$

$n_a(w)$ means total no. of a's in input string and $n_b(w)$ means total no. of b's in input string, problem states that total no. of a's are more than total no. of b's in input string.

2. PDA: Transition diagram:



PDA $P_F = (\{q_0, q_f\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \{q_f\})$

3. Transition moves

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

$\delta(q_0, b, z_0) = (q_0, bz_0)$

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

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

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

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

$\delta(q_0, \varepsilon, a) = (q_f, a)$

4. Instantaneous description

$\delta(q_0, aababab, z_0)$

$\vdash_P (q_0, ababab, az_0)$

$\vdash_P (q_0, babab, aaz_0)$

$\vdash_P (q_0, abab, az_0)$

$\vdash_P (q_0, bab, aaz_0)$

$\vdash_P (q_0, ab, az_0)$

$\vdash_P (q_0, b, aaz_0)$

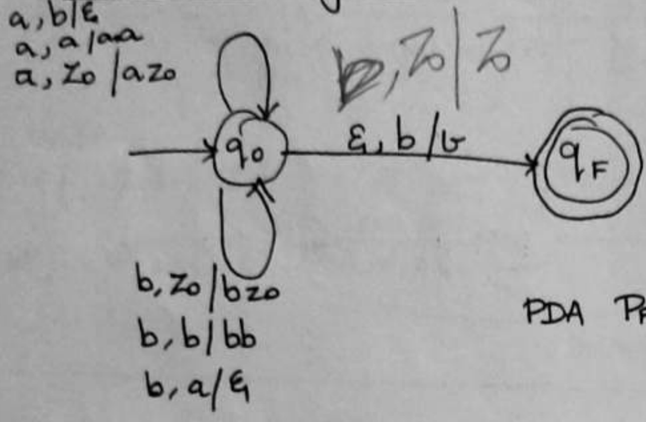
$\vdash_P (q_0, \varepsilon, az_0)$

$\vdash_P (q_F, a)$

$\therefore$ It is accepted.

Design PDA for the language that accepts strings with $n_a(w) < n_b(w)$

Transition diagram:



PDA $P_F = (\{q_0, q_F\}, \{a, b\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_F\})$

Transition moves

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

$\delta(q_0, b, Z_0) = (q_0, bZ_0)$

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

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

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

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

$\delta(q_0, \epsilon, b) = (q_F, b)$

Instantaneous description.

$\delta(q_0, abbab, Z_0)$

$\vdash_P (q_0, bbab, aZ_0)$

$\vdash_P (q_0, bab, Z_0)$

$\vdash_P (q_0, ab, bZ_0)$

$\vdash_P (q_0, b, Z_0)$

$\vdash_P (q_0, \epsilon, bZ_0)$

$\vdash_P (q_F, b)$

$\therefore$ It is accepted

DETERMINISTIC PUSH DOWN AUTOMATA (DPDA)

Definition:

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is deterministic if and only if it satisfies the following conditions

1. $\delta(q, a, x)$ has only one member for any given q in Q, a in Σ, or $a = \epsilon$, and x in Γ
2. If $\delta(q, a, x)$ is non empty for some a in Σ, then $\delta(q, \epsilon, x)$ must be empty.

Problem:

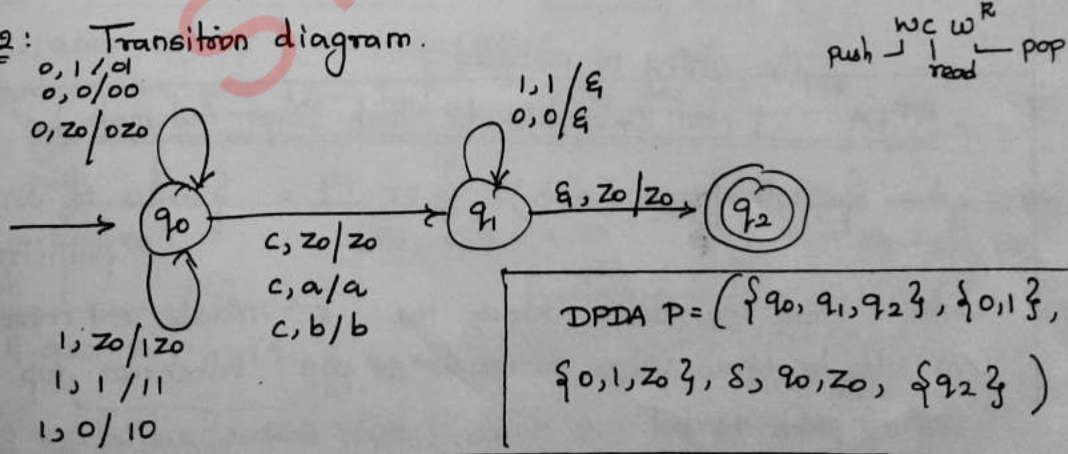
1. $L = \{wcw^R \mid w \text{ is in } (0+1)^*\}$ / odd Palindrome

Idea: The PDA for this is designed in such a way that DPDA is to store 0's and 1's on stack until it sees the middle end marker. After this, it goes to another state in which it matches i/p symbols against stack symbols and pops the stack if they match or else rejected. Thus the PDA is strictly DPDA.

It does not have a choice of move in the start state using the same input and stack symbol.

Step 1: $L = \{c, 0c0, 1c1, 01c10, 11011, 00c00, \ldots\}$

Step 2: Transition diagram.



DPDA $P = (\{q_0, q_1, q_2\}, \{0,1\}, \{0,1,Z_0\}, \delta, q_0, Z_0, \{q_2\})$

Step 3: Transition function

$\delta(q_0, 0, z_0) = (q_0, 0z_0)$
$\delta(q_0, 0, 0) = (q_0, 00)$
$\delta(q_0, 0, 1) = (q_0, 01)$
$\delta(q_0, 1, z_0) = (q_0, 1z_0)$
$\delta(q_0, 1, 1) = (q_0, 11)$
$\delta(q_0, 1, 0) = (q_0, 10)$

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

Step 4: Instantaneous description:

$w_1 = 011C110$

$\delta(q_0, 011C110, z_0) \vdash_P (q_0, 11C110, 0z_0)$
$\vdash_P (q_0, 1C110, 10z_0)$
$\vdash_P (q_0, C110, 110z_0)$
$\vdash_P (q_1, 110, 110z_0)$
$\vdash_P (q_1, 10, 10z_0)$
$\vdash_P (q_1, 0, 0z_0)$
$\vdash_P (q_1, \epsilon, z_0)$
$\vdash_P (q_2, \epsilon, z_0)$

∴ The string is accepted.

$w_2 = 01C1$

$\delta(q_0, 01C1, z_0)$
$\vdash_P (q_0, 1C1, 0z_0)$
$\vdash_P (q_0, C1, 10z_0)$
$\vdash_P (q_0, 1, 10z_0)$
$\vdash_P (q_1, \epsilon, 0z_0)$

No transition moves

∴ String is not accepted.

NPDA [Non Deterministic Push Down Automata]

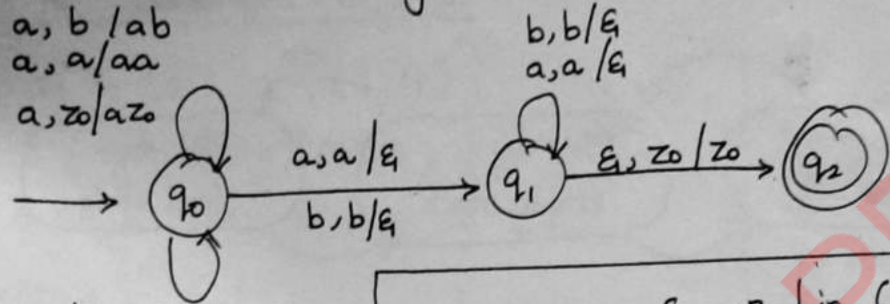
$L = \{ww^R \mid w \in (a,b)^*\}$ or $L = \{w/w$ is an even palindrome$\}$.

Idea: Here we don't know the 'c' middle end marker. So not able to know when to push or pop. Whenever top of stack and input symbol are same, then corresponding one change to centre (ie) when top of stack = input symbol, we have to assume that might be centre has come or not.

PDA → Accept : Centre has come.
PDA → Reject : Centre has not come.

Step 1: $L = \{ \epsilon, aa, bb, abba, aaaa, baab, \dots \}$

Step 2: Transition diagram.



NPDA for $L = \{ww^R \mid \text{in } (a,b)^*\}$

NPDA $P = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \{q_2\})$.

Step 3: Transition function:

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

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

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

$\delta(q_0, b, z_0) = (q_0, bz_0)$

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

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

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

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

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

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

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

Step 4: Instantaneous description $\boxed{w = aaaa}$

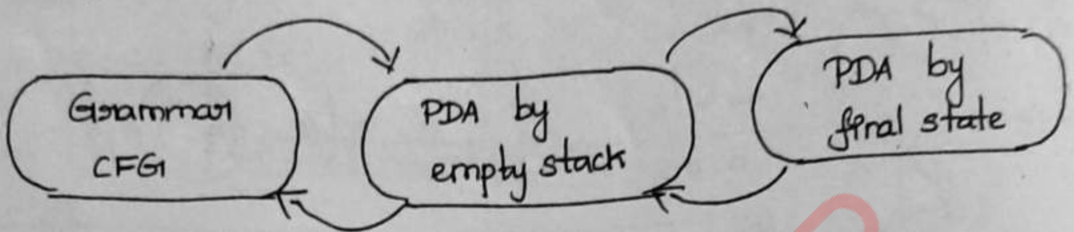
$\delta(q_0, aaaa, z_0) \vdash_P (q_0, aaa, az_0)$

- centre hasn't come (push) → (q_0, aa, aaz_0)
  - push → $(q_0, a, aaaz_0)$
    - push → $(q_0, \epsilon, aaaaz_0)$ ×
    - Pop → (q_1, ϵ, aaz_0) ×
    - Dead Configuration [No Transition]
  - pop → (q_1, a, az_0)
    - pop → (q_2, ϵ, z_0)
    - Final state
    - ∴ String is accepted
- Centre has come (Pop) → (q_1, aa, z_0) ×
  - (Dead configuration) No transition

EQUIVALENCE OF PDA'S AND CFL

3 CLASSES OF LANGUAGE

1. Context free language
2. Language accepted by final state of PDA
3. Language accepted by empty stack by PDA are under the same class.



Problems:

FROM GRAMMAR CFG TO PUSHDOWN AUTOMATA (PDA)

Definition: Let $G = (V, T, P, S)$ be a CFG.

Construct the PDA P that accepts L(G) by empty stack as follows

A PDA $\boxed{P = (\{q\}, T, V \cup T, \delta, q, S)}$ where δ is defined by,

(i) For each variable A in CFG

$$\delta(q, \epsilon, A) = \{(q, \beta) \mid A \to \beta \text{ is a production of } G\}$$

ii) For each terminal a in CFG

$$\delta(q, a, a) = \{(q, \epsilon)\}$$

PROBLEM:

1) Construct the PDA for the following grammar.

$$E \to E+E \mid E*E \mid a$$

Solution:

Step 1: $G: E \to E+E \mid E*E \mid a$

$V = \{E\}$ $\quad\quad$ $P = \{E \to E+E, E \to E*E, E \to a\}$

$T = \{+, *, a\}$ $\quad\quad$ $S = E$

Step 2: PDA $P = (\{q\}, \{+, *, a\}, \{E, *, +, a\}, \delta, q, E)$

Step 3: Transition function of PDA

<u>For Nonterminal Variable 'E'</u>

$\delta(q, \epsilon, E) = \{(q, E+E), (q, E*E), (q, a)\}$

<u>For terminal +, *, a</u>

$\delta(q, +, +) = \{(q, \epsilon)\}$

$\delta(q, *, *) = \{(q, \epsilon)\}$

$\delta(q, a, a) = \{(q, \epsilon)\}$

Selecting string:

LMD:

$E \underset{lmd}{\rightarrow} E*E \quad (E \rightarrow E*E)$

$\underset{lm}{\rightarrow} a*E \quad (E \rightarrow a)$

$\underset{lm}{\rightarrow} a*E+E \quad (E \rightarrow E+E)$

$\underset{lm}{\rightarrow} a*a+E \quad (E \rightarrow a)$

$\underset{lm}{\rightarrow} a*a+a \quad (E \rightarrow a)$

Step 4: Instantaneous description

$w = a*a+a$

$(q, a*a+a, E) \vdash_P (q, \epsilon\, a*a+a, E)$

$\vdash_P (q, a*a+a, E*E)$

$\vdash_P (q, a*a+a, a*E)$

$\vdash_P (q, *a+a, *E)$

$\vdash_P (q, a+a, E)$

$\vdash_P (q, a+a, E+E)$

$\vdash_P (q, a+a, a+E)$

$\vdash_P (q, +a, +E)$

$\vdash_P (q, a, E)$

$\vdash_P (q, a, a)$

$\vdash_P (q, \epsilon, \epsilon)$

Thus the CFG accepts the string $a*a+a$ and it is accepted by PDA by empty stack.

2. $S \to 0S1 \,/\, A$

$A \to 1A0 \,|\, S \,|\, \epsilon$

Step 1: $V = \{S, A\}$

$T = \{0, 1\}$

$P = \{S \to 0S1, S \to A, A \to 1A0, A \to S, A \to \epsilon\}$

$S = S$

Step 2: PDA $P = (\{q\}, \{0,1\}, \{S, A, 0, 1\}, \delta, q, S\}$

Step 3: Transition function

For variables S and A

i) $\delta(q, \epsilon, S) = \{(q, 0S1), (q, A)\}$

ii) $\delta(q, \epsilon, A) = \{(q, 1A0), (q, S), (q, \epsilon)\}$

For Terminals, 1 and 0

$\delta(q, 0, 0) = \{\delta(q, \epsilon)\}$

$\delta(q, 1, 1) = \{\delta(q, \epsilon)\}$

Step 4: Instantaneous description

$w_1 = 0101$

$\delta(q, 0101, S) \vdash_P (q, 0101, 0S1)$

$\vdash_P (q, 101, S1)$

$\vdash_P (q, 101, A1)$

$\vdash_P (q, 101, 1A01)$

$\vdash_P (q, 01, A01)$

$\vdash_P (q, 01, 01)$

$\vdash_P (q, 1, 1)$

$\vdash_P (q, \epsilon, \epsilon)$

Selecting string:

LMD:

$S \xrightarrow[lm]{} 0S1 \quad [S \to 0S1]$

$\xrightarrow[lm]{} 0A1 \quad [S \to A]$

$\xrightarrow[lm]{} 01A01 \quad [A \to 1A0]$

$\xrightarrow[lm]{} 01\epsilon01 \quad [A \to \epsilon]$

$\xrightarrow[lm]{} 0101$

$\therefore$ The string is accepted by empty stack.

3. $S \to aAA$, $A \to aS / bS / a$

Soln:

$V = \{S, A\}$

$T = \{a, b\}$

$P = \{S \to aAA, A \to aS, A \to bS, A \to a\}$

$S = S$

Step 1: PDA $P = (\{q\}, \{a, b\}, \{S, A, a, b\}, \delta, q, S)$

Step 2: For nonterminal (Variables): S, A

$\delta(q, \epsilon, S) = \{(q, aAA)\}$

$\delta(q, \epsilon, A) = \{(q, aS), (q, bS), (q, a)\}$

For terminals: a, b

$\delta(q, a, a) = \{(q, \epsilon)\}$

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

Step 3: Instantaneous description

$w_1 = aaaaaa$

$\delta(q, aaaaaa, S)$

$\vdash_P (q, aaaaaa, aAA)$

$\vdash_P (q, aaaaa, AA)$

$\vdash_P (q, aaaaa, aSA)$

$\vdash_P (q, aaaa, SA)$

$\vdash_P (q, aaaa, aAAA)$

$\vdash_P (q, aaa, AAA)$

$\vdash_P (q, aaa, aAA)$

$\vdash_P (q, aa, AA)$

$\vdash_P (q, aa, aA)$

$\vdash_P (q, a, A)$

$\vdash_P (q, a, a)$

$\vdash_P (q, \epsilon, \epsilon)$ ∴ The string is accepted by empty stack.

Selecting string

LMD:

$S \xrightarrow{lm} aAA$ $(S \to aAA)$

$\xrightarrow{lm} aaSA$ $(A \to aS)$

$\xrightarrow{lm} aaaAAA$ $(S \to aAA)$

$\xrightarrow{lm} aaaaAA$ $[A \to a]$

$\xrightarrow{lm} aaaaaA$ $[A \to a]$

$\xrightarrow{lm} aaaaaa$ $[A \to a]$

4. $I \to a | b | Ia | Ib | I0 | I1$, $E \to I | E*E | E+E | (E)$

Solution :-

$V = \{I, E\}$

$T = \{a, b, 0, 1, *, +, (,)\}$

$P = \{I, E, a, b, 0, 1, *, +, (,)\}$

$S = E$

Step 2: PDA $P = (\{q\}, \{a, b, 0, 1, *, +, (,)\}, \{I, E, a, b, 0, 1, *, +, (,)\}, \delta, q, E)$.

Step 3: For non terminals I, E

Transition functions:

$\delta(q, \epsilon, I) = \{(q, a), (q, b), (q, Ia), (q, Ib), (q, I0), (q, I1)\}$

$\delta(q, \epsilon, E) = \{(q, I), (q, E*E), (q, E+E), (q, (E))\}$

For terminals

$\delta(q, a, a) = \{(q, \epsilon)\}$
$\delta(q, b, b) = \{(q, \epsilon)\}$
$\delta(q, 0, 0) = \{(q, \epsilon)\}$
$\delta(q, 1, 1) = \{(q, \epsilon)\}$
$\delta(q, +, +) = \{(q, \epsilon)\}$
$\delta(q, *, *) = \{(q, \epsilon)\}$
$\delta(q, (, () = \{(q, \epsilon)\}$
$\delta(q,),)) = \{(q, \epsilon)\}$

Step 4: Instantaneous description:

$w_1 = ab01$

$\delta(q, ab01, E) \vdash_P (q, ab01, I)$
$\vdash_P (q, ab01, I1)$
$\vdash_P (q, ab01, I01)$
$\vdash_P (q, ab01, Ib01)$
$\vdash_P (q, ab01, ab01)$
$\vdash_P (q, b01, b01)$
$\vdash_P (q, 01, 01)$
$\vdash_P (q, 1, 1) \quad \vdash_P (q, \epsilon, \epsilon)$

$\therefore$ The string is accepted by empty stack.

CONVERSION FROM PDA TO CFG

RULE 1: If q_0 is start state then Q is set of states of PDA, then start production is given by $\boxed{S \rightarrow [q_0, Z_0, Q]}$

$S \rightarrow$ [initial state, initial stack symbol, each state in Q]

RULE 2: Production Rule for instantaneous description of the form

PUSH: $\delta(q_i, a, Z_0) = (q_{i+1}, Z_1 Z_2)$ then

$[q_i, Z_0, q_{i+K}] \longrightarrow a\,[q_{i+1}, Z_1, q_m][q_m, Z_2, q_{i+K}]$

Pop :

$$\delta(q_i, a, Z_0) = (q_{i+1}, \varepsilon)$$

$$[q_i, Z_0, q_{i+1}] \longrightarrow a$$

READ :

$$\delta(q_i, a, Z_0) = (q_{i+1}, Z_1)$$

$$[q_i, Z_0, q_{i+m}] \longrightarrow a\ [q_{i+1}, Z_1, q_{i+m}]$$

No. of Variables : $V = Q^2 M + 1$ [$Q \rightarrow$ No. of states, $M \rightarrow$ No. of stack symbols]

1) Convert PDA to CFG

$P = (\{p, q\}, \{0,1\}, \{x, z\}, \delta, q, z)$ where δ P defined by

$\delta(q,1,z) = (q, xz)$, $\delta(q,1,x) = (q, xx)$, $\delta(q,\varepsilon,x) = (q,\varepsilon)$

$\delta(q,0,x) = (p,x)$, $\delta(p,1,x) = (p,\varepsilon)$, $\delta(p,0,z) = (q,z)$.

Solution:

$$V = \{ S, \overset{A}{[pxp]}, \overset{B}{[pxq]}, \overset{C}{[qxp]}, \overset{D}{[qxq]}, \overset{E}{[pzp]}, \overset{F}{[pzq]}, \overset{G}{[qzp]}, \overset{H}{[qzq]} \}$$

$[V = Q^2 M + 1 = 2^2 \times 2 + 1 = 9]$

$T = \Sigma = \{0, 1\}$; $S = S$

Production P:

* For start symbol S, $S \rightarrow [q, z, p] \,/\, [q, z, q]$

$$\boxed{S \rightarrow G/H}$$

1,4 p, p 2,3
p, q
q, p
q, q

* For transition functions, (i) Push

$$\delta(q, 1, z) = (q, xz)$$

$[q, z, p] \longrightarrow 1\ [q, x, p]\ [p, z, p]$ ∴ $G \rightarrow 1CE$

$[q, z, p] \longrightarrow 1\ [q, x, q]\ [q, z, p]$ ∴ $G \rightarrow 1DG$

$[q, z, q] \rightarrow 1\ [q, x, p]\ [p, z, q]$

$[q, z, q] \rightarrow 1\ [q, x, q]\ [q, z, q]$

$\boxed{\begin{matrix} H \rightarrow 1CF \\ H \rightarrow 1DH \end{matrix}}$

ii) <u>Push</u>

$2 \times 2 = 4$ Production

$\delta(q, 1, x) = (q, xx)$

$[q, x, p] \rightarrow 1\ [q, x, p]\ [p, x, p]$

$[q, x, p] \rightarrow 1\ [q, x, q]\ [q, x, p]$

$[q, x, q] \rightarrow 1\ [q, x, p]\ [p, x, q]$

$[q, x, q] \rightarrow 1\ [q, x, q]\ [q, x, q]$

$\therefore \boxed{\begin{matrix} C \rightarrow 1CA \\ C \rightarrow 1DC \\ D \rightarrow 1CB \\ D \rightarrow 1DD \end{matrix}}$

iii) <u>Pop</u>

$2^0 = 1$ production

$\delta(q, \epsilon, x) = (q, \epsilon)$

$[q, x, q] \rightarrow \epsilon$

$\therefore \boxed{D \rightarrow \epsilon}$

iv) <u>Read</u>

$2^1 = 2$ Production

$\delta(q, 0, x) = [p, x)$

$[q, x, p] \rightarrow 0\ [p, x, p]$

$[q, x, q] \rightarrow 0\ [p, x, q]$

$\therefore \boxed{\begin{matrix} C \rightarrow 0A \\ D \rightarrow 0B \end{matrix}}$

v) <u>Pop</u>:

$\delta(p, 1, x) = (p, \epsilon)$

$[p, x, p] \rightarrow 1$

$\boxed{A \rightarrow 1}$

vi) <u>read</u>:

$\delta(p, 0, z) = (q, z)$

$[p, z, p] \rightarrow 0\ [q, z, p]$

$[p, z, q] \rightarrow 0\ [q, z, q]$

$\boxed{\begin{matrix} E \rightarrow 0G \\ F \rightarrow 0H \end{matrix}}$

$S \rightarrow G / H$

$G \rightarrow 1CE / 1DG$

$H \rightarrow 1CF / 1DH$

$C \rightarrow 1CA / 1DC / 0A$

$D \rightarrow 1CB / 1DD / 0B / \epsilon$

$A \rightarrow 1$; $E \rightarrow 0G$; $F \rightarrow 0H$

Since B has no transition (Production),

Remove B,

$S \rightarrow G / H$

$G \rightarrow 1CE / 1DG$

$H \rightarrow 1CF / 1DH$

$C \rightarrow 1CA / 1DC / 0A$

$D \rightarrow 1DD / \epsilon$

$A \rightarrow 1$; $E \rightarrow 0G$; $F \rightarrow 0H$

2) PDA $P = (\{q_0, q_1\}, \{0,1\}, \{x,y,z\}, \delta, q_0, z, \{q_1\})$

$\delta(q_0,0,z) = (q_1,z)$, $\delta(q_0,1,z) = (q_0,xz)$, $\delta(q_0,0,x) = (q_1,\epsilon)$

$\delta(q_0,1,x) = (q_0,xx)$, $\delta(q_1,0,z) = (q_1,yz)$, $\delta(q_1,1,z) = (q_0,z)$

$\delta(q_1,0,y) = (q_1,yy)$, $\delta(q_1,1,y) = (q_1,\epsilon)$

Solution:

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

$V = \{ S, \overset{A}{[q_0,z,q_0]}, \overset{B}{[q_0,z,q_1]}, \overset{C}{[q_1,z,q_0]}, \overset{D}{[q_1,z,q_1]},$
$\overset{E}{[q_0,x,q_0]}, \overset{F}{[q_0,x,q_1]}, \overset{G}{[q_1,x,q_0]}, \overset{H}{[q_1,x,q_1]},$
$\overset{I}{[q_0,y,q_0]}, \overset{J}{[q_0,y,q_1]}, \overset{K}{[q_1,y,q_0]}, \overset{L}{[q_1,y,q_1]}$

$T = \{0,1\}$; $S = S$

Production P:

i) Start symbol, S

$S \to [q_0,z,q_0] \,/\, [q_0,z,q_1]$

$\boxed{S \to A / B}$

ii) read: $\delta(q_0,0,z) = (q_1,z)$

$[q_0,z,q_0] \to 0[q_1,z,q_0]$

$[q_0,z,q_1] \to 0[q_1,z,q_1]$

$\boxed{A \to 0C \quad ; \quad B \to 0D}$

iii) Push $\delta(q_0,1,z) = (q_0,xz)$

$[q_0,z,q_0] \to 1[q_0,x,q_0][q_0,z,q_0]$

$[q_0,x,q_0] \to 1[q_0,x,q_1][q_1,z,q_0]$

$[q_0,z,q_1] \to 1[q_0,x,q_0][q_0,z,q_1]$

$[q_0,z,q_1] \to 1[q_0,x,q_1][q_1,z,q_1]$

$A \to 1EA$;	$B \to 1EB$
$A \to 1FC$;	$B \to 1FD$

iv) Pop $\delta(q_0,0,x) = (q_1,\epsilon)$

$[q_0,x,q_1] \to 0$

$\boxed{F \to 0}$

v) Push $\delta(q_0,1,x) = (q_0,xx)$

$[q_0,x,q_0] \to 1[q_0,x,q_0][q_0,x,q_0]$

$[q_0,x,q_0] \to 1[q_0,x,q_1][q_1,x,q_0]$

$[q_0,x,q_1] \to 1[q_0,x,q_0][q_0,x,q_1]$

$[q_0,x,q_1] \to 1[q_0,x,q_1][q_1,x,q_1]$

$E \to 1EE$	$F \to 1EF$
$E \to 1FG$	$F \to 1FH$

vi) Push $\delta(q_1,0,z) = (q_1,yz)$

$[q_1,z,q_0] \to 0[q_1,y,q_0][q_0,z,q_0]$

$[q_1,z,q_0] \to 0[q_1,y,q_1][q_1,z,q_0]$

$[q_1,z,q_1] \to 0[q_1,y,q_0][q_0,z,q_1]$

$[q_1,z,q_1] \to 0[q_1,y,q_1][q_1,z,q_1]$

$C \rightarrow 0KA$

$C \rightarrow 0LC$

$D \rightarrow 0KB$

$D \rightarrow 0LD$

vii) <u>Read</u>: $\delta(q_1, 1, z) = (q_0, z)$

$[q_1, z, q_0] \rightarrow 1\ [q_0, z, q_0]$

$[q_1, z, q_1] \rightarrow 1\ [q_0, z, q_1]$

$\boxed{\begin{array}{l} C \rightarrow 1A \\ D \rightarrow 1B \end{array}}$

viii) <u>Push</u>. $\delta(q_1, 0, y) = (q_1, yy)$

$[q_1, y, q_0] \rightarrow 0\ [q_1, y, q_0][q_0, y, q_0]$

$[q_1, y, q_0] \rightarrow 0\ [q_1, y, q_1][q_1, y, q_0]$

$[q_1, y, q_1] \rightarrow 0\ [q_1, y, q_0][q_0, y, q_1]$

$[q_1, y, q_1] \rightarrow 0\ [q_1, y, q_1][q_1, y, q_1]$

$\boxed{\begin{array}{lll} K \rightarrow 0KI & ; & L \rightarrow 0KJ \\ K \rightarrow 0LK & ; & L \rightarrow 0LL \end{array}}$

(x) <u>Pop</u>: $\delta(q_1, 1, y) = (q_1, \epsilon)$

$[q_1, y, q_1] \rightarrow 1 \qquad L \rightarrow 1$

<u>Transitions</u>

$S \rightarrow A/B$

$A \rightarrow 0C / 1EA / 1FC$

$B \rightarrow 0D / 1EB / 1FD$

$C \rightarrow 0KA / 0LC / 1A$

$D \rightarrow 0KB / 0LD / 1B$

$E \rightarrow 1EE / 1FG$

$F \rightarrow 0 / 1EF / 1FH$

$K \rightarrow 0KI / 0LK$

$L \rightarrow 1 / 0KJ / 0LL$

<u>There is no transition for G, H, I, J, So remove</u>

$S \rightarrow A/B$

$A \rightarrow 0C / 1EA / 1FC$

$B \rightarrow 0D / 1EB / 1FD$

$C \rightarrow 0KA / 0LC / 1A$

$D \rightarrow 0KB / 0LD / 1B$

$E \rightarrow 1EE$

$F \rightarrow 0 / 1EF$

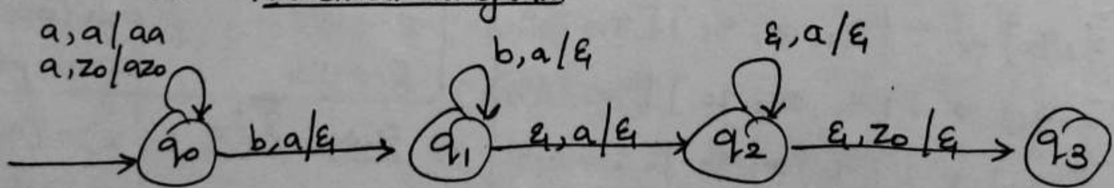
$K \rightarrow 0LK \quad ; \quad L = 1 / 0LL$

<u>CONVERSION FROM PDA BY EMPTY STACK TO FINAL STATE</u>

1) Construct a PDA to accept the language $L = \{a^n b^m / n > m\}$ by reaching empty stack and Convert into PDA by final state.

<u>Soln</u>: 1. $L = \{aab, aaabb, aaab, \ldots\}$

2. <u>Transition diagram</u>



PDA $P_N = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$

$\therefore$ PDA for $L = \{a^n b^m / n > m\}$ by empty stack

3. Transition function:

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

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

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

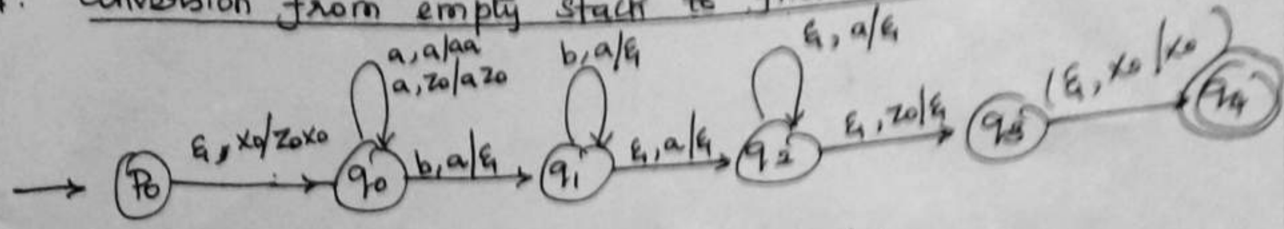
$\delta(q_1, b, a) = (q_1, \epsilon)$

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

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

$\delta(q_2, \epsilon, z_0) = (q_3, \epsilon)$

4. Conversion from empty stack to final state:



PDA for $L = \{a^n b^m \mid n > m\}$ by final state

$P_F = (\{P_0, q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, \{a, z_0, x_0\}, \delta, P_0, x_0, \{q_4\})$

5. Instantaneous description:-

(i) $w = aab$ by empty stack P_N

$(q_0, aab, z_0) \vdash_P (q_0, ab, az_0)$

$\vdash_P (q_0, b, aaz_0)$

$\vdash_P (q_1, \epsilon, az_0)$

$\vdash_P (q_2, \epsilon, z_0)$

$\vdash_P (q_3, \epsilon, \epsilon)$

$\therefore$ The string is accepted by empty stack.

(ii) $w = aab$ by final state P_F

$(P_0, aab, x_0) \vdash_P (P_0, \epsilon aab, x_0)$

$\vdash_P (q_0, aab, z_0x_0)$

$\vdash_P (q_0, ab, az_0x_0)$

$\vdash_P (q_0, b, aaz_0x_0)$

$\vdash_P (q_1, \epsilon, az_0x_0)$

$\vdash_P (q_2, \epsilon, z_0x_0)$

$\vdash_P (q_3, \epsilon, x_0)$

$\vdash_P (q_4, \epsilon, x_0)$

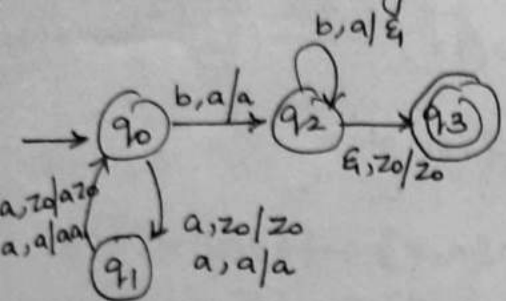
$\therefore$ The string is accepted by final state.

CONVERSION FROM PDA BY FINAL STATE TO PDA BY EMPTY STACK

Design a PDA to accept the language $L = \{a^{2n} b^{n+1} \mid n \geq 1\}$ by reaching final state and convert this PDA by reaching FINAL STATE into PDA by empty stack.

Soln: 1. $L = \{aabb, aaaabbb, aaaaaabbbb, \dots\}$

2. Transition Diagram.



3. Transition Function

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

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

$\delta(q_0, b, a) = (q_2, a)$

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

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

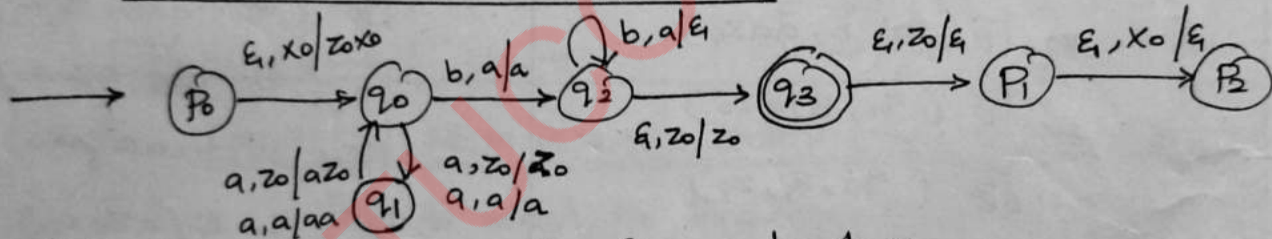
$\delta(q_2, b, a) = (q_2, \varepsilon)$

$\delta(q_2, \varepsilon, Z_0) = (q_3, Z_0)$

PDA for $L = \{a^{2n} b^{n+1} \mid n \geq 1\}$ by final state

$P_F = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \{q_3\})$

4. CONVERSION: PDA OF EMPTY STACK



PDA for $L = \{a^{2n} b^{n+1} \mid n \geq 1\}$ by empty stack

$P_N = (\{P_0, q_0, q_1, q_2, q_3, P_1, P_2\}, \{a, b\}, \{a, Z_0, X_0\}, \delta, P_0, X_0, \{q_3$

5. Instantaneous description:

$w = aabb$ By final state (P_F)

$(q_0, aabb, Z_0) \vdash_P (q_1, abb, Z_0)$

$\vdash_P (q_0, bb, aZ_0)$

$\vdash_P (q_2, b, aZ_0)$

$\vdash_P (q_2, \varepsilon, Z_0)$

$\vdash_P (q_3, \varepsilon, Z_0)$

Accepted by final state

$w = aabb$ by empty stack (P_N)

$(P_0, aabb, X_0) \vdash_P (P_0, \varepsilon\, aabb, X_0)$

$\vdash_P (q_0, aabb, Z_0X_0)$

$\vdash_P (q_1, abb, Z_0X_0) \vdash_P (q_0, bb, aZ_0X_0)$

$\vdash_P (q_2, b, aZ_0X_0) \vdash_P (q_2, \varepsilon, Z_0X_0)$

$\vdash_P (q_3, \varepsilon, Z_0X_0) \vdash_P (P_1, \varepsilon, X_0)$

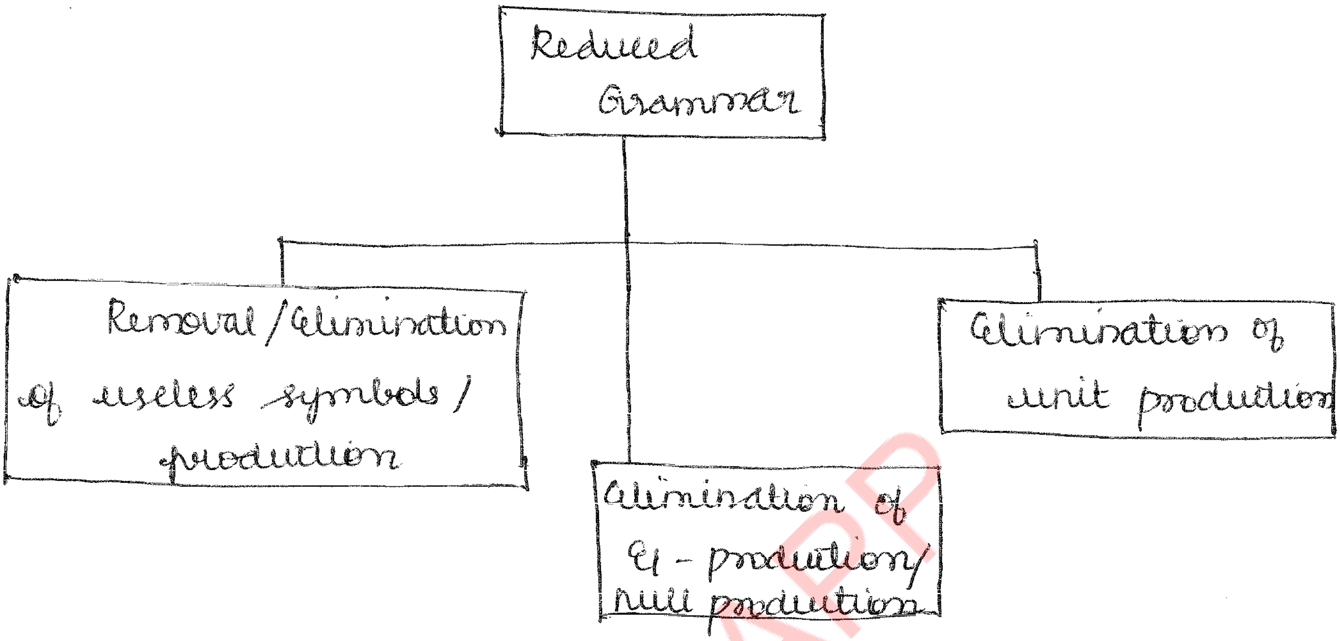
$\vdash_P (P_2, \varepsilon, \varepsilon)$ ∴ Accepted by empty stack

UNIT IV
PROPERTIES OF CONTEXT FREE LANGUAGES

SIMPLICATION OF CFG:

Simplification of CFG means reduction of grammar by removing useless symbols, thus reducing the length of grammar.

The properties of reduced grammar are,



Eliminating Useless symbols / production :

Let $G = (V, T, P, S)$ a grammar. A symbol 'x' is useful if there is a derivation, $S \stackrel{*}{\Rightarrow} \alpha x \beta \stackrel{*}{\Rightarrow} w$ for some α, β and w where w is in T^*, Otherwise it is useless.

There are two ways to find useful production.

(1) Some terminal string must be derived from 'x'.

(2) x must occur in some string derived from S.

Two terms are involved,

(1) Generating symbols (2) Reachable symbols.

Generating symbol: If 'x' is generating if $X \stackrel{*}{\Rightarrow} w$ for some T^* in w,

Steps:

(1) Every symbol of T is generating, therefore it generate itself

(2) If A tends to a $(A \rightarrow a)$, then A is also generating for $\boxed{a \in T \text{ or } a \in \epsilon}$

Reachable symbol: 'x' is reaching if there is a derivation, $\boxed{S \stackrel{*}{\Rightarrow} \alpha \, x \, \beta}$ for some α & β.

Steps:

(1) S is a reachable because S is a start symbol.

(2) If A is reachable, then all production with A in the head, all symbols of those production are also reachable.

PROBLEMS:

(1) $S \rightarrow AB \mid a, \; A \rightarrow BC \mid b, \; B \rightarrow aB \mid C, \; C \rightarrow aC \mid B$

Solution:

(1) Identify all generating variable:

(2) Generating symbols are $\{a, b, c, S, A\}$

(3) Useless symbol is B, C & eliminate B, C

$S \rightarrow a$

$A \rightarrow b$

(4) Removing unreachable production / useless

unreachable production: $A \rightarrow b$

(5) Ans : Useful production : $S \to a$.

(2) $S \to aS \mid A \mid C$

$A \to a$

$B \to aa$

$C \to aCb$

Solution :

(1) Generating Symbols : $\{a, b, A, B, S\}$

(2) Useless symbol : C

$\therefore$ $S \to aS \mid A$

$A \to a$

$B \to aa$

(3) Unreachable symbol / production : B

Ans : $\boxed{\begin{array}{l} S \to aS \\ A \to a \end{array}}$ → Useful production.

(3) $S \to aA \mid a \mid Bb \mid cC$

$A \to aB$

$B \to a \mid Aa$

$C \to cCD$

$D \to ddd$

Solution : (1) Generating symbols : $\{a, b, c, d, S, A, B, D\}$

(2) Useless symbol : C

$\therefore$ $S \to aA \mid a \mid Bb$

$A \to aB$

$B \to a \mid Aa$

$D \to ddd$

(3) Unreachable symbol / production : D

Ans : $\boxed{\begin{array}{l} S \to aA \mid a \mid Bb \\ A \to aB \\ B \to a \mid Aa \end{array}}$ → Useful production

(4) $S \to aA \mid bB$

$A \to aA \mid a$

$B \to bB$

$D \to ab \mid Ea$

$E \to ac \mid d$

Solution:

(1) Generating symbol : $\{a, b, c, d, S, A, E, D\}$

(2) Useless symbol : B.

$\therefore$ $S \to aA$

$A \to aA \mid a$

$D \to ab \mid Ea$

$E \to ac \mid d$

(3) Unreachable symbol : D, E

$\therefore$ Ans = $\boxed{\begin{array}{l} S \to aA \\ A \to aA \mid a \end{array}}$ → Useful production.

Eliminating ϵ / Null production:

A production which is of the form $A \to \epsilon$ is called ϵ- production. If ϵ is in $L(G)$, it is not possible to eliminate all ϵ- production. The same is possible if ϵ is not in $L(G)$.

For each variable A, if $A \overset{*}{\Rightarrow} \epsilon$, then A is called as nullable variable.

We need to check whether the variable is nullable or not.

If $B \to C_1 . C_2 . C_3 \dots . C_n$ where each C_i is nullable, then B is nullable.

PROBLEMS:

(1) $S \rightarrow aSa \mid bAb$

$A \rightarrow \epsilon$

Solution:

(i) $V = \{S, A\}$

(ii) Null production : $A \rightarrow \epsilon$

(iii) Nullable variable : $\{A\}$

(iv) Eliminate wherever A is there which should not affect corresponding grammar.

If we remove A, bAb
$\Rightarrow b\epsilon b$
$\Rightarrow bb$

$\therefore S \rightarrow aSa \mid \cancel{bAb} \mid bb$

$A \rightarrow \epsilon$ ×   (bAb can be eliminated because A is useless symbol)

$$\boxed{\therefore S \rightarrow aSa \mid bb}$$

(2) $S \rightarrow AB$

$A \rightarrow aAA \mid \epsilon$

$B \rightarrow bBB \mid \epsilon$

Solution:

(i) $V = \{S, A, B\}$

(ii) Null production : $\{A \rightarrow \epsilon, B \rightarrow \epsilon\}$

(iii) Nullable variable : $\{A, B, S\}$

(iv) Find production with & without nullable variable.

$S \rightarrow AB \mid A \mid B \mid \cancel{\epsilon}$ $(\because AB \rightarrow A\epsilon)$ A

$A \rightarrow aAA \mid aA \mid \overset{\text{dup}}{\cancel{aA}} \mid a \mid \cancel{\epsilon}$

$B \rightarrow bBB \mid bB \mid \cancel{bB} \mid b \mid \cancel{\xi}$ duplicate copy

$\therefore$ $\boxed{\begin{array}{l} S \rightarrow AB \mid A \mid B \\ A \rightarrow aAA \mid aA \mid a \\ B \rightarrow bBB \mid bB \mid b \end{array}}$ [Eliminate null & duplicate values]

(3) $A \rightarrow 0B1 \mid 1B1$

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

<u>Solution :</u>

(1) $V = \{A, B\}$

(2) Null production : $B \rightarrow \xi$

(3) Nullable variable : $\{B\}$

(4) find production with & without nullable variables

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

$B \rightarrow 0B \mid 0 \mid 1B \mid 1 \mid \cancel{\xi}$

$\therefore$ $\boxed{\begin{array}{l} A \rightarrow 0B1 \mid 01 \mid 1B1 \mid 11 \\ B \rightarrow 0B \mid 0 \mid 1B \mid 1 \end{array}}$

(4) $S \rightarrow a \mid Ab \mid aBa$

$A \rightarrow b \mid \xi$

$B \rightarrow b \mid A$

<u>Solution :</u>

(1) Variable : $\{S, A, B\}$

(2) Null production : $A \rightarrow \xi$

(3) Nullable variable : $\{A, B\}$

(4) find production,

$S \to a \mid Ab \mid b \mid aBa \mid aa$

$A \to b \mid \cancel{\epsilon}$

$B \to b \mid A \mid \cancel{\epsilon}$

$$\therefore \boxed{\begin{array}{l} S \to a \mid Ab \mid b \mid aBa \mid aa \\ A \to b \\ B \to b \mid A \end{array}}$$

Elimination of unit production:

A unit production is a production which is of the form $A \to B$ where both A & B are variables.

UNIT PAIR: If the sequence of derivation steps are $A \Rightarrow B_1 \Rightarrow B_2 \ldots B_n \Rightarrow \alpha$, then these unit productions are replaced by a non-unit production, $B_n \to \alpha$ directly from A.

$$\therefore A \to \alpha$$

(A, B) such that $A \overset{*}{\Rightarrow} B$ is called an unit pair.

How to eliminate unit production:

Given a CFG, $G = (V, T, P, S)$ with unit production, then construct a new CFG $G_1 = (V, T, P_1, S)$

(1) Find all the unit pair of G.

(2) For each unit pair (A, B) if there is a production $A \to B$ replace it with $A \to \alpha$ provided $B \to \alpha$ is a production in G.

PROBLEMS :

(1) $S \rightarrow Aa \mid B$

$B \rightarrow A \mid bb$

$A \rightarrow a \mid bc \mid B$

Solution :

(i) Find all unit production

$S \rightarrow B$

$B \rightarrow A$

$A \rightarrow B$

(ii) $S \rightarrow B$
$\rightarrow A$
$\rightarrow a \mid bc$

$S \rightarrow B$
$\rightarrow bb$

$B \rightarrow A$
$\rightarrow a \mid bc$

$B \rightarrow bb$

$A \rightarrow B$
$\rightarrow A \mid bb$
$\rightarrow a \mid bc \mid bb$

$\therefore$

$$\boxed{\begin{array}{l} S \rightarrow Aa \mid a \mid bc \mid bb \\ B \rightarrow a \mid bc \mid bb \\ A \rightarrow a \mid bc \mid bb \end{array}}$$

(2) $S \rightarrow 0A \mid 1B \mid C$

$A \rightarrow 0S \mid 00$

$B \rightarrow 1 \mid A$

$C \rightarrow 01$

Solution :

(i) Find all unit productions

$S \rightarrow C$

$B \rightarrow A$

(ii) $S \rightarrow C$ $\quad$ $B \rightarrow A$

$\rightarrow 01$ $\quad$ $\rightarrow 0S \mid 00$

$\therefore$

$$\boxed{\begin{aligned} S &\rightarrow 0A \mid 1B \mid 01 \\ A &\rightarrow 0S \mid 00 \\ B &\rightarrow 1 \mid 0S \mid 00 \\ C &\rightarrow 01 \end{aligned}}$$

$\Rightarrow$ Remove unreachable production

C is unreachable

$\therefore$ **Ans:**

$$\boxed{\begin{aligned} S &\rightarrow 0A \mid 1B \mid 01 \\ A &\rightarrow 0S \mid 00 \\ B &\rightarrow 1 \mid 0S \mid 00 \end{aligned}}$$

---

(3) $S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow C \mid b$

$C \rightarrow D$

$D \rightarrow E \mid bC$

$E \rightarrow d \mid Ab$

Solution:

(i) Find all unit production

$B \rightarrow C$

$C \rightarrow D$

$D \rightarrow E$

(ii) $B \rightarrow C$ $\quad$ $D \rightarrow E$ $\quad$ $C \rightarrow D$

$\rightarrow d \mid Ab \mid bc \mid b$ $\quad$ $\rightarrow d \mid Ab$ $\quad$ $C \rightarrow d \mid Ab \mid bC$

$\therefore$

$$\boxed{\begin{aligned} S &\rightarrow AB \\ A &\rightarrow a \\ B &\rightarrow d \mid Ab \mid bC \mid b \\ C &\rightarrow d \mid Ab \mid bC \\ D &\rightarrow d \mid Ab \mid bC \\ E &\rightarrow d \mid Ab \end{aligned}}$$

(iii) Remove unreachable production:

D and E

Ans : ∴

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow d \mid Ab \mid bC \mid b$

$C \rightarrow d \mid Ab \mid bC$

NORMAL FORM OF CFG :

(1) Chomsky Normal form (CNF)

(2) Greibach Normal Form (GNF)

CONVERSION FROM CFG INTO CNF :

Any CFL without ϵ is generated by a grammar in which all productions are of the form $A \rightarrow BC$ (or) $A \rightarrow a$ where A, B, C are variables and a is a terminal.

Steps :

(1) Write down the rule of CFG

$$\boxed{\begin{array}{lcl} \text{Non-Terminal (NT)} & \rightarrow & \text{NT NT} \\ \text{NT} & \rightarrow & \text{Terminal} \end{array}}$$

(2) write the given production.

(3) Simplify the CFG

(3.1) elimination of ϵ-production

(3.2) elimination of unit production

(3.3) elimination of useless production.

(4) Convert CFG into CNF

(5) Write down the resultant productions.

PROBLEMS :

(1) construct the grammar $(\{S, A, B\}, \{a, b\}, P, S)$ has the production $S \rightarrow bA \mid aB$ Convert into CNF.

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

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

Solution :

(1) Rule of CNF :

$$\boxed{\begin{array}{l} NT \rightarrow NT \ NT \\ NT \rightarrow T \end{array}}$$

(2) write the given production

$S \rightarrow bA \mid aB$

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

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

(3) Simplify the CFG.

(3.1) Eliminate ε- production :

There is no ε-production in the given Grammar. Then CFG is,

$S \rightarrow bA \mid aB$

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

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

(3.2) Eliminate unit production :

There is no unit production in the given Grammar. Then CFG is,

$S \rightarrow bA \mid aB$

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

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

(3.3) elimination of useless production :

There is no useless production in the given Grammar. Then CFG is,

$S \to bA \mid aB$

$A \to bAA \mid aS \mid a$

$B \to aBB \mid bS \mid b$

(4) Simplify CFG to CNF :

$S \to bA$ (Rule 1)

$S \to aB$ (Rule 2)

$A \to bAA$ (Rule 3)

$A \to aS$ (Rule 4)

$A \to a$ (Rule 5) // CNF format

$B \to aBB$ (Rule 6)

$B \to bS$ (Rule 7)

$B \to b$ (Rule 8) // CNF format

Rule 1 :

$S \to \underline{b}A$

$$\boxed{\begin{array}{l} S \to C_b A \\ C_b \to b \end{array}}$$

Rule 2 :

$S \to \underline{a}B$

$$\boxed{\begin{array}{l} S \to C_a B \\ C_a \to a \end{array}}$$

Rule 3 :

$A \to \underline{b}AA$

$\to C_b \underline{AA}$

$$\boxed{\begin{array}{l} A \to C_b D_1 \\ C_b \to b \\ D_1 \to AA \end{array}}$$

Rule 4 :

$A \to \underline{a}S$

$$\boxed{A \to C_a S}$$

Rule 6 :

$B \to \underline{a}BB$

$\to C_a \underline{BB}$

$$\boxed{\begin{array}{l} B \to C_a D_2 \\ D_2 \to BB \end{array}}$$

Rule 7 :

$B \to \underline{b}S$

$$\boxed{B \to C_b S}$$

(5) Resultant productions are,

$$\boxed{\begin{array}{l} S \to C_b A \mid C_a B \\ A \to C_b D_1 \mid C_a S \mid a \\ B \to C_a D_2 \mid C_b S \mid b \\ D_1 \to AA \\ D_2 \to BB \\ C_a \to a \\ C_b \to b \end{array}}$$

---

(2) $S \to ASB \mid \epsilon$
$A \to aAS \mid a$. convert into CNF.
$B \to SbS \mid A \mid bb$

Solution :

(1) Rule of CNF :

$$\boxed{\begin{array}{l} NT \to NT\ NT \\ NT \to T \end{array}}$$

(2) Given production :

$S \to ASB \mid \epsilon$
$A \to aAS \mid a$
$B \to SbS \mid A \mid bb$.

(3) Simplify CFG :

(3.1) Elimination of ϵ - production :

$V = \{S, A, B\}$

Null production : $S \to \epsilon$

Nullable variable : $\{S\}$

$S \rightarrow ASB \mid AB \mid \cancel{\epsilon}$

$A \rightarrow aAS \mid aA \mid a$

$B \rightarrow SbS \mid bS \mid Sb \mid b \mid A \mid bb$

$\therefore$
$$\boxed{\begin{array}{l} S \rightarrow ASB \mid AB \\ A \rightarrow aAS \mid aA \mid a \\ B \rightarrow SbS \mid bS \mid Sb \mid b \mid A \mid bb \end{array}}$$

(3.2) <u>Eliminate unit production</u> :

- Find all unit production.

$B \rightarrow A$

- $B \rightarrow A$ $\quad\quad$ $B \rightarrow A$ $\quad\quad$ $B \rightarrow A$

$\rightarrow aAS$ $\quad\quad$ $\rightarrow a$ $\quad\quad$ $\rightarrow aA$

$\therefore$ There is no unreachable production

$\therefore$
$$\boxed{\begin{array}{l} S \rightarrow ASB \mid AB \\ A \rightarrow aAS \mid aA \mid a \\ B \rightarrow SbS \mid bS \mid Sb \mid b \mid aAS \mid aA \mid a \mid bb \end{array}}$$

(3.3) <u>eliminate useless production</u> :

There is no useless production in the given Grammar G.

Then CFG,
$$\boxed{\begin{array}{l} S \rightarrow ASB \mid AB \\ A \rightarrow aAS \mid aA \mid a \\ B \rightarrow SbS \mid bS \mid Sb \mid b \mid aAS \mid aA \mid a \mid bb \end{array}}$$

(4) Simplify CFG to CNF :

- $S \to ASB$

$$\boxed{\begin{array}{l} S \to AD_1 \\ D_1 \to SB \end{array}}$$

$S \to AB$

$$\boxed{S \to AB}$$

- $A \to aAS$

$\to C_a AS$

$$\boxed{\begin{array}{l} A \to C_a D_2 \\ C_a \to a \\ D_2 \to AS \end{array}}$$

$A \to aA$

$$\boxed{A \to C_a A}$$

$A \to a$

- $B \to SbS$

$\to S C_b S$

$$\boxed{\begin{array}{l} B \to S D_3 \\ D_3 \to C_b S \\ C_b \to b \end{array}}$$

$B \to Sb$

$$\boxed{B \to S C_b}$$

$B \to bS$

$$\boxed{B \to C_b S}$$

$$\boxed{B \to b}$$

$B \to aAS$

$\to C_a AS$

$$\boxed{B \to C_a D_2}$$

$B \to aA$

$$\boxed{B \to C_a A}$$

$$\boxed{B \to a}$$

$B \to bb$

$\to C_b b$

$$\boxed{B \to C_b C_b}$$

(5) Final productions are,

$$\boxed{\begin{array}{l} S \to AD_1 \mid AB \\ A \to C_a D_2 \mid C_a A \mid a \\ B \to SD_3 \mid SC_b \mid C_b S \mid b \mid C_a D_2 \mid C_a A \mid a \mid C_b C_b \\ D_1 \to SB \\ D_2 \to AS \\ D_3 \to C_b S \\ C_a \to a \\ C_b \to b \end{array}}$$

GREIBACK NORMAL FORM :

A CFG is in GNF if the productions are in the following form,

$$\boxed{\begin{array}{l} A \rightarrow b \quad (\text{or}) \\ A \rightarrow b C_1 C_2 \cdots\cdots C_n \end{array}}$$

where $A, C_1, C_2 \cdots C_n$ are variables and b is a terminal.

Note :

Rule :

$$\boxed{\begin{array}{l} NT \rightarrow T \\ NT \rightarrow T\ NT \cdots\cdots NT \end{array}}$$

CONVERSION FROM CFG INTO GNF :

Steps :

(1) Simplify CFG (eliminating ∈ - production, unit production, useless production)

(2) Check whether the simplified CFG in CNF format or not. If not convert it into CNF.

(3) Change the names of the non-terminal symbols into some A_i in ascending order of i.

(4) Alter the rules so that, non-terminal symbols are in ascending order such that if the production is of the form $A_i \rightarrow A_j \alpha$ then $i < j$ & should never be $i \geq j$.

(5) Remove left recursion production $\boxed{A_i \rightarrow A_i \alpha}$

Rules : By Introducing new variable, $B_i \rightarrow \alpha B_i \mid \alpha$

(6) Check whether the production is in GNF format or not. If it is not, then convert it into GNF.

(7) Write the final set of production in given CFG order

PROBLEM:

(1) $S \to CA \mid BB$

$B \to b \mid SB$

$C \to b$

$A \to a$

Solution:

(1.1) Eliminate ϵ - production:

There is no ϵ - production in given Grammar.

(1.2) Eliminate unit production:

No unit production

(1.3) Eliminate useless production:

No useless production.

(2) Check whether simplified CFG is CNF format or not.

All are in CNF.

$\therefore$ $S \to CA \mid BB$

$B \to b \mid SB$

$C \to b$

$A \to a$

All production are in CNF format

(3) Change the name of NT (Non-terminal) symbols.

Replace S by A_1

C by A_2

A by A_3

B by A_4.

we get,

$$A_1 \rightarrow A_2 A_3 \,|\, A_4 A_4$$
$$A_4 \rightarrow b \,|\, A_1 A_4$$
$$A_2 \rightarrow b$$
$$A_3 \rightarrow a$$

(4) Alter the rules so that, NT symbols are in ascending order. $[A_i \rightarrow A_j \alpha]$

$$A_1 \rightarrow A_2 A_3$$
$i=1,\ j=2 \qquad i<j$

$$A_1 \rightarrow A_4 A_4$$
$i=1,\ j=4 \qquad i<j$

$A_4 \rightarrow b$ // GNF format

$A_4 \rightarrow A_1 A_4 \rightarrow$ 1
$i=4,\ j=1 \qquad i>j$

Sub $A_1 = A_2 A_3 \,/\, A_4 A_4$ in 1

$\therefore A_4 \rightarrow A_2 A_3 A_4 \,|\, A_4 A_4 A_4 \rightarrow$ 2
$i=4,\ j=2 \quad i>j \qquad i=4,\ j=4 \quad i \geq j$.

sub $A_2 = b$ in 2

$A_4 \rightarrow b A_3 A_4 \,|\, A_4 A_4 A_4 \rightarrow$ 3

(5) Here $i=j$ & then remove left recursion

$A_4 \rightarrow b A_3 A_4$ (T NT NT) // GNF format

$$\overset{A}{A_4} \rightarrow \overset{A}{A_4}\,\overset{\alpha}{A_4 A_4}$$

since $\boxed{\begin{array}{l} A \rightarrow \alpha \\ B \rightarrow \alpha B \,|\, \alpha \end{array}}$

Introducing B_4 as new variable

$$B_4 \to A_4 A_4 B_4 \mid A_4 A_4 .$$

$$A_4 \to b \mid bA_3A_4 \mid bB_4 \mid bA_3A_4B_4$$ // GNF format

$\therefore$

$$\boxed{\begin{array}{l} A_1 \to \underline{A_2} A_3 \mid \underline{A_4} A_4 \\ A_4 \to b \mid bB_4 \mid bA_3A_4 \mid bA_3A_4B_4 \\ A_2 \to b \\ A_3 \to a \end{array}}$$

(6) Check whether given production are in GNF or not

$$\boxed{\begin{array}{l} A_1 \to bA_3 \mid bA_4 \mid bB_4A_4 \mid bA_3A_4B_4 \mid bA_3A_4B_4A_4 \\ A_4 \to b \mid bB_4 \mid bA_3A_4 \mid bA_3A_4B_4 \\ A_2 \to b \\ A_3 \to a \\ B_4 \to A_4A_4B_4 \mid A_4A_4 \end{array}}$$

---

(2) $S \to AB$

$A \to BS \mid b$. Convert into GNF .

$B \to SA \mid a$

Solution :

(1.1) Eliminate ε-production : No ε-production

(1.2) Eliminate unit production : No unit production

(1.3) Eliminate useless production : No useless production

(2) Check whether the simplified CFG in CNF format or not ,

$$S \rightarrow AB$$
$$A \rightarrow BS \mid b$$
$$B \rightarrow SA \mid a$$
All are in CNF.

(3) Change the names of NT symbols into some A_i in ascending order of i.

Replace S by A_1

A by A_2

B by A_3

we get:

$$A_1 \rightarrow A_2 A_3$$
$$A_2 \rightarrow A_3 A_1 \mid b$$
$$A_3 \rightarrow A_1 A_2 \mid a$$

(4) Alter the rules so that, NT symbols are in ascending order. $A_i \rightarrow A_j \alpha$

$$A_1 \rightarrow A_2 A_3$$
$i=1,\ j=2 \qquad i<j$

$$A_2 \rightarrow A_3 A_1 \mid b$$
$i=2,\ j=3 \qquad i<j$

$$A_3 \rightarrow A_1 A_2 \mid a \rightarrow (1)$$
$i=3,\ j=1 \qquad i>j$

Sub $A_1 = A_2 A_3$ in (1)

$$\therefore A_3 \rightarrow \underline{A_2} A_3 A_2 \mid a \rightarrow (2)$$
$j=2,\ i=3 \qquad i>j$

Sub $A_2 = A_3 A_1 \mid b$ in (2)

$$A_3 \rightarrow A_3 A_1 A_3 A_2 \mid \overset{T}{b} \overset{NT}{A_3} \overset{NT}{A_2} \mid \overset{T}{a}$$
$i=3,\ j=3 \qquad i=j$

(5) Here $i = j$ & then remove left recursion

$A_3 \to bA_3 A_2 \mid a$ // GNF format

$A_3 \to \overset{A}{A_3} A_1 \overset{\alpha}{A_3 A_2}$

Since $\boxed{\begin{array}{l} A \to \alpha \\ B \to \alpha B \mid \alpha \end{array}}$ Introduce B_3 new variable

$\therefore \quad B_3 \to A_1 A_3 A_2 B_3 \mid A_1 A_3 A_2$

$A_3 \to bA_3A_2 \mid a \mid bA_3A_2B_3 \mid aB_3$

$\therefore$

$A_1 \to A_2 A_3$

$A_2 \to A_3 A_1 \mid b$

$A_3 \to bA_3A_2 \mid a \mid bA_3A_2B_3 \mid aB_3$

(6) Check whether given production are in GNF or not.

$A_1 \to \underline{A_2} A_3$

$\to \underline{A_3} A_1 A_3 \mid b$

$A_1 \to bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_2B_3A_1A_3 \mid aB_3A_1A_3 \mid b$.

$A_2 \to \underline{A_3} A_1 \mid b$

$\to bA_3A_2A_1 \mid aA_1 \mid bA_3A_2B_3A_1 \mid aB_3A_1 \mid b$

$A_3 \to bA_3A_2 \mid a \mid bA_3A_2B_3 \mid aB_3$

$\therefore$

$A_1 \to A_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_2B_3A_1A_3 \mid aB_3A_1A_3 \mid b$

$A_2 \to bA_3A_2A_1 \mid aA_1 \mid bA_3A_2B_3A_1 \mid aB_3A_1 \mid b$

$A_3 \to bA_3A_2 \mid a \mid bA_3A_2B_3 \mid aB_3$

$B_3 \to bA_3A_2A_1A_3A_2B_3 / aA_1A_3A_3A_2B_3 / bA_3A_2B_3A_1A_3A_3A_2B_3 /$
$aB_3A_1A_3A_3A_2B_3 / bA_3A_2B_3 / bA_3A_2A_1A_3A_3A_2 / aA_1A_3A_3A_2 /$
$bA_3A_2B_3A_1A_3A_3A_2 / aB_3A_1A_3A_3A_2 / bA_3A_2$

UNIT - IV TURING MACHINE :

Definitions of Turing Machine - Models - Computable Languages & Function - Techniques For Turing Machine - Construction - Multi head & Multi Tape Turing Machine - The Halting problem - Partial solvability - Problems about Turing Machine - Chomskian Hierarchy of Languages.

INTRODUCTION - TURING MACHINE (TM):

- During the year 1936, Alan Turing introduced a new mathematical model called Turing Machine
- Turing Machine is an abstract machine (or) mathematical model to represent a real computer.
- Turing Machine is a tool, for studying the computability of mathematical function.
- Turing Hypothesis believed that a function is computable if and only if it can be computed by turing machine.
- Turing machine can solve any problem that a modern computer can solve.
- Turing machine is used to define the language and to compile the integer functions.
- Turing machine accepts recursive language or recursive enumerable language.
- Turing machine differs from PDA and FA.
- FA has finite memory and PDA has infinite memory and access in LIFO order
- But TM has both infinite memory and no restriction in accessing the input.

• TM has infinite tape memory & the tape head can move either left or right to access the input

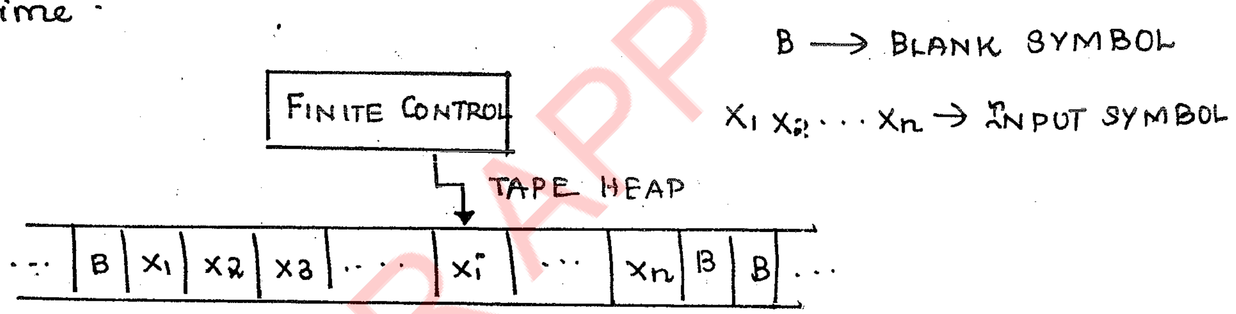
MODEL OF TURING MACHINE:

Turing Machine has

1. Finite control - which contains set of states and transitions between the states.

2. Turing Machine has an input tape (i.e) divided into cells & each cell can hold any one of the finite number of symbols over alphabet.

3. It has a tape head that scans one cell on the input tape at a time.



INPUT TAPE

WORKING OF TURING MACHINE:

• The Turing Machine, the input initially consists of a finite length string of symbols chosen from the I/p alphabet & the i/p is placed on the input tape.

• All other tape cells extending infinitely into the left & right of the input tape contains the spirial symbol called "Blank symbol"

• The tape head is positioned at one of the tape cells for scanning the input symbol from the input tape.

• Initially the tape head points at the left most cell of the input tape

FORMAL NOTATION / DEFINITION OF A TURING MACHINE:

Turing Machine has 7-tuple:

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

$Q \rightarrow$ The Finite set of states of the Finite control.

$\Sigma \rightarrow$ The Finite set of input symbols.

$\Gamma \rightarrow$ The complete set of tape symbols, Σ is always a subset of Γ.

$(\Sigma \subseteq \Gamma)$

$\delta \rightarrow$ The Transition Function $\boxed{\delta(q, x) = (P, Y, D)}$.

where $q \rightarrow$ a state, $x \rightarrow$ a tape symbol, $P \rightarrow$ new state / same state in Q, $Y \rightarrow$ symbol in Γ, written in the cell being scanned, replacing whatever symbol was there.

$D \rightarrow$ Direction, either Left or Right and telling us the direction in which the head moves.

$q_0 \rightarrow$ The start state, a member in Q, in which the Finite control is found initially.

$B \rightarrow$ The blank symbol. This symbol is in Γ but not in Σ

$F \rightarrow$ The set of Final / Accepting states i.e $F \subseteq Q$.

PROCESSING OF MOVE IN A TURING MACHINE:

- The single move of a Turing Machine depends on the current state of Finite control and the tape symbol present in the input tape.
- The Following changes happen in one ~~sta~~ move of a TM.

$\rightarrow$ Changes the state after consuming an i/p symbol. It may also be in the same state or transfer to any new state

$\rightarrow$ The Tape symbol to be replaced for the scanned i/p tape symbol

→ Deciding the move of the tape head to left or right of i/p tape

→ Whether to halt the TM or not.

INSTANTANEOUS DESCRIPTIONS OF A TM: (ID).

- The execution sequence of an i/p string is represented by the ID of a TM.
- Each move of TM is represented by the ID.
- ID of a TM describes the current configuration and it can be of following types,

✓	Accepting configuration
✓	Rejecting Configuration

- A move of TM can be represented as a pair of ID separated by the symbol $\vdash$:
- Each move is represented by $\alpha_1 q \alpha_2$ where
  α_1 & α_2 are the strings from T^* and q is the state of TM
- The move can be of single move or zero or more moves as

$$\vdash_m = \text{single move} \quad \vdash_m^* = \text{zero / more moves}$$

Let us use the string

$\underline{x_1 x_2 \ldots x_{i-1}\, q\, x_i\, x_{i+1} \ldots x_n}$ to represent ID.

where 1. q is the state of TM.

2. The Tape head is scanning the ith symbol from left.

3. $x_1 x_2 \ldots x_n$ is the portion of the tape between the leftmost & rightmost non-Blank.

If the transition function of TM is

CASE 1: $\boxed{\delta(q, x_i) = (P, Y, L)}$

i.e. the next move is leftward. Then

$$x_1 x_2 \ldots x_{i-1}\, q\, x_i\, x_{i+1} \ldots x_n \vdash_m x_1\, x_2 \ldots x_{i-2}\, P\, x_{i-1}\, Y\, x_{i+1} \ldots x_n$$

NOTE: This move reflects the change to state P and the fact that the tape head is now positioned at cell $i-1$.

There are 2 important exceptions

1. If $i=1$, then M moves to the blank to the left of $x_1 \cdots x_n$. In that case, $x_1 x_2 \cdots x_{i-1} q x_i x_{i+1} \cdots x_n \vdash_m PBYx_2 \cdots x_n$.
2. If $i=n$, then the symbol B written over x_n joins the infinite sequence of trailing blanks and doesn't appear in next ID.

$$x_1 x_2 \cdots x_{n-1} q x_i \cdots x_n \vdash_m x_1 x_2 \cdots x_{n-2} P x_{n-1} Y$$

CASE 2: $\boxed{\delta(q, x_i) = (P, Y, R)}$ i.e, the next move is Rightward, then

$x_1 x_2 \cdots x_{i-1} q x_i x_{i+1} \cdots x_n \vdash_m x_1 x_2 \cdots x_{i-1} Y P x_{i+1}$. Here the move reflects the fact that the head is $\cdots x_n$ moved to cell etc.

AGAIN THERE ARE 2 IMPORTANT EXCEPTIONS:

1. If $i=n$, then the $i+1^{st}$ cell holds a blank and that cell was not part of the previous ID. Thus we insert,

$$x_1 x_2 \cdots x_{i-1} q x_i x_{i+1} \cdots x_n \vdash_m x_1 x_2 \cdots x_{n-1} YPB$$

2. If $i=1$ & $Y=B$, then the symbol B written over x_1 joins the infinite sequence of leading blanks & doesn't appear in next ID i.e $x_1 x_2 \cdots q x_i \cdots x_n \vdash_m YPx_2 \cdots x_n$.

LANGUAGE OF A TM:

- The set of languages accepted by TM is recursively enumerable language.
- The input string is placed on the input tape & the tape head begins at the leftmost input symbol

If the TM enters an accepting state, then i/p is accepted else the i/p string is not accepted.

The languages accepted by TM M is defined as L(M) and it is denoted by $L(M) = \{ w \mid w \text{ is in } \Sigma^* \; q_0 w \vdash_{M}^{*} \alpha_1 P \alpha_2$ for some state P in F & α_1 and α_2 is in $\Gamma^* \}$.

HALTING OF TM:

- There is another notion of "acceptance" i.e commonly used for TM: acceptance by halting.
- We say a TM halts if it enters a state q, scanning a i/p tape symbol X, and there is no move in this situation (i.e) $\delta(q, X)$ is undefined.
- TM always halts, when it is an accepting state. Unfortunately, it is not always possible to require that a TM halts even if it doesn't accept.
- Those lang with TM that donot halt eventually, regardless of whether or not they accept are called recursive.
- TM that always halt, regardless of whether or not they accept, are a good model of an "algorithm". If an algorithm to solve a given problem exists, then we say the problem is "decidable". So TM's that always halt.

COMPUTABLE LANGUAGE AND FUNCTIONS:

DESIGN A TM FOR COMPUTABLE FUNCTIONS.

PROBLEMS:

1. DESIGN a TM to process zero function such that $f(x) = 0$. where x is input.

SOLUTION:

STEP 1: IDEA OF CREATION:

The idea to design this TM is that x is the i/p, if $x=5$, then i/p tape contains 5 no. of 1's in the input and steps are as follows.

(i) The TM initially in the state q_0 and if it reads '1' as the left most symbol, it replaces '1' to 'B' & moves to right without changing the state.

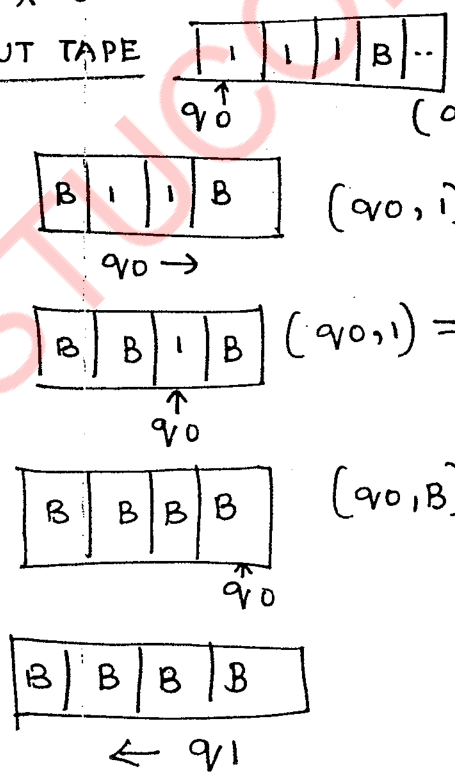
(ii) The TM remains in the same state q_0 and replaces all 1's to 'B' until it sees 'B'.

(iii) At state q_0, if it finds 'B' it enters the final state q_1, then halt the TM.

STEP 2: DIAGRAMMATIC REPRESENTATION.

EXAMPLE $x = 3$.

INPUT TAPE



$(q_0, 1) = (q_0, B, R)$.

$(q_0, 1) = (q_0, B, R)$

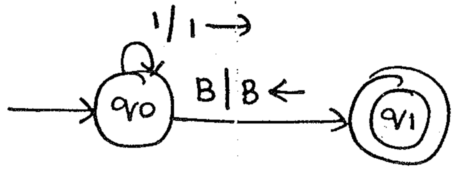
$(q_0, 1) = (q_0, B, R)$.

$(q_0, B) = (q_1, B, L)$ halts.

STEP 3 : TRANSITION TABLE

STATE	1	B
$\rightarrow q_0$	(q_0, B, R)	(q_1, B, L)
$* q_1$	–	–

STEP 4 : TRANSITION DIAGRAM.



TM FOR $f(x) = 0$.

STEP 5 : TM DEFINITION IS $M = (\{q_0, q_1\}, \{1\}, \{1, B\}, \delta, q_0, B, \{q_1\})$

δ : $\delta(q_0, 1) = (q_0, B, R)$

$\delta(q_0, B) = (q_1, B, L)$

STEP 6 : INSTANTANEOUS DESCRIPTION :

EXAMPLE $x = 2$ $\delta(q_0, 11B) \vdash_m (Bq_0 1B) \vdash_m (BBq_0 B) \vdash_m (Bq_1 BB)$

String accepted and all 1's changed to Blank and the zero functio[n] is implemented.

2. Design a TM to implement the Function $f(n) = x + 1$.

SOLUTION : If $x = 3$ then

Input Tape

1	1	1	B	...

output Tape

1	1	1	1	B	...

STEP 1 :

1. TM is initially in the state q_0 and it reads '1' in the leftmost input tape.

2. At state q_0 when it reads '1' it remains in the same state, without changing '1' and just move the tape head to righ[t]

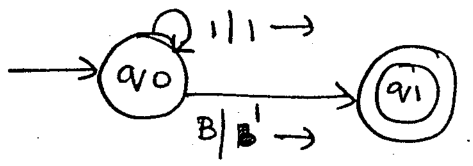
3. At state q_0, it skips all 1's and searches for the 1st blank symbol B.

4. At state q_0, when it finds 1st 'B', it enters the final state q_1 & changes 'B' to '1'.

STEP 2: TRANSITION TABLE

	1	B
$\rightarrow q_0$	$(q_0, 1, R)$	$(q_1, 1, R)$
$* q_1$	–	–

STEP 3: TRANSITION DIAGRAM



TM for $f(x) = x+1$.

STEP 4: TM Definition $M = (\{q_0, q_1\}, \{1\}, \{1, B\}, \delta, q_0, B, \{q_1\})$

δ: $\delta(q_0, 1) = (q_0, 1, R)$

$\delta(q_0, B) = (q_1, 1, R)$.

STEP 4: INSTANTANEOUS DESCRIPTION: $x = 3$

$\delta(q_0, 111B) \vdash_m (1q_011B) \vdash_m (11q_01B) \vdash_m (111q_0B) \vdash_m (1111q_1B)$

String is accepted.

3. Design a TM to implement the function $f(x) = x+2$.

SOLUTION: EXAMPLE: $X=3$

Input tape:

1	1	1	B	...

output tape:

1	1	1	1	1	B	..

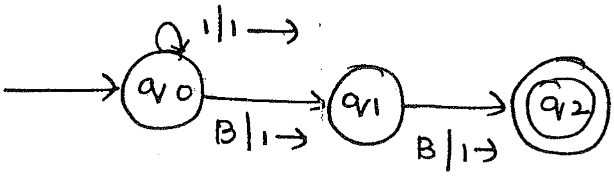
STEP 1:

1. At state q_0, the initial state of TM, it reads the leftmost 1, it skips 1 and searches for the 1st Blank symbol 'B' and moves to right.
2. At state q_0, when it reads 1st B, it changes B to '1' and moves to right to see the next Blank symbol 'B' and changes to 'q_1'
3. At state q_1, when it finds the 2nd 'B' blank symbol, it changes B to '1' and moves to right and enters the accepting state q_2.

STEP 2: TRANSITION TABLE:

	1	B
$\rightarrow q_0$	$(q_0, 1, R)$	$(q_1, 1, R)$
q_1	–	$(q_2, 1, R)$
$* q_2$	–	–

STEP 3: TRANSITION DIAGRAM.



TM for $f(x) = x+2$.

STEP 4: TM definition $M = (\{q_0, q_1, q_2\}, \{1\}, \{1, B\}, \delta, q_0, B, \{q_2\})$

δ: $\delta(q_0, 1) = (q_0, 1, R)$

$\delta(q_0, B) = (q_1, 1, R)$

$\delta(q_1, B) = (q_2, 1, R)$.

STEP 5: ID $x = 3$.

$\delta(q_0, 111B) \vdash_m (q_0 111B) \vdash_m (1q_0 11B) \vdash_m (11q_0 1B) \vdash_m (111q_0 B) \vdash_m$

$(1111q_1 B) \vdash_m (11111q_2 B)$

String is accepted.

4. Design a TM to implement the concatenation function $f(x, y) = xy$ (or) to implement addition function $f(x, y) = x+y$

SOLUTION:

STEP 1:

Let us assume that x is represented by the 1^x and y is represented by 1^y in the input tape. The 1^x and 1^y is separated by the separator symbol '#' and is shown below.

$x = 2 \quad y = 3$

Input :

1	1	#	1	1	1	B	...

output :

1	1	1	1	1	B	...

$x+y = 2+3 = 5$

The sum of 2 values are performed by replacing the last '1' by Blank symbol and the steps are as follows:

a. At initial state q_0, when it reads '1', it skips the 1's and remain in the same state.

b. At state q_0, when it reads '#' it reaches the state q_1 and changes '#' to '1' and moves right

c. At state q_1, it skips all 1's and searches for 'B' by moving righ

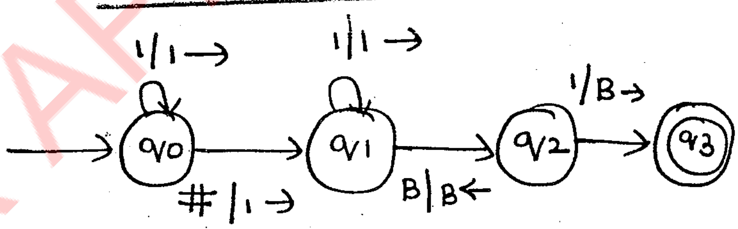
d. At state q_1, when it sees blank symbol, it moves left and changes state to q_2.

e. At state q_2, when it finds '1' it replaces '1' to B and enters the Final state q_3.

STEP 2: TRANSITION TABLE

State	1	#	B
$\rightarrow q_0$	$(q_0, 1, R)$	$(q_1, 1, R)$	-
q_1	$(q_1, 1, R)$	-	(q_2, B, L)
q_2	(q_3, B, R)	-	-
$* q_3$	-	-	-

STEP 3: TRANSITION DIAGRAM.



TM for $f(x,y) = x+y$.

STEP 4: TM definition $M = (\{q_0, q_1, q_2, q_3\}, \{1\}, \{1, \#, B\}, \delta, q_0, B, \{q_3\})$

STEP 5: ID EXAMPLE $x=2$ $y=3$

$\delta(q_0, 11\#111B) \vdash_m (q_0 11\#111B) \vdash_m (1 q_0 1\#111B) \vdash_m (11 q_0 \#111B)$

$\vdash_m (111 q_1 111 B) \vdash_m (1111 q_1 11B) \vdash_m (11111 q_1 1B) \vdash_m (111111 q_1 B)$

$\vdash_m (11111 q_2 1) \vdash_m (11111 B q_3 B)$

String accepted. The function $f(x,y) = x+y$ is implemented.

5. Design a TM to perform subtraction $f(x,y) = \begin{cases} x-y & \text{if } x>y \\ 0 & \text{if } x \le y \end{cases}$

SOLUTION:

The idea to create a TM to perform subtraction is, the i/p is represented as $1^m \# 1^n$. The value 1^m and 1^n is separated by a separator symbol '#' and $1^m \# 1^n$ is surrounded by B.

This proper subtraction function say that

$$f(m,n) = \begin{cases} m-n, & \text{if } m>n \\ 0, & \text{if } m \le n \end{cases}$$

So we have to design a TM such that if $m>n$ the subtracted value that is $1^m - 1^n$ should be on the tape. And if $m \le n$, then tape should have only 'B'

If $m=4$, $n=2$ (i.e) $m>2$

Input:

1	1	1	1	#	1	1	B	...

output:

B	B	1	1	B	B	...

$m-n=2$

If $m=2$, $n=4$, $m \le n$.

Input:

1	1	#	1	1	1	1	B	...

output:

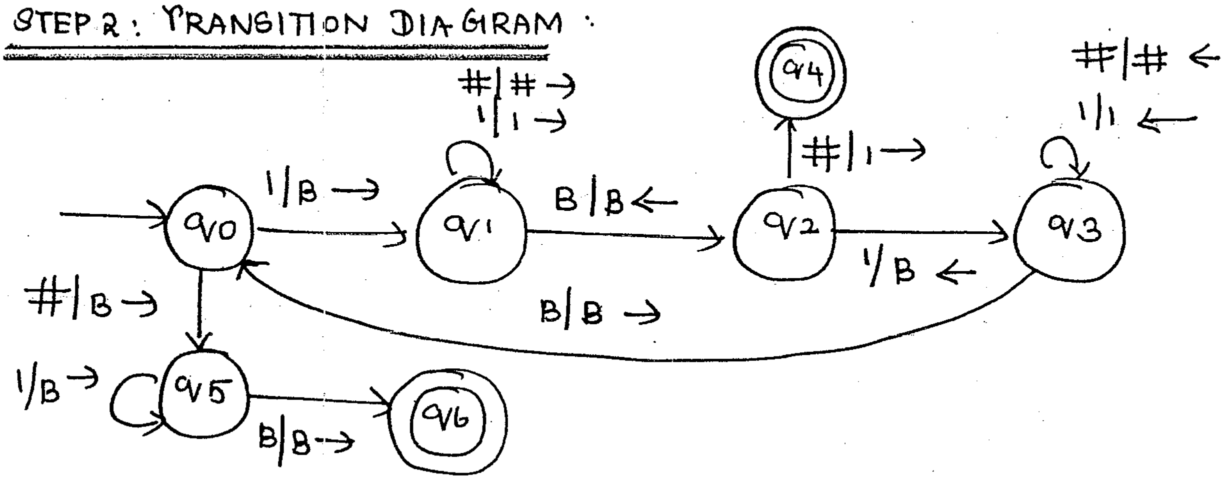
B	B	B	B	...

$m-n=0$

- The idea to design this TM is that the TM process in such a way that for each '1' on the leftmost side, it replaces '1' on the right most side to 'B'. ['1' appearing before 'B']
- After replacing with 1's to the left and right when the m/c encounters separator symbol on right side, it is clear that n value ends.
- When 'n' value ends, it starts replacing '#' to '1' and enters final | accepting state.
- Similarly if $m \le n$, then m/c encounters the symbol '#'

from initial state then it starts replacing all 1's and '#' to Blank and enter the Final state.

STEP 2: TRANSITION DIAGRAM:



STEP 3: TRANSITION TABLE:

	1	#	B
→q_0	(q_1, B, R)	(q_5, B, R)	–
q_1	$(q_1, 1, R)$	$(q_1, \#, R)$	(q_2, B, L)
q_2	(q_3, B, L)	$(q_4, 1, R)$	–
q_3	$(q_3, 1, L)$	$(q_3, \#, L)$	(q_0, B, R)
* q_4	–	–	–
q_5	(q_5, B, R)	–	(q_6, B, R)
* q_6	–	–	–

STEP 4: TM definition $M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{1\}, \{1, \#, B\}, \delta, q_0, B, \{q_4, q_6\})$

STEP 5: ID: $m = 2 \quad n = 1$

$\delta(q_0, 11\#1B) \vdash_m (q_0 11\#1B) \vdash_m (Bq_1 1\#1B) \vdash_m (B1q_1\#1B) \vdash_m (B1\#q_1 1B) \vdash_m (B1\#1q_1B) \vdash_m (B1\#q_3B) \vdash_m (B1q_3\#B) \vdash_m (Bq_3 1\#B) \vdash_m (q_3B1\#B) \vdash_m (Bq_0 1\#B) \vdash_m (BBq_1\#B) \vdash_m (BB\#q_1B) \vdash_m (BBq_2\#B)$

$\vdash (BB1q_4 B)$

String accepted and now the input tape contain one 1's and the function $f(m-n) = m-n$ is implemented.

Eg:2 $m=1, n=2$.

$\delta(q_0, 1\#11B) \vdash_m (q_0 1\#11B) \vdash_m (Bq_1\#11B) \vdash_m (B\#q_1 11B) \vdash_m (B\#1q_1 1$

$\vdash_m (B\#11q_1 B) \vdash_m (B\#1q_2 1B) \vdash_m (B\#q_3 1B) \vdash_m (Bq_3\#1B)$

$\vdash_m (q_3 B\#1B) \vdash_m (Bq_0\#1B) \vdash_m (BBq_5 1B) \vdash_m (BBBq_5 B) \vdash_m (BBBBq_6$

String accepted. Since m is less than n, then the i/p tape contains zero value.

6. Design a TM to implement multiplication function $f(x,y) = x*y$.

STEP 1:

The idea to design this TM is that we place the input as $1^x\#1^y\#$ on the TM. Now the multiplication is done by performing successive addition and it is shown below.

$x=2 \quad Y=3$

Input:

1	1	#	1	1	1	#	B	..

$x=2 \quad y=3$

$x*y = 2*3 = 6.$

output:

B	B	B	B	B	1	1	1	1	1	1	B	..

STEPS:

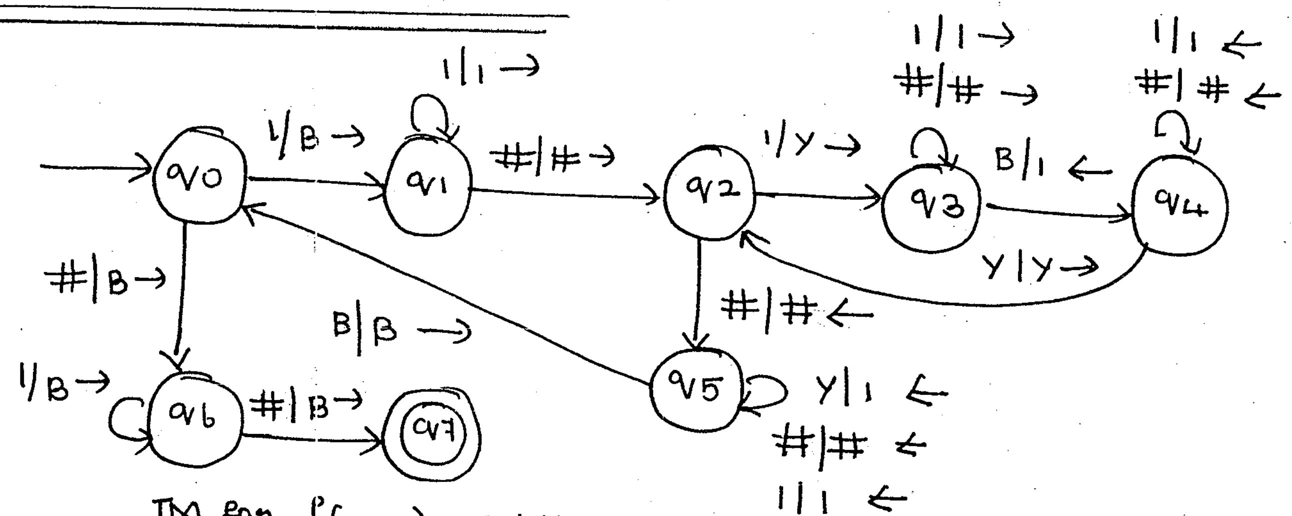
a. At initial state when '1' finds in the i/p, replace it to 'B' and move right for searching #

b. After finding '#', copy the 'Y' no. of 1's for 'x' no. of times in B symbols

c. After performing 'x' no. of copy with 'y' no. of 1's we replac $\#1^y\#$ to 'B' then reach to final state and the tape contains

1^{xy}.

STEP 2: TRANSITION DIAGRAM:



TM for $f(x,y) = x * y$.

STEP 3: Transition Table.

states	1	#	B	y
→ q0	(q1, B, R)	(q6, B, R)	–	–
q1	(q1, 1, R)	(q2, #, R)	–	–
q2	(q3, y, R)	(q5, #, L)	–	–
q3	(q3, 1, R)	(q3, #, R)	(q4, 1, L)	–
q4	(q4, 1, L)	(q4, #, L)	–	(q2, y, R)
q5	(q5, 1, L)	(q5, #, L)	(q0, B, R)	(q5, 1, L)
q6	(q6, B, R)	(q7, B, R)	–	–
q7	–	–	–	–

STEP 4: INSTANTANEOUS DESCRIPTION: $x = 2, y = 1$.

$\delta(q_0, 11\#1\#B) \vdash_m (q_0 11\#1\#B) \vdash_m (Bq_1 1\#1\#B) \vdash_m (B1q_1\#1\#B)$

$\vdash_m (B1\#q_2 1\#B) \vdash_m (B1\#y q_3 \#B) \vdash_m (B1\#y\# q_3 B) \vdash_m (B1\#y q_4 \#1$

$\vdash_m (B1\#q_4 y\#1) \vdash_m (B1\#y q_2 \#1) \vdash_m (B1\#q_5 y\#1) \vdash_m (B1q_5\#1\#1)$

$\vdash_m (Bq_5 1\#1\#1) \vdash_m (q_5 B1\#1\#1) \vdash_m (Bq_0 1\#1\#1) \vdash_m (BBq_1\#1\#1)$

$\vdash_m (BB\#q_2 1\#) \vdash_m (BB\#y q_3 \#1B) \vdash_m (BB\#y\# q_3 1B)$

$\vdash_m (BB\#Y\#1q_3B) \vdash_m (BB\#Y\#q_4 11) \vdash_m (BB\#y\, q_4\# 11)$
$\vdash_m (BB\#q_4 y\#11) \vdash_m (BB\#\, yq_2\#11) \vdash_m (BB\#q_5 y\#11) \vdash_m (BBq_5\#\#11)$
$\vdash_m (Bq_5 B\#1\#11) \vdash_m (BBq_0\#1\#11) \vdash_m (BBBq_6 1\#11) \vdash_m$
$(BBBBq_6\#11) \vdash_m (BBBBBq_2 11)$

String is accepted and the $f(x,y) = f()\ x * y$ is implemented.

7. Design a TM to perform 1's complement of a no. over $\Sigma = \{0,1\}$.

SOLUTION:

On Reading the i/p;

→ If the symbol = 0 replaces it by '1' & move right

→ If the symbol = 1 replace it by '0' & move right

→ Perform step 1 & 2 until the i/p symbols are processed from left to right

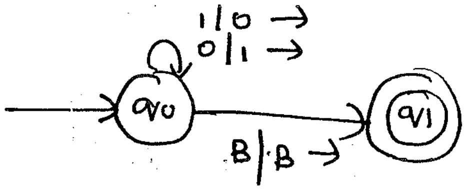
→ Halt the m/c when it encounters the 1st Blank symbol.

Example: 1011 → 0100
i/p   o/p

STEP 2: TRANSITION TABLE:

	0	1	B
$\to q_0$	$(q_0,1,R)$	$(q_0,0,R)$	(q_1,B,R)
$* q_1$	–	–	–

STEP 3: TRANSITION DIAGRAM.



STEP 4: TM Definition $M = (\{q_0,q_1\}, \{0,1\}, \{0,1,B\}, \delta, q_0, B, \{q_1\})$

STEP 5: Ip. $w = 101$

$\delta(q_0, 101B) \vdash_m (q_0 101B) \vdash_m (0q_0 01B) \vdash_m (01q_0 1B) \vdash_m (010q_0 B)$
$\vdash_m (010Bq_1)$

String accepted and 1's complement is implemented.

8. Design a TM to perform 2's complement of a no over $\Sigma = \{0,1\}$.

NOTE: Don't change the bits from the right towards left until the 1st 1 has been processed perform complementation to the rest of the bits from right to left [after 1st 1 is processed]

SOLUTION:

STEP 1:

a. Traverse Right & Locate Right most bit.

b. If the bit = 0, perform no replaces & move left.

c. If the bit = 1, perform no change & move left.

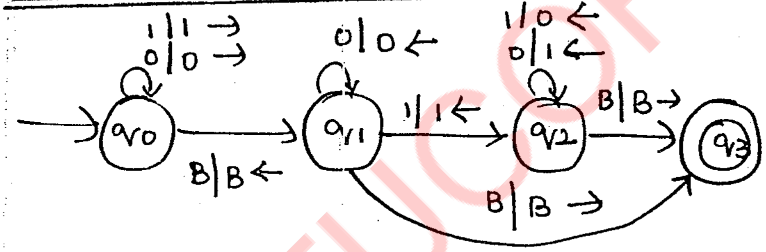
d. If the next bit symbol = '0' replace it by '1' and move left.

e. Else if the next bit symbol = '1' replace it by '0' & move left.

f. Perform steps until all the i/p symbols are processed [From Right to Left]

g. Halt the m/c.

STEP 2:- TRANSITION DIAGRAM:



STEP 3: TRANSITION TABLE

	0	1	B
→ q0	(q0,0,R)	(q0,1,R)	(q1,B
q1	(q1,0,L)	(q2,1,L)	(q3,B,(
q2	(q2,1,L)	(q2,0,L)	(q3,B,(
* q3	—	—	—

STEP 4: TM Definition:

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

STEP 5: ID. $w = 101$.

$\delta(q_0, 101B) \vdash_m (q_0 101B) \vdash_m (1 q_0 01B) \vdash_m (10 q_0 1B) \vdash_m (101 q_0 B)$

$\vdash_m (10 q_1 1B) \vdash_m (1 q_2 01B) \vdash_m (q_2 111B) \vdash_m (q_2 B011B) \vdash_m (B q_3 011B)$

String is accepted and function is implemented.

COMPUTABLE LANGUAGE

1. Design a TM that accepts the language $L = \{a^n b^n \mid n \geq 1\}$.

SOLUTION:

STEP 1: IDEA OF CREATION:

a. The idea to create this TM is to place $a^n b^n$ in the i/p tape

b. Let the TM initially be in the state q_0 (initial state).

c. while in q_0, the machine reads 'a' and changes ~~to~~ 'a' to X and moves to the right and changes its state to q_1 and starts scanning the next input

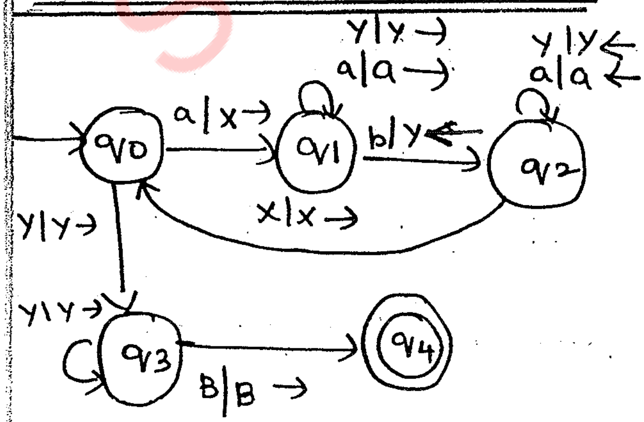
d. From the q_1, while reading 'a' it doesnot change state but simply moves to the right until seeing 1st 'b'

e. When seeing 'b' from state q_1, it reach the state q_2 and change 'b' to 'y' and moves to left to see 'X'

f. The From state q_2 when it sees X, it the state to q_0 and repeat the process.

g. The major idea is that for each 'a', we try to 'b' and alternatively, the process is repeated

STEP 2: TRANSITION DIAGRAM.



REJECTING STATE.

$(q_3, b) = (q_{reject}, b, R)$ [b>a]

$(q_3, a) = (q_{reject}, a, R)$ [ba]

$(q_3, b) = (q_{reject}, B, R)$ [a>b]

STEP 3: TRANSITION TABLE.

	a	b	Y	X	B
$\rightarrow q_0$	(q_1, X, R)	–	(q_3, Y, R)	–	–
q_1	(q_1, a, R)	(q_2, Y, L)	(q_1, Y, R)	–	–
q_2	(q_2, a, L)	–	(q_2, Y, L)	(q_0, X, R)	–
q_3	–	–	(q_3, Y, R)	–	(q_4, B, R)
$*\ q_4$	–	–	–	–	–

STEP 4: TM definition $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, \{a, b, X, Y, B\}, \delta, q_0, B, \{q_4\})$

STEP 5: ID $w_1 = aabb$.

$\delta(q_0, aabb) \vdash_m (q_0 aabb) \vdash_m (X q_1 abb) \vdash_m (X a q_1 bb) \vdash_m (X q_2 a Y b)$
$\vdash_m (q_2 X a Y b) \vdash_m (X q_0 a Y b) \vdash_m (X X q_1 Y b) \vdash_m (X X Y q_1 b) \vdash_m (X X q_2 Y Y)$
$\vdash_m (X q_2 X Y Y) \vdash_m (X X q_0 Y Y) \vdash_m (X X Y q_3 Y) \vdash_m (X X Y Y q_3 B) \vdash_m (X X Y Y B q_4)$

String "aabb" is accepted.

ID $w_2 = aab$.

$\delta(q_0, aab) \vdash_m (q_0 aab) \vdash_m (X q_1 ab) \vdash_m (X a q_1 b) \vdash_m (X q_2 a Y)$
$\vdash_m (q_2 X a Y) \vdash_m (q_0 a Y) \vdash_m (X X q_1 Y) \vdash_m (X X Y q_1 B)$

String "aab" is rejected.

2. Design a TM that accepts the language $L = \{a^n b^n c^n \mid n \geq 1\}$.

SOLUTION:

The Construction is similar to the design $a^n b^n$. Here we have to replace each 'a' by 'x' & 'b' by 'y' and 'c' by 'z' respectively.

IDEA :

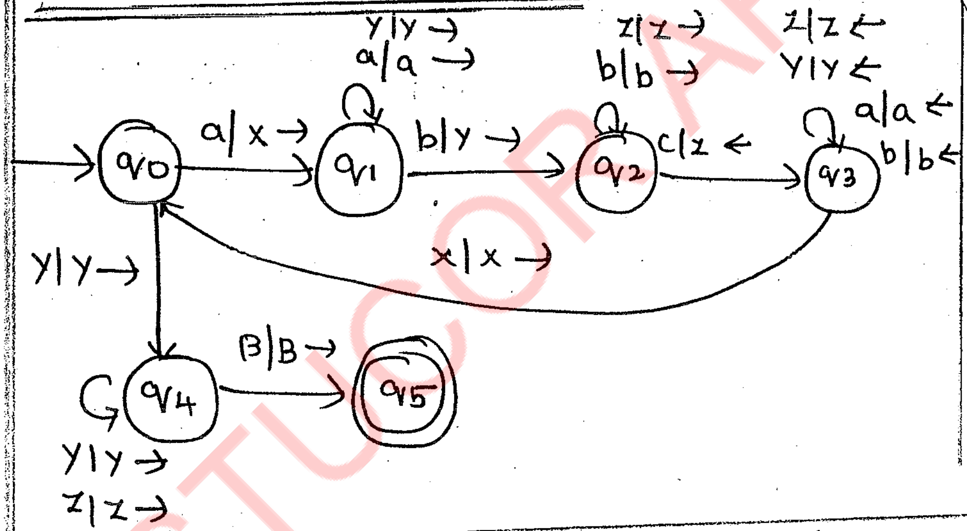
a. Initially the TM is at q_0. At q_0 if it finds a's replace it by x's and move right with state q_1.

b. At q_1, if it finds b's, replace it by y's and moves right with state q_2.

c. At state q_2, if it finds c's replace it by z and enters q_3 by moving left

d. At q_3, if it finds the leftmost x by skipping z by a then it goes to state q_0. Repeat the process till at q_0, if finds y.

STEP 2: TRANSITION DIAGRAM.



REJECTING STATE.

$(q_3, \cancel{b}\,c) = (q_{reject}, \cancel{b}\,c, R)$

$(q_3, a) = (q_{reject}, a, R)$

$(q_3, b) = (q_{reject}, b, R)$

~~(q3, b-~~

STEP 3: TRANSITION TABLE.

	a	b	c	x	y	z	B
q_0	(q_1, Y, R)	–	–	–	(q_4, Y, R)	–	–
q_1	(q_1, a, R)	(q_2, Y, R)	–	–	(q_1, Y, R)	–	–
q_2	–	(q_2, b, R)	(q_3, z, L)	–	–	(q_2, z, L)	–
q_3	(q_3, a, L)	(q_3, b, L)	–	(q_0, x, R)	(q_3, Y, L)	(q_3, z, L)	–
q_4	–	–	–	–	(q_4, Y, R)	(q_4, Y, R)	(q_5, B, R)
q_5	–	–	–	–	–	–	–

STEP 4: TM Definition $M = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{a, b, c\}, \{a, b, c, x, y, z, B\}, \delta, q_0, B, \{q_5\})$.

STEP 5: ID $w_1 = aabbcc$.

$\delta(q_0, aabbcc) \vdash_m (q_0 aabbcc) \vdash_m (x q_1 abbcc) \vdash_m (x a q_1 bbcc)$
$\vdash_m (x a y q_2 bcc) \vdash_m (x a y b q_2 cc) \vdash_m (x a y q_3 b z c) \vdash_m (x a q_3 y b z c)$
$\vdash_m (x q_3 a y b z c) \vdash_m (q_3 x a y b z c) \vdash_m (x q_0 a y b z c) \vdash_m (x x q_1 y b z c)$
$\vdash_m (x x y q_1 b z c) \vdash_m (x x y y q_2 z c) \vdash_m (x x y y z q_2 c) \vdash_m (x x y y q_3 z z)$
$\vdash_m (x x y q_3 y z z) \vdash_m (x x q_3 y y z z) \vdash_m (x q_3 x y y z z) \vdash_m (x x q_0 y y z z)$
$\vdash_m (x x y q_4 y z z) \vdash_m (x x y y q_4 z z) \vdash_m (x x y y z q_4 z)$
$\vdash_m (x x y y z z q_4 B) \vdash_m (x x y y z z B q_5)$

string "aabbcc" is accepted.

3. Design a TM for language L. The set of strings with an equal no. of 0's and 1's.

SOLUTION:

Assume that the i/p string may start with either 0 or 1, but it should have equal no. of 0's and 1's.

For eg 0101, 0110, 1001 . . .

a. Change all 0's to x's and all 1's to y's, whether the i/p may be in any position till reaches the blank symbol.

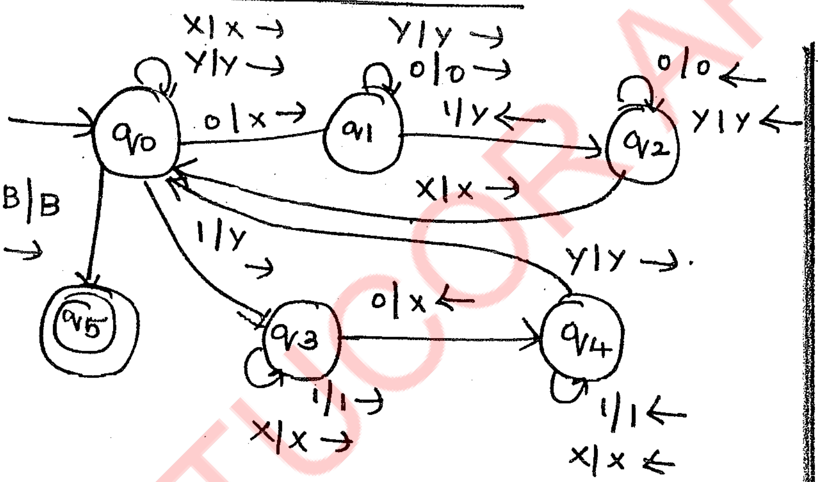
b. Initially, the TM is at state q_0. At q_0, if it finds the leftmost symbol as '0' change it to X and enters q_1

then moves right. If it finds 1 by skipping 0's y's at q_1, change it to y and enters state q_2. At state q_2, the TM searches for the leftmost x by skipping 0's and y's and enters q_0. Repeat the process till the TM finds blank symbol at q_0

c. At q_0, if it finds the leftmost symbol as 1, change it to y and enters state q_3. At q_3, if it finds 0's by skipping 1's and x's, change it to x and enters state q_4 by moving left. At q_4, it searches for the leftmost y. If it finds y at q_4, the TM enters state q_0. Repeat the process till it finds blank symbol.

d. For all other state changes, the input is rejected.

STEP 2: TRANSITION DIAGRAM:



REJECTING STATE.

$(q_3, 1) = (q_{reject}, B, R)$.

$(q_1, B) = (q_{reject}, B, R)$.

STEP 3: TABLE:

	0	1	x	y	B
$\rightarrow q_0$	(q_1, x, R)	(q_3, y, R)	(q_0, x, R)	(q_0, y, R)	(q_5, B, R)
q_1	$(q_1, 0, R)$	(q_2, y, L)	–	(q_1, y, R)	–
q_2	$(q_2, 0, L)$	–	(q_0, x, R)	(q_2, y, L)	–
q_3	(q_4, x, L)	$(q_3, 1, R)$	(q_3, x, R)	–	–
q_4	–	$(q_4, 1, L)$	(q_4, x, L)	(q_0, y, R)	–
q_5	–	–	–	–	–

STEP 4: TM definition $M = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{0,1\},$
$\{0,1,X,Y,B\}, \delta, q_0, B, \{q_5\})$

STEP 5: ID $w_1 = 1001$

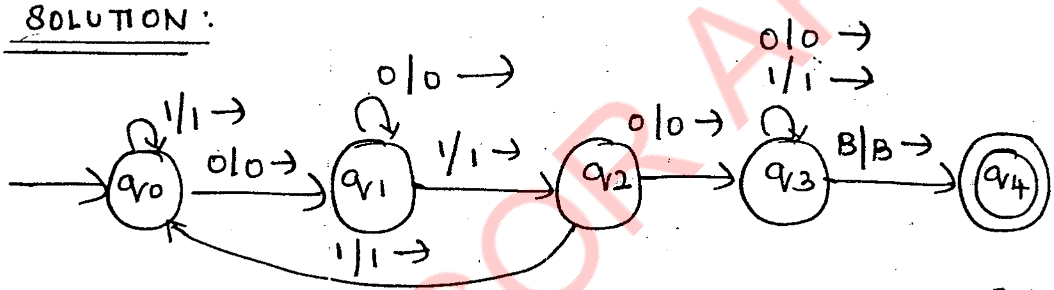
$\delta(q_0, 1001) \vdash_m (q_0 1001) \vdash_m (Y q_3 001) \vdash_m (q_4 Y X 01) \vdash_m (Y q_0 X 01)$
$\vdash_m (Y X q_6 01) \vdash_m (Y X X q_1 1) \vdash_m (Y X q_2 X Y 1) \vdash_m (Y X X q_0 Y) \vdash_m (Y X X Y q_0 B$
$\vdash_m (Y X X Y B q_5) \Rightarrow$ string is accepted.

$w_2 = 0100$.

$\delta(q_0, 0100) \vdash_m (q_0 0100) \vdash_m (q_2 X Y 00) \vdash_m (X q_0 Y 00) \vdash_m$
$(X Y q_0 00) \vdash_m (X Y X q_1 0) \vdash_m (X Y X 0 q_1 B) \Rightarrow$ Rejected [No Transition]

4. Design a TM to accept the language L contains a substring "010"

SOLUTION:



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

5. Design the TM to accept the language of palindromes over the alphabet $\{a,b\}$ or to accept the lang. $L = \{ww^R \mid w \in \{a,b\}\}$.

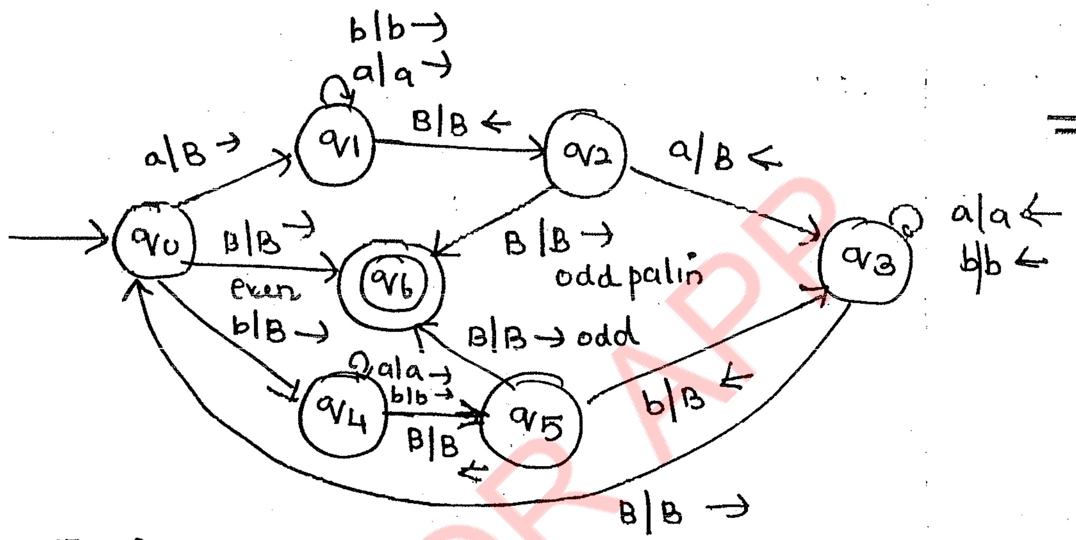
SOLUTION:

STEP 1: IDEA OF CREATION:

- The TM that we are designing now should accept the strings of palindromes such as ababa, abbbba .... The idea to design this TM, is that if the m/c reads 'a' on the left most symbol, replace 'a' to 'B' and move to right and changes last 'a' to B.

- Similarly if the m/c reads 'b' then it replaces b to B and moves to right by searching B and last b and replace b to B.
- So the overall idea is for each 'a' that is first 'a' on the left if matches the last 'a' on the rightmost side and for each b on the 1st time on the left, it matches last b on right si

STEP 3: TRANSITION DIAGRAM.



REJECTING STATE

$\delta(q_2, a) = (q_{reject}, a, L$

$\delta(q_5, b) = (q_{reject}, b, L)$

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

STEP 4: $M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{a,b\}, \{a,b,B\}, \delta, q_0, B, \{q_6\})$.

6. Design the TM to compute the fn $F(w) = w\,cw^R$. where w is any string of a's & b's.

SOLUTION:

STEP 1: IDEA OF CREATION.

- The idea to create this TM is that to read the string w and to create wcw^R.

→ Here we initially read all the symbols in the string w upto 'B' and then moves on the left one position and symbol.

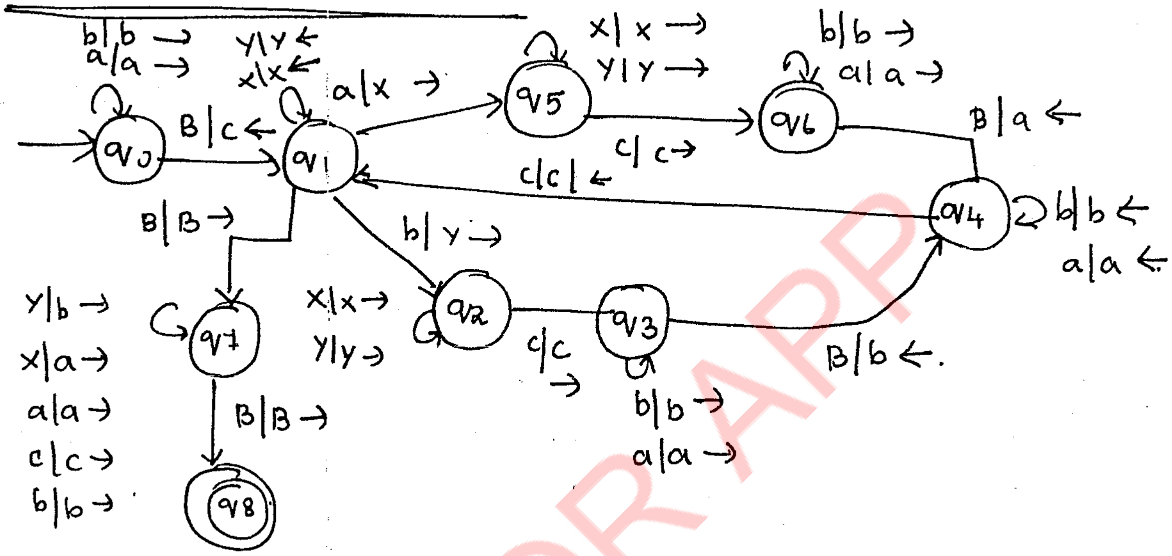
→ If the symbol is 'a', then we replace it by x and if the symbol is 'b', it is replaced by Y.

→ After replacing the symbol, we move to the right and replace B by 'a' or 'b' based on the symbol read before the B.

→ After processing all the strings w and we replace 'x' by 'a' and 'y' by b.

→ After replacing the entire string symbol in 'w', we move to the right side until blank symbol.

STEP 2: Transition Diagram.



Rejecting state

$\delta(q_1, a) = (q_{reject}, a, L)$

$\delta(q_1, b) = (q_{reject}, b, L)$.

STEP 3: TRANSITION TABLE

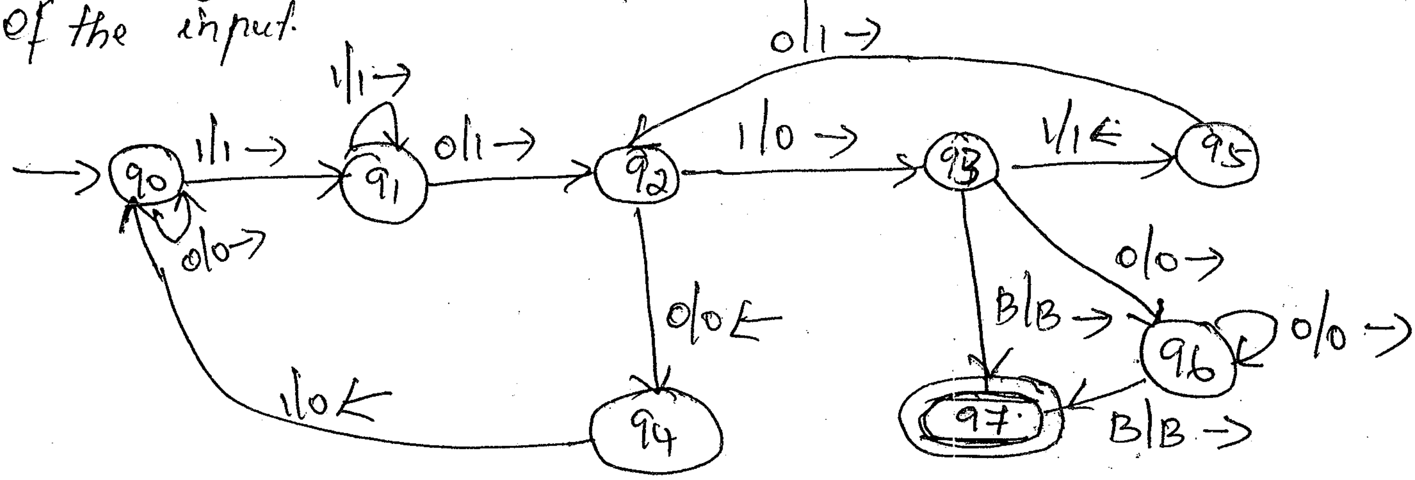
STEP 5: ID – any string.

STEP 4: TM Definition

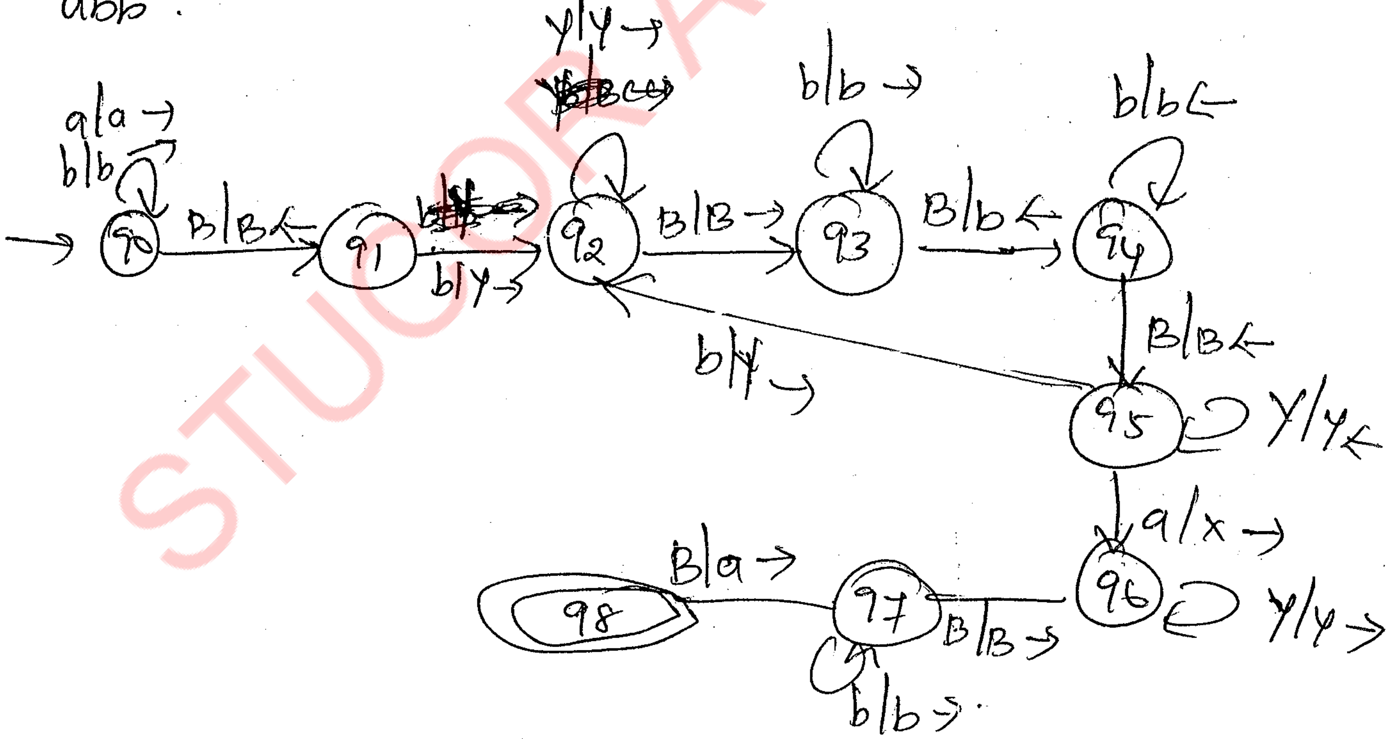
$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}, \{a, b, c\}, \{a, b, c, B\}, \delta, q_0, B, \{q_8\})$

Design a TM which recognizes the input language having a substring as 101 and replaces every occurrence of 101 by 110.

Soln The TM has to be constructed considering 101 as a substring and leaving 110 substring after complete scan of the input.

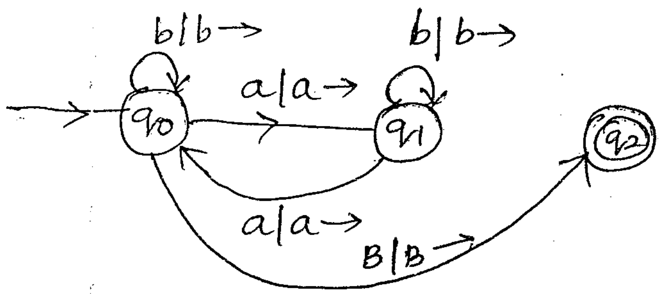


Design a TM which reverse the given string "abb".



Qn: Design the TM to accept the set of all strings over alphabet $\{a, b\}$ with even number of a's.

Solution:

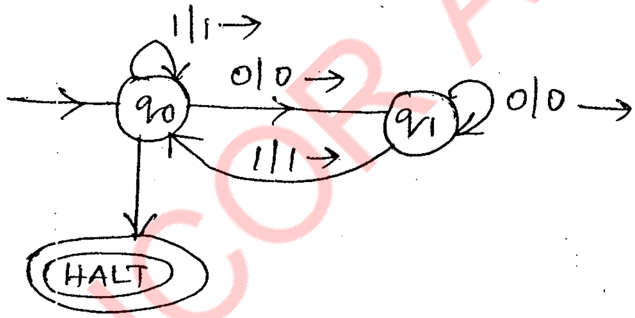


TM $M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, b, B\}, \delta, q_0, B, \{q_2\})$

Qn: Design a TM that accepts the language of odd integers written in binary.

Soln:

Logic: The binary string that ends with 1 is always an odd integer. Hence the TM will be



Techniques for Turing Machine construction

* Here let us see some of the programming techniques that are used to construct an efficient TM that functions as powerful as a conventional computer.

* The different techniques that are used to design a TM are as follows,

(1) storage in finite control

(2) Multiple tracks or multi head TM

(3) Multiple tape (or) multi tape TM

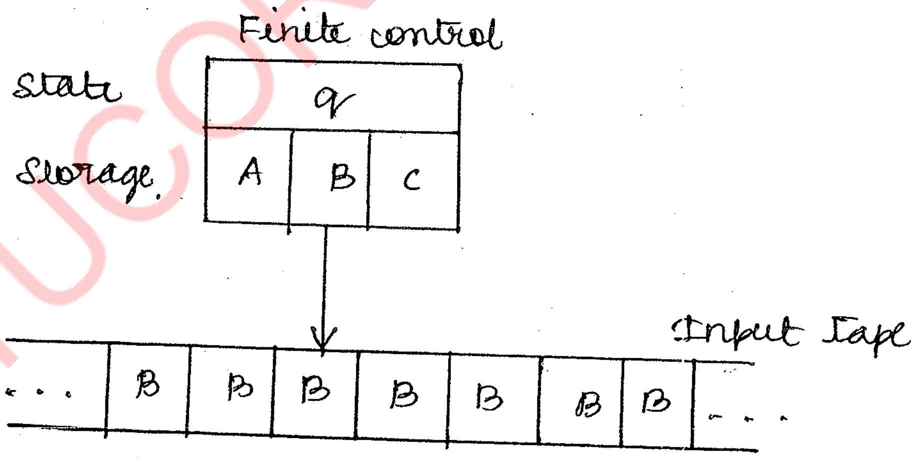
(4) Subroutines

Storage in finite control :

* In TM, generally the finite control contains the FA with the state transitions.

* And the finite control in TM represents the set of states.

* But here in the storage of finite control, we store the data along with the state, so here we use the finite control to hold finite amount of data and it is shown below,



[TM with storage in finite control]

* This type of TM makes the state to remember and to have a memory for the symbol scanned in the input. From the above TM, the state is q and this state q contains A, B, C as the symbol in storage

with q.

* This type of TM can be designed to store in the state with any data from the i/p.

* Each state contain the 'B' blank symbol as its storage initially.

* This type of TM is used to store any symbol in the input and to check whether the stored symbol appears in the input.

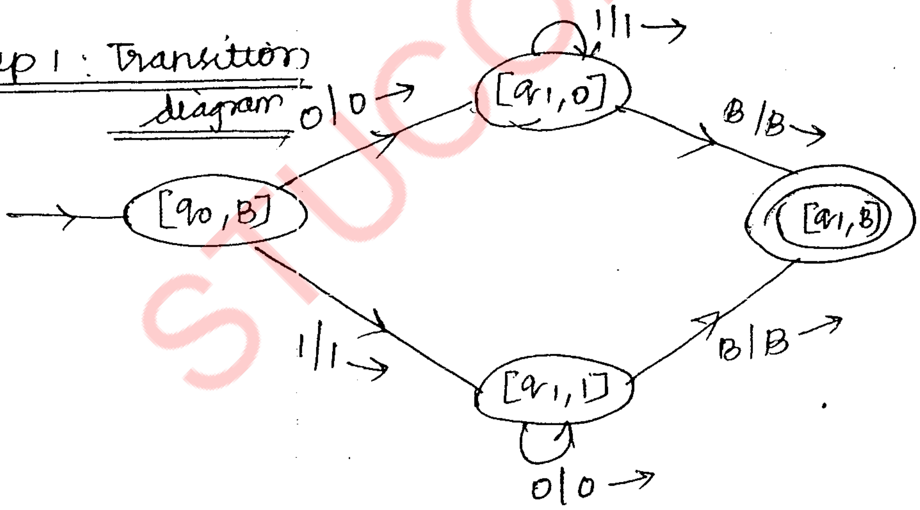
PROBLEMS [For storage in finite control]

Qn: Design a TM to accept the string $01^* + 10^*$.

Soln:

To design a TM, that it should accept the strings such as 01111, 10000 .... etc so the string should have the first symbol as '0' or '1' and it should not appear else where in the input.

Step 1: Transition diagram



Step 2: Transition table

state	0	1	B
$\rightarrow [q_0, B]$	$([q_1, 0], 0, R)$	$([q_1, 1], 1, R)$	–
$[q_1, 0]$	–	$([q_1, 0], 1, R)$	$([q_1, B], B, R)$
$[q_1, 1]$	$([q_1, 1], 0, R)$	–	$([q_1, B], B, R)$
$* [q_1, B]$	–	–	–

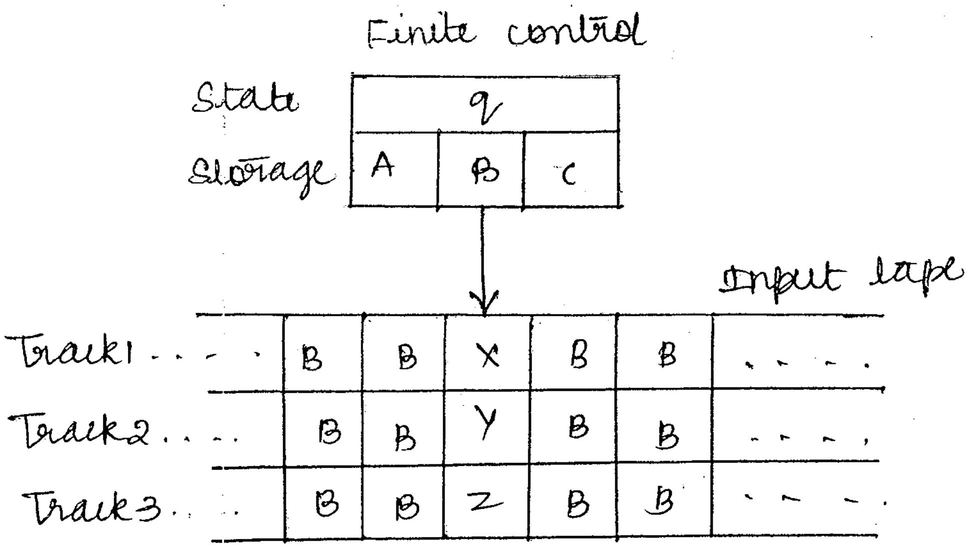
Step 3 : TM definition

$$\text{TM } M = (\{[q_0, B], [q_1, 0], [q_1, 1], [q_1, B]\}, \{0, 1\}, \{0, 1, B\}, \delta, [q_0, B], B, \{[q_1, B]\})$$

Multiple Tracks or Multi Head Turing Machine:

Now we are going to extend this TM to include multiple tracks in the input tape.

- In this TM, where the finite control contains the state and its storage and the input tape contains multiple tracks.
- Each track in the i/p tape contains one symbol.
- The tape alphabet of TM consists of tuples with one component in each track and the number of components in the tuple depends on the number of tracks of the input tape.



• Here, the cell scanned by the tape head contains the symbol [X, Y, Z].

• The multiple tracks of TM is used to find whether the number is odd / even.

• The multiple tracks can be used to check whether the number is prime.

<u>Example</u>: Design a TM using multiple tracks to check whether the given input number is prime or not.

<u>Soln</u>:

* Store the i/p symbol in the 1st track of i/p tape.

* Store the number 2 is binary in the 2nd tracks of i/p tape.

* Copy the i/p in the 3rd Track also.

* All the symbols in the three tracks of the TM are in binary form.

* Now subtract the 2nd Track from third Track until we get '0' or any remainder.

* If the remainder is zero, then the number is not prime, since the prime number is one which is divided by 1 and itself.

* If the remainder is non-zero value, then the 2^{nd} track value is incremented by 1 and again subtraction procedure is continued.

* If the value of the 2^{nd} & 1^{st} track is equal, then the number is prime number. Let us take an i/p value 5 and it is stored as,

Track 1	1	0	1	B	...
Track 2	B	1	0	B	...
Track 3	1	0	1	B	...

Divide the value 2 in 2^{nd} track from value in 3^{rd} track

1	0	1	B	...
B	1	0	B	...
1	0	1	B	...

→

1	0	1	B	..
B	1	0	B	..
0	1	1	B	..

→

1	0	1	B	..
B	1	0	B	..
B	0	1	B	..

The remainder is 1, so increment the value of 2^{nd} track by 1.

1	0	1	B	...
B	1	1	B	...
1	0	1	B	..

→

1	0	1	B	---
B	1	1	B	---
0	1	0	B	---

The remainder is 2, so increment the value of 2^{nd} track

1	0	1	B	...
1	0	0	B	...
1	0	1	B	...

$\longrightarrow$

1	0	1	B	...
1	0	0	B	...
0	0	1	B	...

The remainder is 1, so increment value of 2nd track by 1.

1	0	1	B	...
1	0	1	B	...
1	0	1	B	...

Now the value of first & second track is equal, so the number 5 is a prime number.

Example 2: Input string = '7'

1	1	1	B	...
B	1	0	B	...
1	1	1	B	...

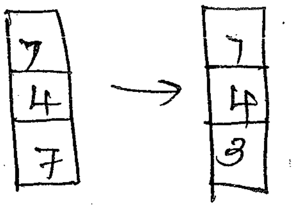
Subtracting 2 from 7, we get.

7	→	7	→	7	→	7
2		2		2		2
7		5		3		1

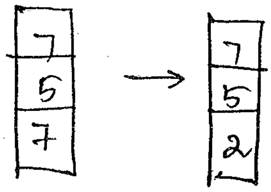
The remainder is 1, so increment the value of 2nd track by 1

7	→	7	→	7
3		3		3
7		4		1

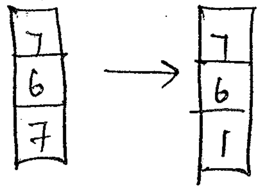
The remainder is 1, so increment the value of 2nd track by 1.



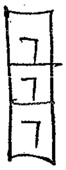
Remainder is 3, so increment value of 2^{nd} track by 1



Remainder is 2, so increment the value of 2^{nd} track by 1.

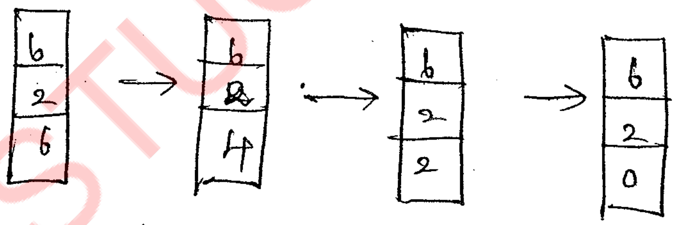


Remainder is 1, so increment value of 2^{nd} track by 1.



Now the value of 1^{st} & 2^{nd} track is equal, so the no. 7 is a prime number.

<u>Example 3:</u> I/P string = 6



On dividing 2 from 6, we get 4 thus by subtracting 4 by 2, we get 2 and again by subtracting 2 by 2 we get 0, since the remainder is 0, the number 6 is not a prime number.

PROBLEMS :

Qn : Build a multitrack turing machine for checking whether given number is prime or not ?

Soln : Here we can build a two track TM. We will consider the input $\Sigma = \{0, 1\}$ is a binary input string. Let n be the number to be checked.

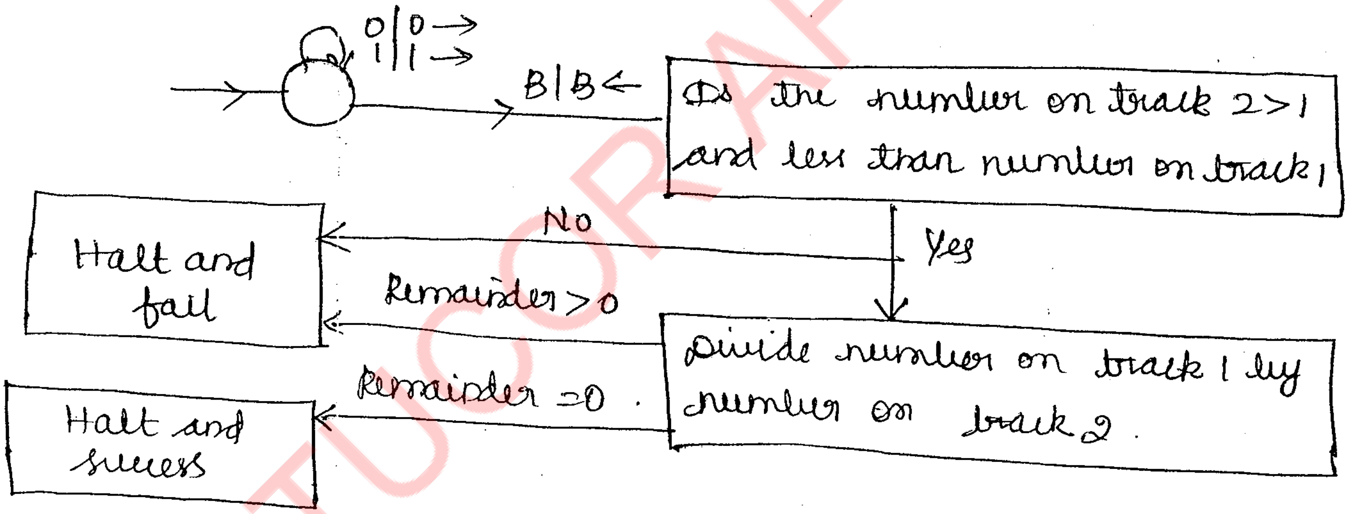
(1) We will guess a number m, where $1 < m < n$.

(2) Divide n by m.

(3) If there is 0 remainder then it halts and succeed.

(4) Otherwise it halts & fails.

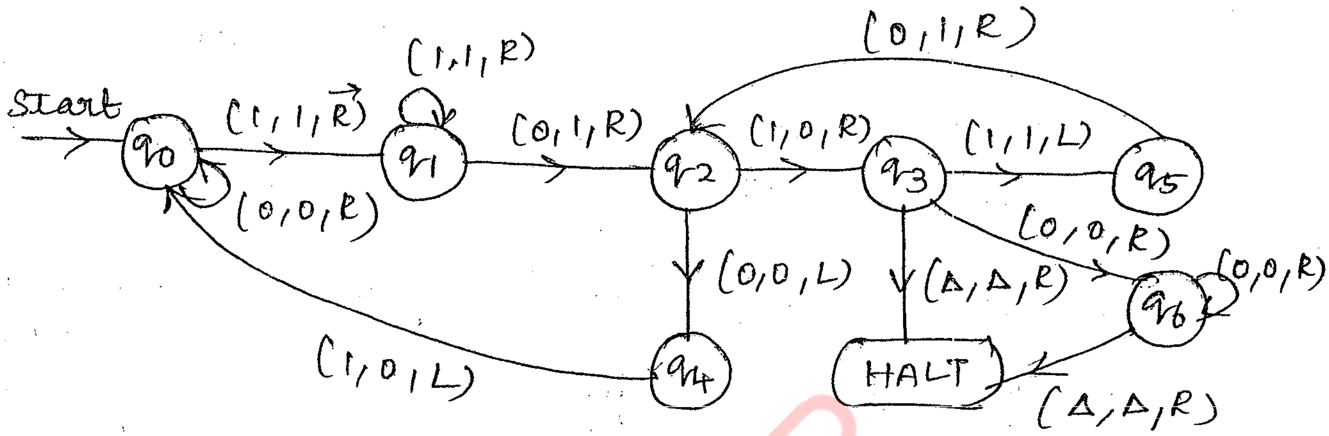
It can be modelled as,



Qn : Design a TM which recognise the i/p language having a substring as 101 & replaces every occurrence of 101 by 110.

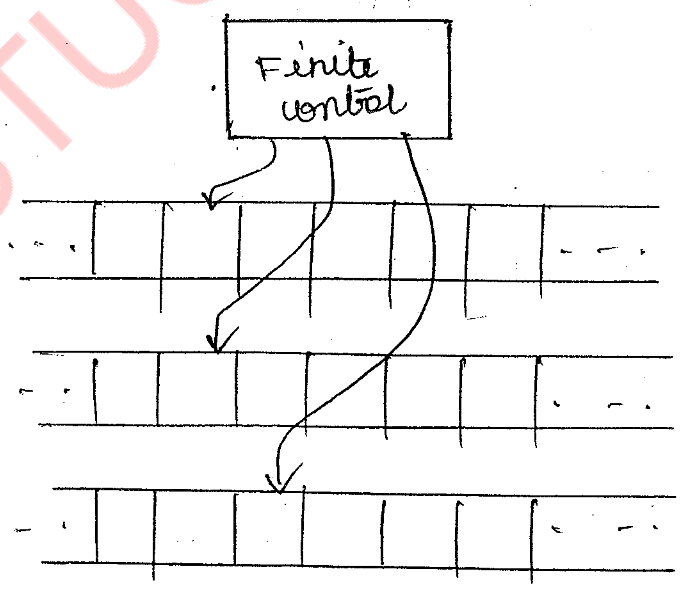
Soln : Replacement of any symbol by some another one means after reading of that specific symbol we should print the replacement symbol.

* In this case 101 has to be replaced by 110. Then TM has to be constructed considering 101 as a substring and leaving 110 substring after complete scan of the input.



Multitape Turing Machine:

The multitape TM has a finite control state and some finite number of tapes. Each tape in the multiple (or) multitape TM is divided into cells and each cell can hold any symbol & the multitape TM is shown below,



The multitape has the following,

(1) The i/p which is the finite sequence of i/p symbols and is placed on the 1st tape.

(2) All the other cells of all the tapes hold the blank symbols.

(3) The finite control is in the initial state.

(4) The head of the first tape is at the left end of the input.

(5) All the other tape head will be at some arbitrary cell.

Since the tapes other than 1st tape are completely blanks, there is no need to see where the head is placed initially and all the cells of those tapes look the same. A move of the multitape TM depends on the following.

(1) State of the finite control

(2) Symbol scanned by each tape head.

In a single move, the multitape TM does the following,

(1) The finite control enters a new state.

(2) On each tape, a new tape symbol is written on the cell scanned.

(3) Each of the tape head makes a move, which can be either left, right or stationary.

(4) The heads move independently, so different heads may move in different directions and some heads may

not at all move.

Checking off symbols :

The TM can be extended by using checking off symbol. This method is used by the TM for the languages that contain the repeated string, and to compare the length of the two substrings.

The examples languages are,

$$L = \{ wcw \mid w = \{a, b\}^* \}$$

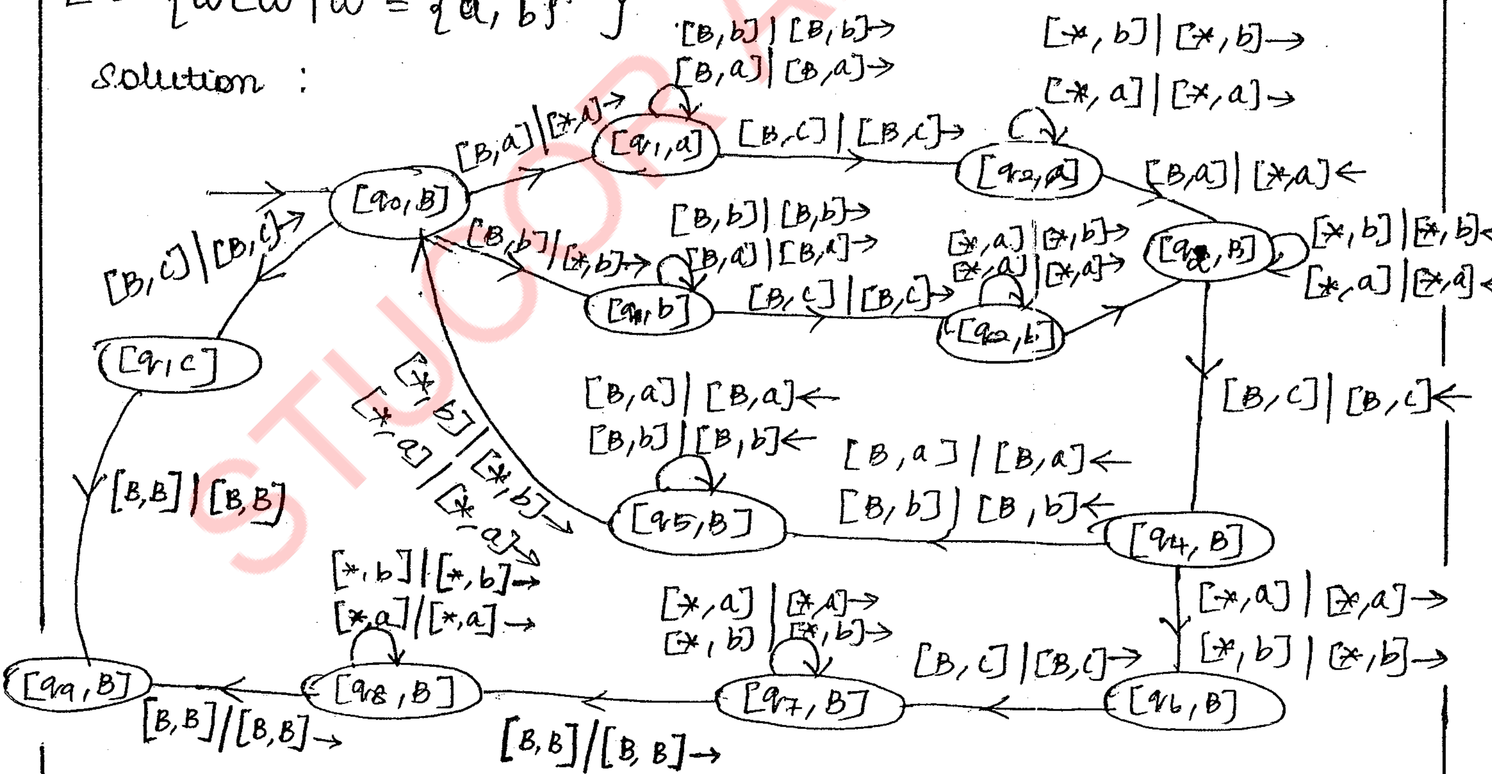
$$L = \{ ww \mid w = \{0, 1\}^* \}$$

$$L = \{ ww^R \mid w = \{0, 1\}^* \}$$

PROBLEM :

1) Design a turing machine to recognize the language $L = \{ wcw \mid w = \{a, b\}^* \}$

Solution :



Subroutines :

* There are some problems, in which some tasks need to be performed repeatedly and it can be done by subroutines. The subroutines are also called as function. The subroutine in the turing machine is a set of state that specifically performs some tasks.

→ The set of states in the subroutine has one start state and another state namely the return state.

→ The return state of the subroutine doesnot have moves and it pass the control to other set of states of the turing machine that calls the subroutine.

→ The subroutine is called whenever there is a transition to its initial state.

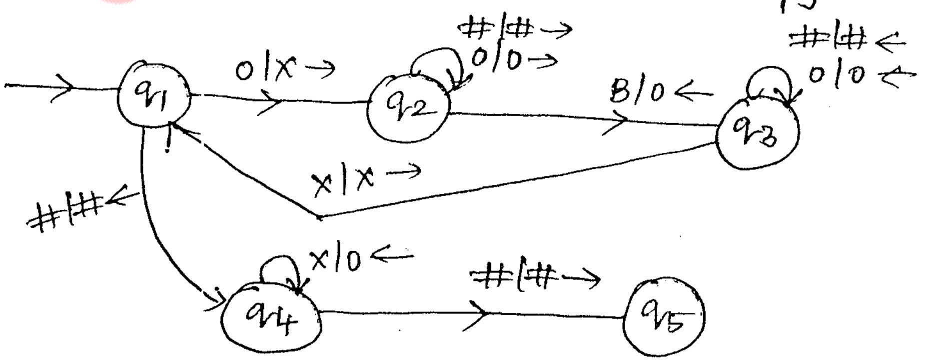
→ The calls are made to the start state of different copies of the subroutine and each copy returns to a different state.

→ The subroutines of the TM perform some task simultaneously.

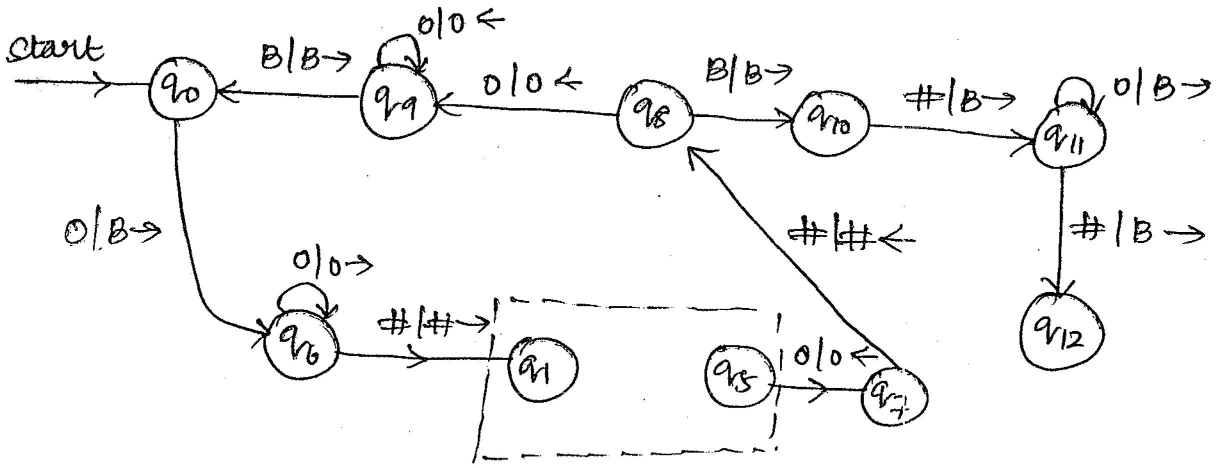
PROBLEM :

Design a TM to perform the multiplication function $f(m,n) = m * n$ using subroutine.

Soln: Transition diagram for subroutine copy.



Transition diagram for main program



The complete multiplication program uses the subroutine copy.

Transition table for subroutine copy program.

state	0	#	X	B
→ q_1	(q_2, X, R)	$(q_4, \#, L)$	—	—
q_2	$(q_2, 0, R)$	$(q_2, \#, R)$	—	$(q_3, 0, L)$
q_3	$(q_3, 0, L)$	$(q_3, \#, L)$	(q_1, X, R)	—
q_4	(—	$(q_5, \#, R)$	$(q_4, 0, L)$	—

Transition table for main program.

state	0	#	B
→ q_0	(q_6, B, R)	—	—
q_5	$(q_7, 0, L)$	—	—
q_6	$(q_6, 0, R)$	$(q_1, \#, R)$	—
q_7	—	$(q_8, \#, L)$	—
q_8	$(q_9, 0, L)$	—	(q_{10}, B, R)
q_9	$(q_9, 0, L)$	—	(q_0, B, R)
q_{10}	—	(q_{11}, B, R)	(q_7, B, R)
q_{11}	(q_{11}, B, R)	(q_{12}, B, R)	—
* q_{12}	—	—	—

Non - Deterministic Turing Machine [NTM]

* Non determinism is a powerful feature of TM.

* These NTM machines are easy to design and are equivalent to deterministic TM.

* A NTM accepts a string, w if there exists a least one sequence of moves from the initial state to final state.

Definition :

A NTM is defined as,

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

where $Q \rightarrow$ set of states including initial, having rejecting states.

$\Sigma \rightarrow$ finite set of input alphabets.

$\Gamma \rightarrow$ finite set of tape symbols.

$\delta \rightarrow$ Transition function defined by

$$\delta : Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R, N\})$$

where $P \rightarrow$ powerset.

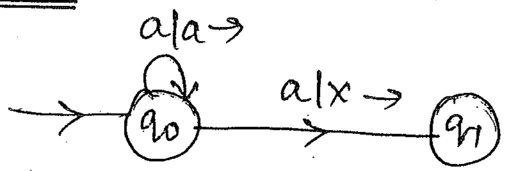
$q_0 \rightarrow$ initial state.

$B \rightarrow$ blank symbol

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

The transition function δ takes on the states tape symbols and head movement.

Example :



The above transition takes on two paths for the same input a. The transition of 'a' at q_0 is defined as

$$\delta(q_0, a) = \{(q_0, a, R), (q_1, x, R)\}$$

THE HALTING PROBLEM :

* The Halting problem is the problem of finding if the program & machine halts or loop forever.

* The halting problem is undecidable over TM.

Description :

* Consider the TM M and a given string w, the problem is to determine whether M halts by either accepting (or) rejecting w or run forever.

Example :

```
while(1)
{
  Printf ("Halting problem");
}
```

* The above code goes to an infinite loop since the argument of while loop is true forever.

* Thus it doesn't halts.

* Hence Turing problem is the example for undecidability.

* This concept of solving the halting problem being proved as undecidable was done by Turing in 1936.

* The undecidability can be proved by reduction techniques.

* Representation of the Halting set :

The halting set is represented as,

$$h(M, w) = \begin{cases} 1 & \text{if } M \text{ halts on input } w \\ 0 & \text{otherwise.} \end{cases}$$

where $M \rightarrow$ TM

$w \rightarrow$ I/P string

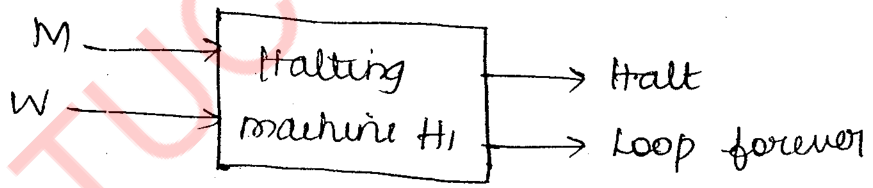
Theorem : Halting problem of TM is unsolvable / undecidable.

Proof :

* The theorem is proved by the method of proof by contradiction.

* Let us assume that TM is solvable/decidable

Construction of H_1

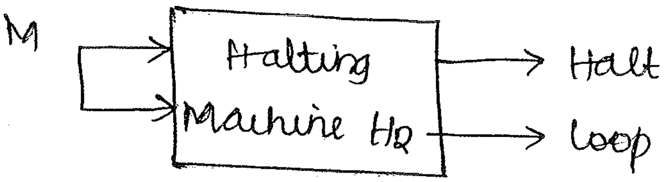


* Consider a string describing M and i/p string w for M.

* Let H1 generate "halt", if H_1 determines that the turing machine, M stops after accepting the i/p w.

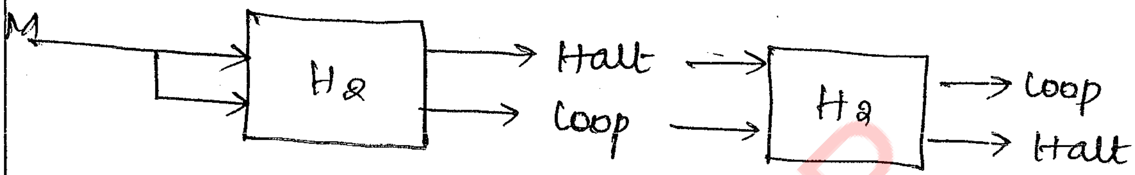
* Otherwise H1 loops forever when, M doesn't stop on processing w.

Construction of H2.

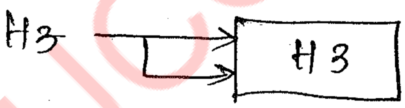


- H_2 is constructed with both the i/ps being M.
- H_2 determines M and halts if M halts otherwise loops forever.

Construction of H3.



- Let H3 be constructed from the outputs of H2.
- If the outputs of H_2 are halt, the H_3 loops forever
- else if the o/p of H_2 is loop forever than H_3 halts
- Thus H3 acts constructor to that of H_2.



- Let the output of H_3 be given as input to itself
- If the i/p is loop forever, then H_3 acts contradictory to it, hence halts.
- And if the i/p is halt, then H_3 loops by the construction
- Since the result is incorrect in both the cases, H_3 doesn't exist.
- Thus H_2 doesn't exist because of H_3.

- Similarly H_1, doesn't exist, because of H_2.
- Thus Halting problem is undecidable.

Partial solvability:

Problem Types,

There are basically three types of problems namely,

* Decidable / solvable / Recursive
* Undecidable / unsolvable
* Semidecidable / partial solvable / Recursively enumerable.

Decidable / solvable problems:

* A problem, P is said to be decidable if there exists a turing machine, TM that decides P.
* Thus P is said to be recursive.
* Consider a TM, M that halts with either "yes" or "no" after computing the input.

$$w \in \Sigma^* \longrightarrow \boxed{M} \begin{array}{l} \rightarrow \text{Yes (if } w \in L) \\ \rightarrow \text{No (if } w \notin L) \end{array}$$

* The machine finally terminates after processing.
* It is given by the function.

$$F_p(w) = \begin{cases} 1 & \text{if } p(w) \\ 0 & \text{if } \neg p(w) \end{cases}$$

* The machine that applies $F_p(w)$ is said to be turing computable.

Undecidable problem :

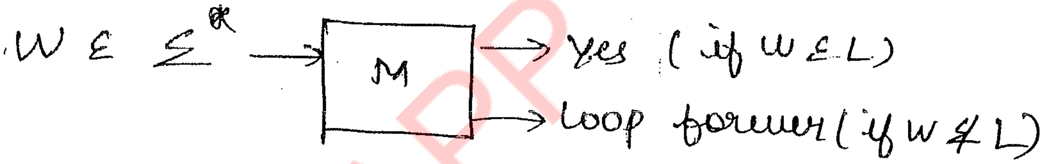
A problem, P is said to be undecidable, if there is a TM, TM that doesn't decide P.

Semidecidable / partial solvable / Recursively enumerable

* A problem, P is said to be semi-decidable, if P is recursively enumerable.

* A problem is RE if M terminates with "yes" if it accepts $w \in L$; and doesn't halt if $w \notin L$.

* Then the problem is said to be partial solvable (or) Turing acceptable.



* Partial solvability of machine is defined as

$$F_p(w) = \begin{cases} 1 & \text{if } P(w) \\ \text{undefined} & \text{if } \neg P(w). \end{cases}$$

Properties :

The semi-decidable [RE language are closed under

(1) union

(2) Intersection

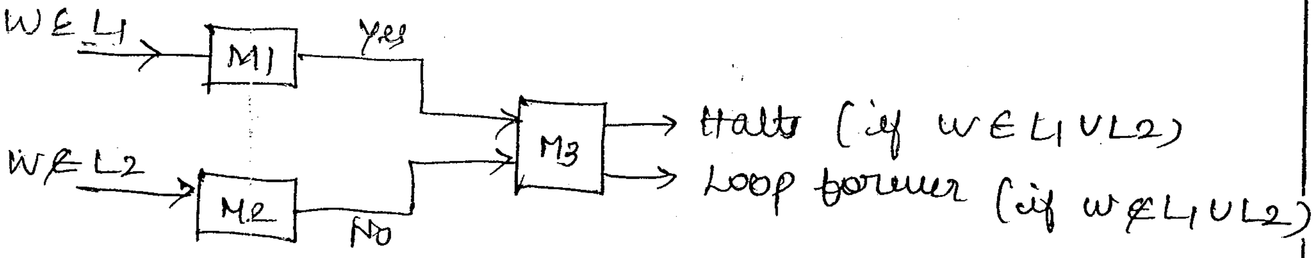
(3) But not under complementation.

Closure under union :

- Let L_1 & L_2 be two RE language.
- And consider M_1 that is a semi acceptor for L_1

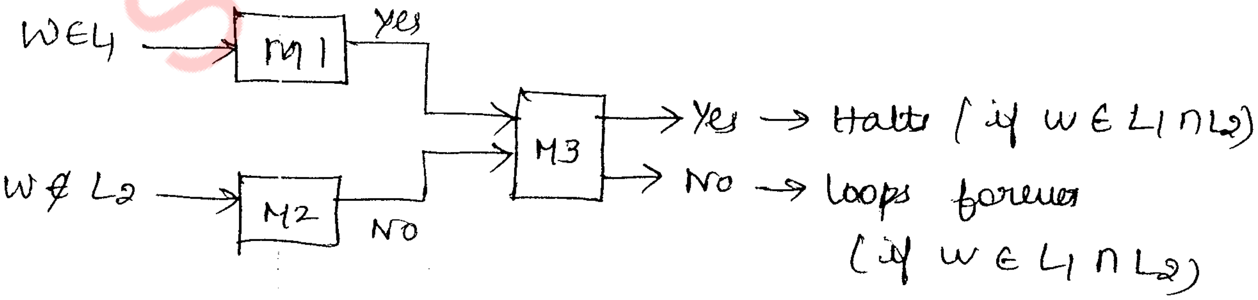
L_1 and M_2 be a TM for L_2.

- Let 'w' $\in L_1$ but not in L_2. Then $w \in L_1 \cup L_2$ and eventually M_3 halts if M_3 takes on both M_1 and M_2 and halts if any of them halts.
- If $w \notin L_1$ & $w \notin L_2$ then $w \notin L_1 \cup L_2$, which causes M_3 to loop forever.



Closed under Intersection :

- Let L_1 and L_2 be two RE languages accepted by M_1 and M_2 respectively.
- Let $w \in L_1$ and $w \in L_2$ be the input string.
- The TM, M_3 is constructed that takes on M_1 & M_2 and halts if both M_1 and M_2 halts.
- If $w \in L_1 \cap L_2$, M_3 halts with "yes".
- Else M_3 loop forever.



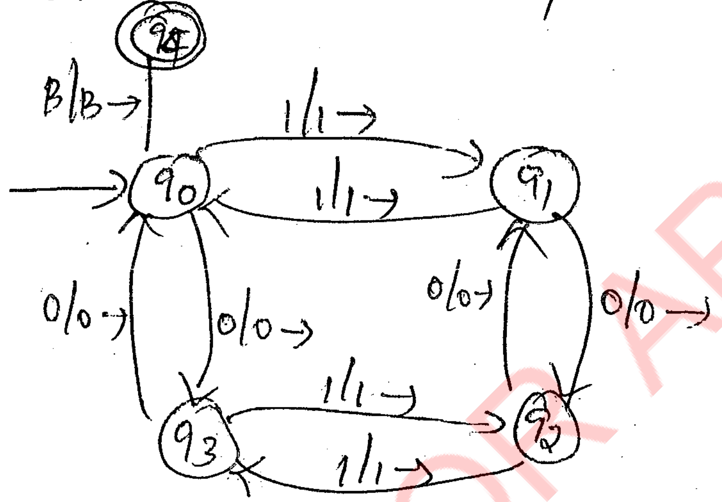
Note:

The next example of partial solvability is the halting problem ; acceptance problem.

CHOMSKY HIERARCHY OF LANGUAGES:

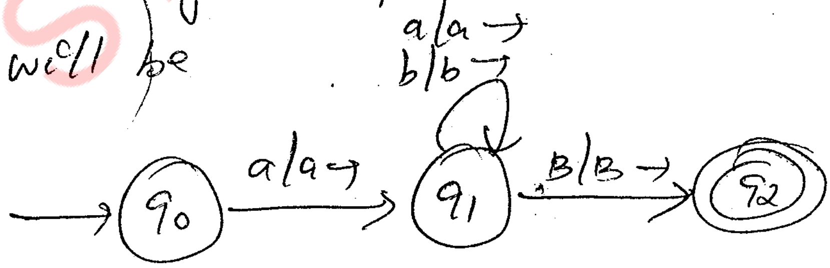
Refer UNIT - II.

Design a TM to accept the string with even number of 0's & 1's over the alphabet 0 & 1.



Design a TM with not more than three states that accepts the language $a(a+b)^*$. Assume $\Sigma = \{a, b\}$.

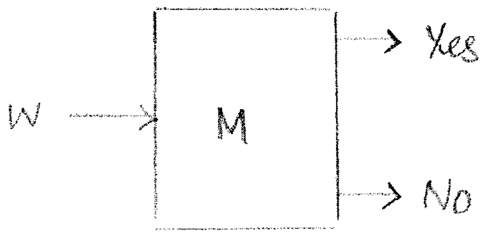
Soln Let Regular Expression $= a(a+b)^*$ The corresponding TM will be



UNIT V
UNDECIDABILITY

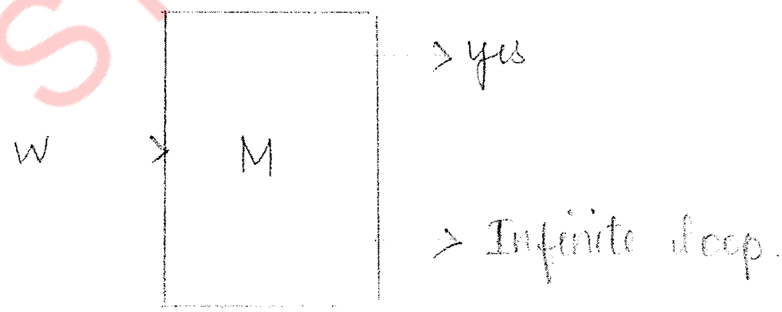
→ RECURSIVE LANGUAGE : (2m)

A language is recursive if there exists a turing machine that accepts every string of the language and rejects the string that is not in the language.



→ LANGUAGE THAT IS NOT RECURSIVE ENUMERABLE : (2m)

A language is recursively enumerable if there exists a turing Machine that accepts every string the language and does not accept strings that are not in the language.



→ DIAGONALIZATION LANGUAGE (L_d) : (2m)

The diagonalization language L_d, is the set of strings w_i where w_i is not in $L(M_i)$.

$\boxed{\therefore\ L(M_i) = \phi \text{ if } w_i \text{ valid TM code.}}$

L_d consists of all strings w such that the TM 'M' whose code is w does not accept the input w.

$$
\begin{array}{c|cccc}
i \downarrow \backslash j \rightarrow & 1 & 2 & 3 & 4 \ldots \\
\hline
1 & 0 & 1 & 0 & 0 \ldots \\
2 & 1 & 0 & 1 & 1 \ldots \\
3 & 1 & 1 & 0 & 0 \ldots \\
4 & 1 & 1 & 1 & 1 \ldots \\
\vdots & \vdots & \vdots & \vdots & \vdots
\end{array}
$$

Diagonal.

It is clear that whether the Tm M_i accepts the input string or not.

If $(i,j) = 1$, Yes it is accepted.

If $(i,j) = 0$, No it is rejected.

→ DIAGONALIZATION : (2m)

The process of complementing the diagonal to construct the characteristic vector of a language that cannot be the language that appears in any row is called Diagonalization.

Then the complement of the diagonal cannot be the characteristic vector of any turing machine.

THEOREM 1 : L_d is not recursively enumerable

L_d is not a recursively enumerable language.
(ie) There is no Turing Machine that accepts L_d.

PROOF: Suppose L_d is accepted by some TM M defined by L(M). Since L_d is a language over alphabet $\{0,1\}$, M would be in the list of Turing Machines constructed, where it includes all turing machines with input alphabet $\{0,1\}$. So there may be atleast one code for M, say i that is

$$M = M_i$$

Then

By definition, $L_d = \{w_i \mid M_i \text{ does not accept}\}$
Here we have two possibilities

- $w_i \in L_d$

* This means that (i,i) entry is '0' and so M_i does not accept w_i. But our assumption there is that there exists a Turning machine M_i, which accepts w_i. There is a contradiction.

- $w_i \notin L_d$

* This means that (i,i) entry is '1' and so M_i accept w_i. But by definition of L_d, M_i does not accept w_i. So there is a contradiction.

Thus it is clear that L_d is not recursively enumerable, and L_d is not recursive too.

COMPLEMENTATION OF RECURSIVE AND RECURSIVE ENUMERABLE LANGUAGES

The recursive languages are closed under complementation.

THEOREM : 2 If L is a recursive language, so is $\bar{L}$

Let L be a recursive language and M a Turing Machine that halts on all inputs and accepts L.

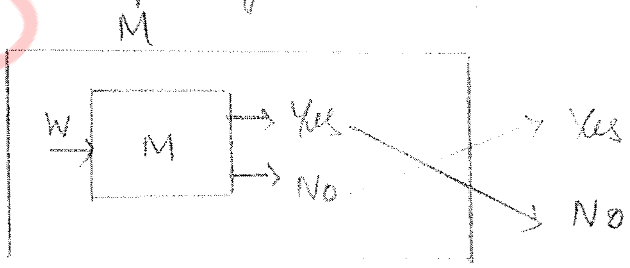
Construct a Turing Machine M' from M such that,

(i) The accepting states of M are made non-accepting states of M with no transitions.

(ii) Create a new accepting state P which has no transitions

(iii) For each pair of non-accepting state of M and a tape symbol of M such that M has no transition. Make the transition to the accepting state p'.

If M enters a final state on input w, then M halts without accepting.



If M halts without accepting, M enters to final state.

M	M'
⇒ Accept → final state	Rejects.
⇒ Rejects	Accept → final state.

It is clear the M has either of the event accepts or rejects.

So M' accepts $\bar{L}$ which is also recursive.

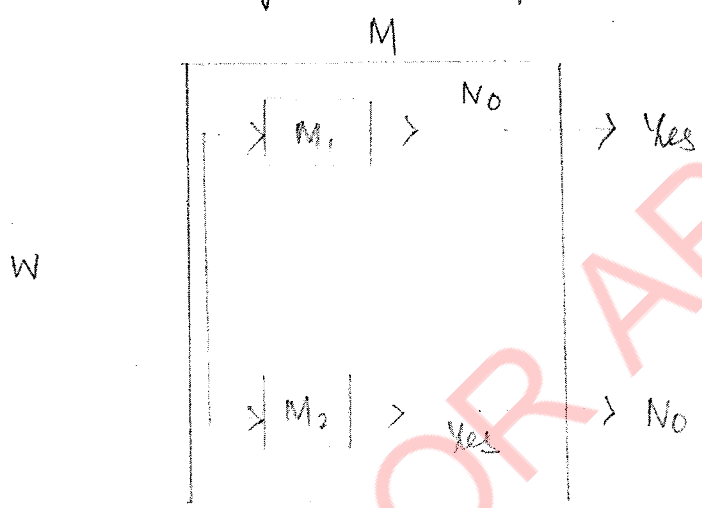
THEOREM 3: If a language L and its complement $\bar{L}$ are both recursively enumerable, then L is recursive.

Proof: Let M_1 and M_2 be the Turing Machine which accepts the language L and $\bar{L}$ respectively.

Construct M to simulate simultaneously M_1 and M_2.

M accepts w if M_1 accepts w.

M ~~rejects~~ rejects w if M_2 accepts w.

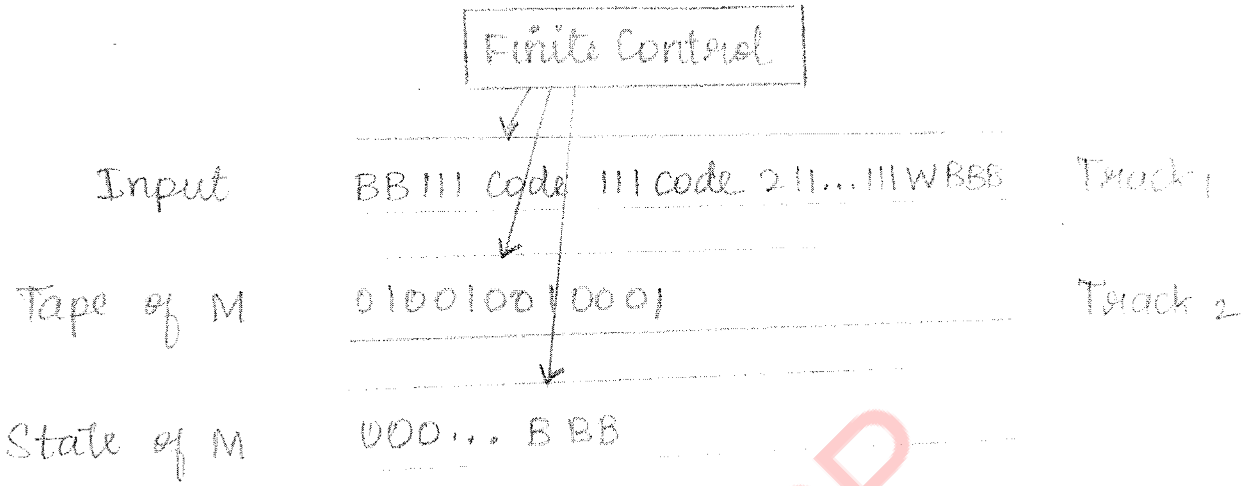


w is in either L or $\bar{L}$, so exactly one of M_1 or M_2, will accept.

Obviously M will always say either 'Yes' or 'No' but not both.

THEOREM 4: Lu is recursively enumerable.

PROOF: In order to prove this theorem, it is necessary to construct a Turing machine that accepts Lu. The turing machine U consists of a three track input tape where the first track holds the input tape (<M, w>), the second track contains the tape of M where tape symbols are written in unary form and the third track represents the state q_i which is also in unary form.



The operation of U are as follows:

1.) First make sure that the code for M is a legitimate code for some Turing machine M. Otherwise it halts without accepting.

2.) Initialize the second tape with the input w, in its encoded form. Keep 0 the start state of M on the third tape and move the head of U's second tape of the first stimulated cell.

3.) If 0^i is the current state with 0^j the current input symbol appeared on track three and two respectively then U finds the corresponding transition of the form $0^i 1 0^j 1 0^k 1 0^l 1 0^m$ on track 1 and replaces 0^i by 0^k and 0^j by 0^l.

4.) Move the head on tape two to the position corresponding to the value of M.

5.) The universal Turing Machine U accepts $0^i 1 0^j 1 0^k 1 0^l 1 0^m$ if M has the transition of the form $\delta(0^i, 0^j) = (0^k, 0^l, 0^m)$. Otherwise M halts without accepting.

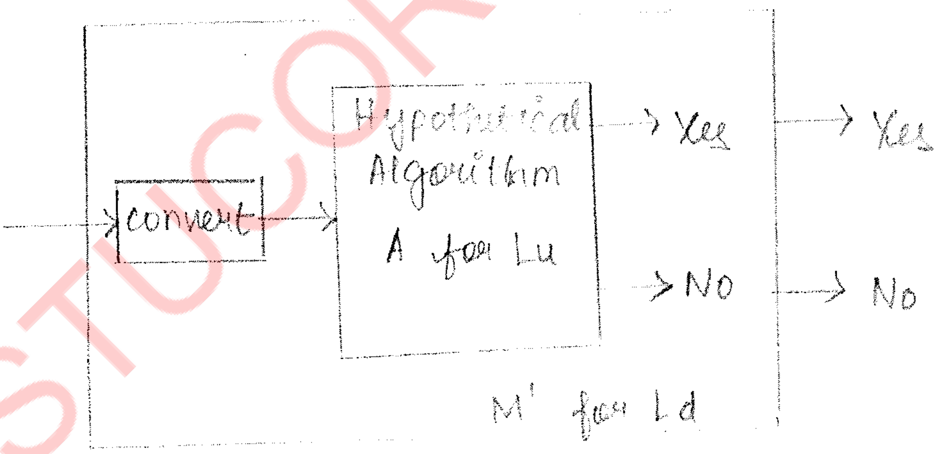
Thus U simulates M and accepts w. Thus Lu is recursively enumerable.

THEOREM 5 : Lu is Recursive enumerable but not recursive.

PROOF :

It is already proved that Lu is RE.

Assume Lu is recursive. According to the closure property of recursive sets, the complement of Lu (ie) $\overline{Lu}$ is also recursive.



Suppose A is an algorithm recognizing Lu Ld can be recognized as follows:

⇒ Given string w in $(0+1)^*$ determined by an easy calculation, the value of i such that $w = w_i$. Integer i in binary is the corresponding code for

for some TM M_i.

→ Give the input $\langle M_i, w_i \rangle$ to algorithm A and accept w if and only if M_i accepts w_i.

→ So the constructed algorithm also accepts w if and only $w = w_i$ which is in $L(\overline{M_i})$. This is the algorithm constructed for L_d.

But no such algorithm exists and also our assumption of there is an algorithm A for L_u exists is false.

Hence L_u is RE but not recursive.

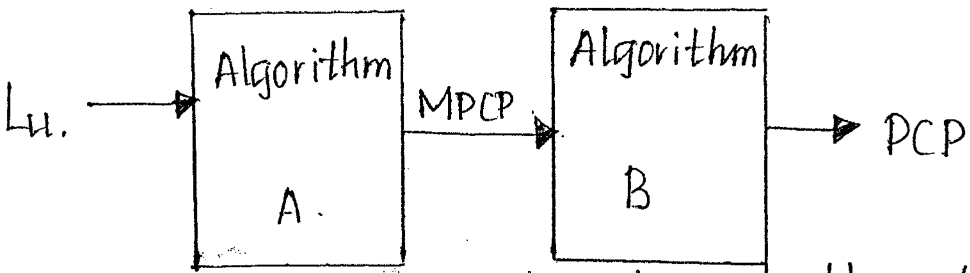
Universal Language (L_u) : (2m)

$$L_u = \{ \langle M, w \rangle \mid M \text{ accepts } w \}$$

L_u is the set of strings representing a TM and an input accepted by that TM. So there is a TM U called Universal Turing Machine.

POST CORRESPONDENCE PROBLEM

In this, the undecidable problem about Turing machine are reduced to undecidable problems about real things. The goal is to prove that Problem about strings to be undecidable



we reduce Lu to modified PCP then to PCP using algorithm.

Definition.

An instance of Post's Correspondence problem (PCP) consists of two lists of strings over Σ

$$A = w_1, w_2, \ldots w_k.$$

$$B = x_1, x_2 \ldots x_k \text{ for some integer } k$$

The instance of PCP has a solution if there is any sequence of integers $i_1, i_2, \ldots i_m$ with $m \geq 1$ such that.

$$w_{i1}, w_{i2}, w_{i3} \ldots, w_{im} = x_{i1}, x_{i2} \ldots x_{ik}.$$

is a solution to this instance of PCP

MODIFIED POST'S CORRESPONDENCE PROBLEM

[MPCP]

Modified PCP

1. In order to simplify the reduction of L_u to PCP, an intermediate version of PCP called Modified Post's Correspondence problem is used.

The Modified PCP is the following

Given lists A and B, of K string each from Σ^*

$$A = W_1, W_2, \ldots W_K \qquad B = X_1, X_2, \ldots X_K.$$

It has the solution such that

$$W_1, W_{i1}, W_{i2} \ldots W_{ir} = X_1, X_{i1}, \ldots X_{ir}.$$

The difference between the MPCP and PCP is that in the MPCP, a solution is required to be stored with the first string on each list.

Theorem 6

Statement:

If there is a reduction from to then

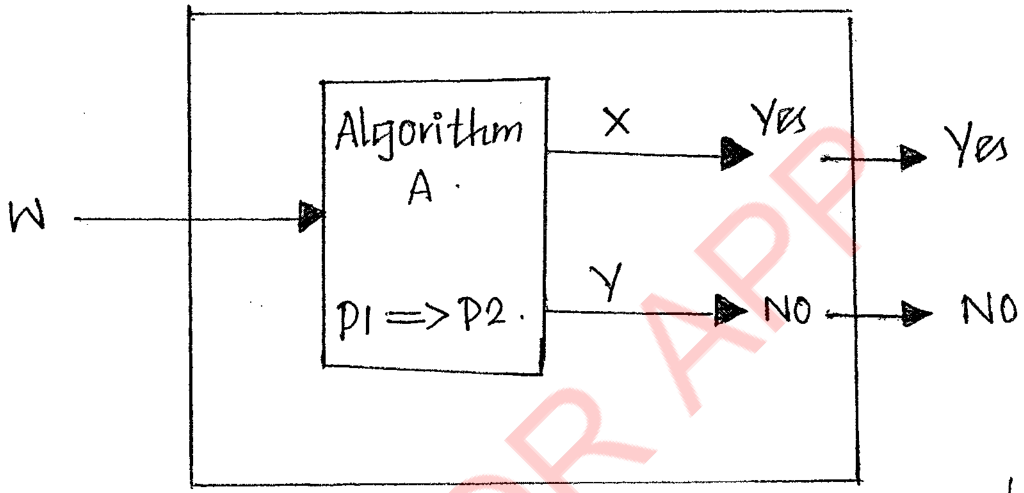
i) If $P1$ is undecidable, then So is P_2

ii) If $P2$ is non-RE, then So is P_2

Proof:

i) If P_2 is undecidable, then So is P_2

Assume P_1 is undecidable. Let A be the algorithm which Converts the instances of P_1 to instances of P_2



Suppose W be the instances of P_1, given to the algorithm A that Converts W into an instance X of

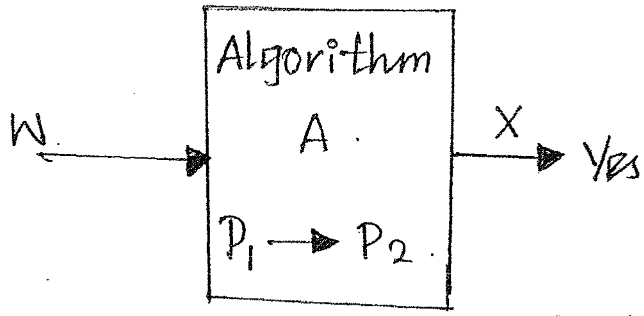
Algorithm output	Inference
Yes	X is in P_2 [∵ W is in P_1]
NO	X is not in P_2 [∵ W is not in P_3]

This is the Contradiction to our assumption that P_1 is undecidable

Thus, if P_1 is undecidable, then P_2 is also undecidable.

ii) If P_1 is non-RE, then so is P_2

Assume that P_1 is non-RE, but P_2 is RE. Since P_2



If w is in P_1 then x is in P_2, So the TM will accept w

If w is not in P_1, then x is not in P_2, so the TM ... may or may not halt but will not accept w

This is a contradiction to our assumption. Thus if P_1 is non-RE, then P_2 is non RE

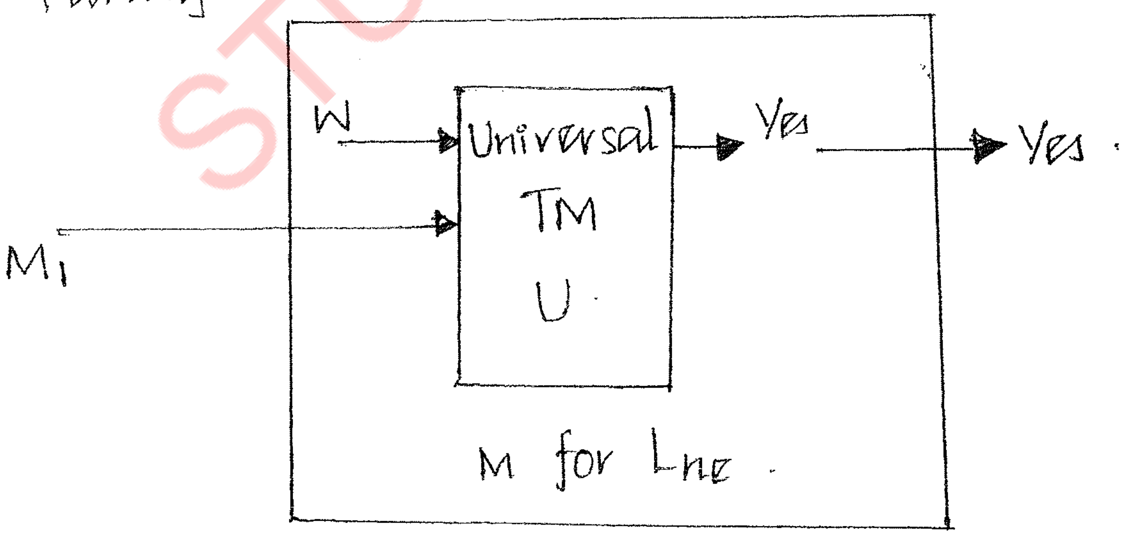
Theorem 7

Statement :

L_{ne} is recursively enumerable

Proof :

The construction is based on a non-deterministic Turing machine



The theorem is proved as follows

i. A Turing Machine Code M_i is given a input to the TM

ii. A guesses an input w in a right way that M_i might accept.

iii. M is simulated to the Universal Turing Machine U, which tests whether M_i accepts w.

iv. If M_i accepts w, then M accepts w

Thus M_i accepts any string that will be guessed right by the Turing Machine M. If $L(M_i) = \phi$, then no guess is made by Turing Machine M, so M does not accept M_i.

Thus $L(M) = L_{ne}$

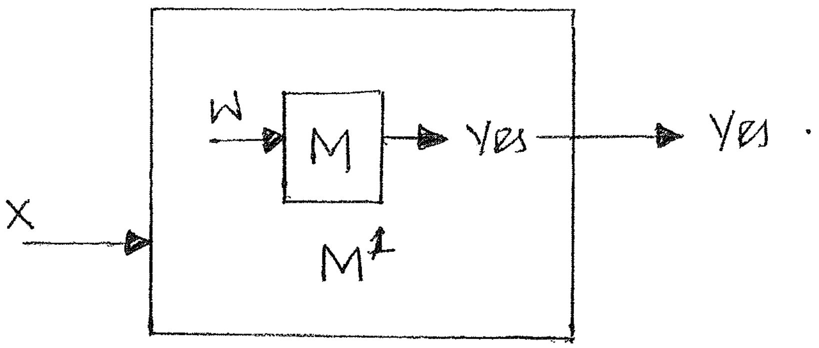
Theorem 8

Statement:

L_{ne} is not recursive.

Proof:

The algorithm of this theorem should be designed in such a way that it converts an input that is binary-coded pair (M, w) into a Turing Machine M^1 such that $L(M^1) = \phi$ if and only if M accepts input w



The Turing Machine M^1 is designed to perform the below operation

1. M ignores its own input x rather it replaces its input by w, the input string accepted by Turing Machine M. M is designed to accept a specific pair (M, w) whose length is n, having a sequence of n states like $q_0, q_1, q_2, \ldots q_n$, where q_0 is the start state

a) For $i = 0, 1 \ldots n-1$, if the turing machine is at state q_1, M writes $(i+1)$st bit of the code for (M, w) and goes to state q_{i+1} moving right.

b) In state q_n, M moves right by replacing any non blanks to blanks

2. When the Turing machine M reaches a blank in state q_n, it uses a similar collection of states to reposition its head at the left end of the tape.

3. M stimulates a universal Turing machine U on its present tape. If U accepts, then M accepts, if U does not accept, M never accepts. The simulation made here is by reduction of L_u to L_{ne}.

Assume L_{ne} is recursive. Then the algorithm for L_u is as follows

1. Convert (M, W) to the Turing machine M^1 by reduction of L_u to L_{ne}

2. By the hypothetical algorithm of L_{ne}, Tell
   i) If $L(M^1) = \phi$, M does not accepts W
   ii) if $L(M^1) = \phi$ M accepts W

By the algorithm of L_u, this is not true. Our assumption is contradictory and conclude that L_{ue} is not recursive.

Theorem 9

Statement

MPCP reduces to PCP

Proof:

Assume that $i_1, i_2, \ldots\ldots i_m$ is a solution to the given MPCP instance with lists A and B

We know that, according to MPCP

$$W_1 W_{i1} W_{i2} \ldots W_{im} = X_1 X_{i1} \ldots X_{im}$$

Replace w's by y's and x's by z's. Then we have the string like $y_1, y_{i1}, y_{i2} \ldots y_{im}$ and $Z_1, Z_{i1}, Z_{i2} \ldots Z_{im}$. The only difference is that the first string would be missing a * at the beginning and the second string would be missing a* at the end

$$* y_1, y_{i1}, y_{i2} \ldots y_{im} = Z_1, Z_{i1}, Z_{i2} \ldots Z_{im}$$

Using the construction rule, Put $y_0 = * y_1$ and $Z_0 = Z_1$. Then fix the initial * by replacing the first index by 0

$$y_1, y_{i1}, y_{i2} \ldots y_{im} = Z_1, Z_{i1}, Z_{i2} \ldots Z_{im} *$$

Append the index K+1 i.e $y_{K+1} = \$$ and Z_{K+1} ~~*\$~~

$Z_{K+1} = * \$$

$$* y_1, y_{i1}, y_{i2} \ldots y_{im} y_{K+1} = Z_1, Z_{i1}, Z_{i2} \ldots Z_{im} Z_{K+1}$$

Thus $0, i, i_2, \ldots i_m \cdot K+1$ is a solution to the instance of PCP.

For MPCP, $i_1, i_2, \ldots i_m$ is a solution. If we remove the *'s and the final \$ from the string $y_1, y_{i_1}, y_{i_2} \ldots y_{i_m} y_{k+1}$, we get the string

$$w_1 w_{i_1} w_{i_2} \ldots w_{i_m}.$$

If we remove *'s and \$ from the string $z_1, z_{i_1}, z_{i_2}, \ldots z_{i_m} z_{k+1}$, we get

$$x_1 x_{i_1} x_{i_2} \ldots x_{i_m}$$

So for PCP

$$y_1, y_{i_1}, y_{i_2} \ldots y_{i_m} y_{k+1} = z_1, z_{i_1}, z_{i_2} \ldots z_{i_m} z_{k+1}$$

which follows that

$$w_1 w_{i_1} w_{i_2} \ldots w_{i_m} = x_1 x_{i_1} x_{i_2} \ldots x_{i_m}$$

Thus a solution to the PCP instance implies a solution to the MPCP instance. Thus there is a reduction of MPCP to PCP, which confirms that if PCP were decidable, PCP would also be decidable.

Theorem 10

Statement

Post's Correspondence problem is undecidable

Proof:

The proof of this theorem is telling how to reduce L_u to MPCP. It can be proved using the statement

"M accepts w if and only if the constructed MPCP instance has a solution".

if Part

MPCP instance has a solution

To Prove: M accepts w

Assume a partial solution begins with

A : #

B : $\#q_0 w\#$

states and the tape symbols can only be handled by the pairs of rule (3) and all other tape symbols and # must be handled by pairs from rule (2). After reaching the accepting states, the rules (4) and (5) can be used. Thus, unless M reaches an accepting state, all partial solutions have the form

A : x

B : xy

where

x - sequence of ID's of M representing a computation of M on input w possibly followed by # and the beginning of the next ID a

y - Completion of a, # and beginning of the ID that follows a, upto the point that x ended within a itself

Thus as long as M does not enter an accepting state, the partial solution is not a solution and also B is longer than A.

Only-if part M accepts w

If w is in L(M), then start with the pair from rule (1) and simulate the computation of M on w. And using the rule (3) pair to copy the state from each ID and simulate one move of M. Then by using rule (2), to copy tape symbols and the marker # as needed. If M reaches an accepting state, then the pairs from rule (4) and rule (5) allow the A string to catch upto the B string and form a solution.

Thus if there is a solution M must enter an accepting State and so M accepts w

2 mark

20. Define the classes P and NP.?

P Consists of all those languages or problems accepted by some Turing Machine that runs in some polynomial amount of time, as a function of its input length Ex: Krushkal's problem.

NP is the class of language or problems that are accepted by non-deterministic TM's with a polynomial bound on the time taken along any sequence of non-deterministic choices ~~and~~ and also Verifiable in Polynomial time.
Ex: Travelling Salesman

5. Define NP hard problem?

A problem is said to be NP hard if there is a polynomial time reduction i.e can be solvable in polynomial time.

Example:

Decision problems, Search problems.

13. Define the basic recursive primitive function?

i) Zero function $z(x) = 0$

ii) Successor function $s(x) = x + 1$

iii) Projection function

$$P_1(x_1, x_2) = x_1$$

$$P_2(x_1, x_2) = x_2$$

14. What do you mean by NP Complete problem?

A language L is NP Complete if its satisfies the following condition

1. L is in NP
2. For every language L' in NP there is a polynomial-time reduction of L' to L

Example: Travelling Salesman Problem.

19. What are the properties of recursively enumerable sets which are undecidable?

1. Emptiness
2. Finiteness
3. Regularity
4. Context-freedom

UNIT - 5

UNDECIDABILITY

Non Recursive Enumerable (RE) Language – Undecidable Problem with RE – Undecidable Problems about TM – Post‘s Correspondence Problem, The Class P and NP

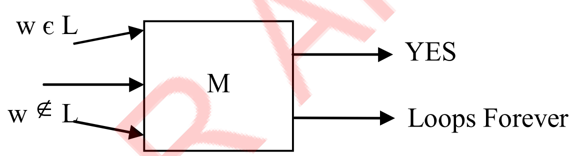
RECURSIVE AND RECURSIVELY ENUMERABLE LANGUAGES

Recursively Enumerable Language

A language $L \subseteq \sum^*$ is recursively enumerable if there exist a Turing machine, M that accepts every string, w $\in$L and does not accept strings that are not in L.

If the input string is accepted, M halts with the answer, “YES”.

If the string is not an element of L, then M may not halt and enters into infinite loop.

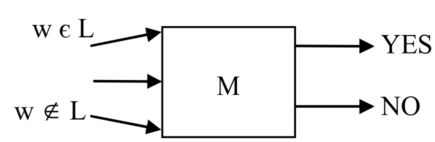


The language, L is Turing Acceptable.

Recursive Language

A language is said to be recursive if there exists of Turing machine, M that accepts every string, w $\in$L and rejects those strings that are not in L.

If the input is accepted, M halts with the answer,” YES”



w $\notin$L the Turing machine doesn‟t accept the string.

If w $\notin$L, then M halts with answer, "NO". This is also called as Turing Decidable language.

PROPERTIES OF RECURSIVE AND RE LANGUAGES

1. The union of two recursive language is recursive
2. The language L and its complement $\bar{L}$ are recursively enumerable, then L is recursive.
3. The complement of a recursive language is recursive.
4. . The Union of two recursively enumerable languages is recursively enumerable.
5. The intersection of two recursive language is recursive.
6. The intersection of two recursively enumerable language is recursively enumerable

Proofs on the Properties

Property-1

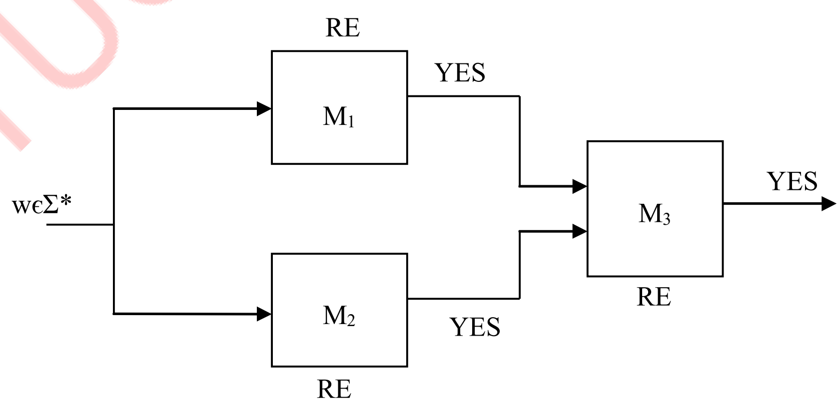
The union of two recursively enumerable languages is recursively enumerable.

Proof:

Let L_1 and L_2 be two recursively enumerable languages accepted by the Turing machines M_1 and M_2.

If a string $w \in L_1$ then M_1 returns "YES", accepting the input string: Else loops forever. Similarly if a string $w \in L_2$ then M_2 returns "YES", else loops forever.

The Turing machine M_3 that performs the union of L_1 and L_2 is given as



Here the output of M_1 and M_2 are written on the input tape of M_3. The turning machine, M_3 returns "YES", if at least one of the outputs of M_1 and M_2 is "YES". The M_3 decides on $L_1 U L_2$ that halts with the answer, "YES" if $w \in L_1$ or $w \in L_2$. Else M_3 loops forever if both M_1 and M_2 loop forever.

Hence the union of two recursively enumerable languages is also recursively enumerable.

Property – 2

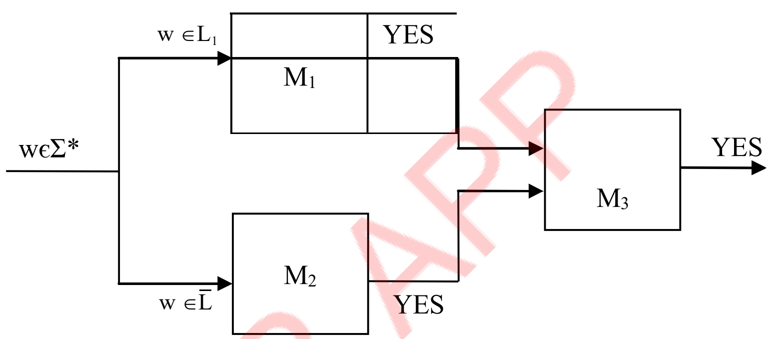
A language is recursive if and only if both it and its complement are recursively enumerable.

Proof

Let L and $\bar{L}$ be two recursively enumerable languages accepted by the Turing machines M_1 and M_2. If a string, $w \in L$, it is accepted by M_1 and M_1 halts with answer "YES". Else M_1 enters into infinite loop.

If a string, $w \in \bar{L}$ $[w \notin L]$, then it is accepted by M_2 and M_2 halts with answer "YES". Otherwise M_2 loops forever.

The Turing machine, M_3 that simulates M_1 and M_2 simultaneously is given as



From the above design of TM, if $w \in L$, if $w \in L$, then M_1 accepts w and halts with "YES".

If $w \notin L$, then M_2 accepts $w[w \in L]$ and halts with "YES".

Since M_1 and M_2 are accepting the complements of each other, one of them is guaranteed to halt for every input, $w \in \Sigma^*$.

Hence M_3 is a Turing machine that halts for all strings.

Thus if the language and its complement are recursively enumerable, then they are recursive.

Property - 3

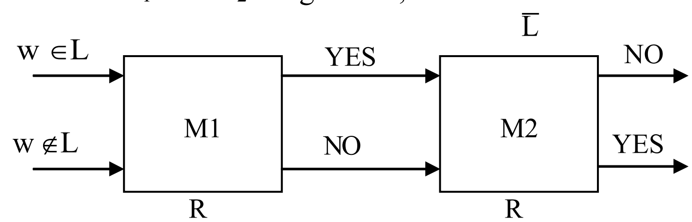
The complement of a recursive language is recursive.

Proof

Let L be a recursive language accepted by the turning machine, M_1.

Let $\bar{L}$ be a recursive language accepted by the Turing machine M_2.

The construction of M_1 and M_2 are given as,



Let w ∈L, then M_1 accepts w and halts with "YES".

M_1 rejects w if w ∉Land halts with "NO"

M_2 is activated once M_1 halts.

M_2 works on $\bar{L}$ and hence if M_1 returns "YES", M_2 halts with "NO".

If M_1 returns "NO", then M_2 halts with "YES"

Thus for all w, where w ∈L or w ∉L , M_2 halts with either "YES" or "NO"

Hence the complement of a recursive language is also recursive.

Property – 4

The union of two recursive language is recursive.

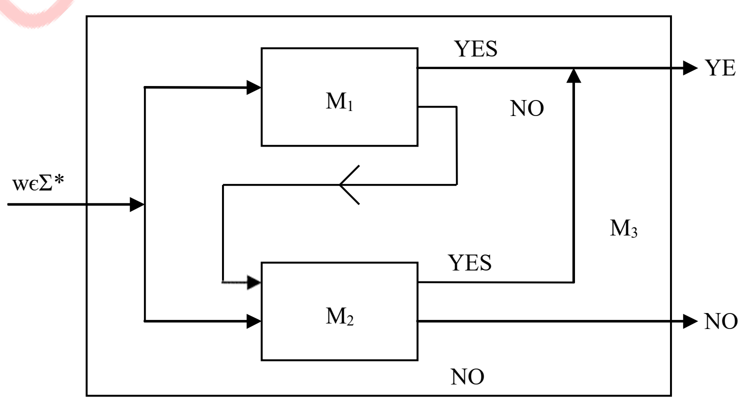
Proof:-

Let L_1 and L_2 be two recursive languages that are accepted by the Turing machines M_1 and M_2, given by

$L(M_1) = L_1$

$L(M_2) = L_2$

Let M_3 be the Turing machine constructed by the union of M_1 and M_2. M_3 is constructed as follows.



The Turing machine M_3 first simulates M_1 with the input string, w.

If $w \in L_1$, then M_1 accepts and thus M_3 also accepts since $L(M_3) = L(M_1)$ u $L(M_2)$.

If M_1 rejects string $[w \notin L_1]$, then M_3 simulates M_2. M_3 halts with "YES" if M_2 accepts „w", else returns "NO".

Hence M_3, M_2, M_1 halt with either YES or NO on all possible inputs.

Thus the union of two recursive languages is also recursive.

Property – 5

The intersection of two recursive language is recursive.

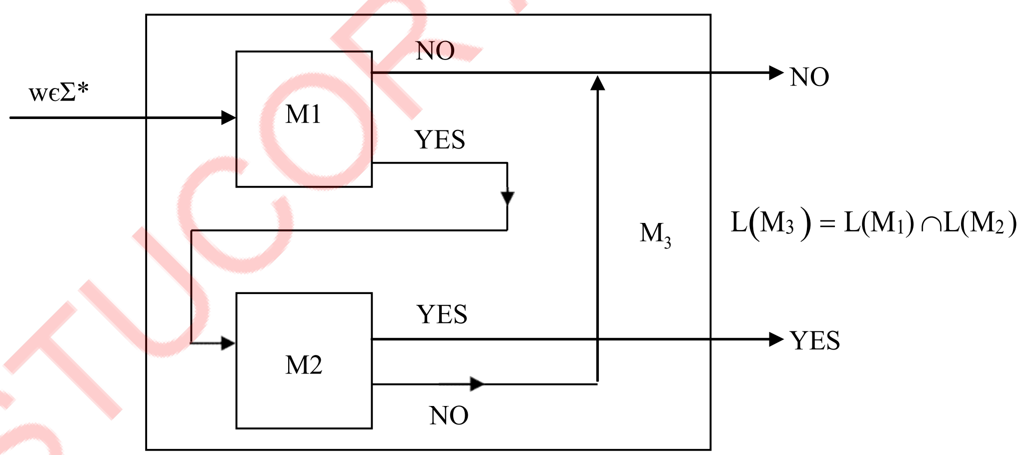
Proof:-

Let L_1 and L_2 be two recursive languages accepted by M_1 and M_2 where

$L(M1) = L_1$

$L(M2) = L_2$

Let M_3 be the Turing machine that is constructed by the intersection of M_1 and M_2, M_3 is constructed as follows.



The Turing machine M_3 simulates M1 with the input string,w.

If $w \notin L_1$, then M_1 halts along with M_3 with answer "NO", since $L(M_3)=L(M_1) \cap L(M_2)$. If then M_1 accepts with the answer "YES" and M_3 simulates M_2.

If M_2 accepts the string, then the answer of M_2 and M_3 are "YES" and halts. Else, M_2 and M_3 halts with answer "NO".

Thus, the intersection of two recursive languages is recursive.

Property – 6

Intersection of two recursively enumerable languages is recursively enumerable.

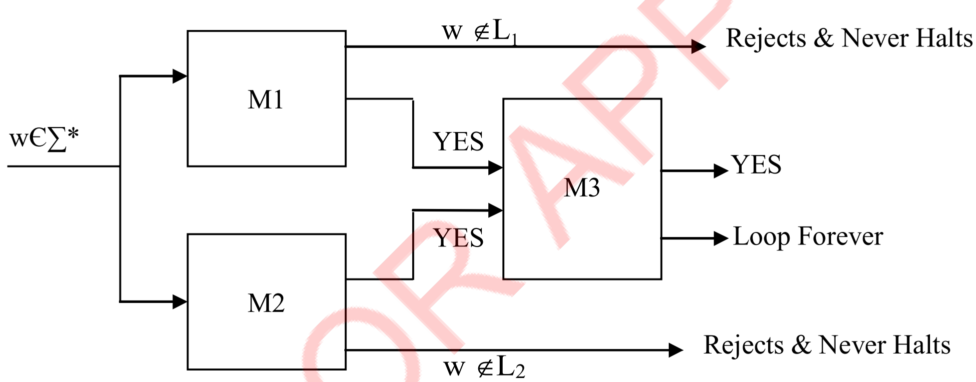
Proof:-

Let L_1 and L_2 be two recursively enumerable languages accepted by the Turing machine M_1 and M_2.

If a string $w \in L_1$ then M_1 returns "YES" accepting the input. Else will not halt after rejecting $w \notin L_1$.

Similarly if a string, $w \in L_2$, then M_2 returns "YES" else rejects „w" and loop forever.

The Turing machine, $M_3 = M_1 \cap M_2$ is given as



Here the output of M_1 and M_2 are written the input tape of M_3. The machine, M_3 returns "YES" if both the outputs of M_1 and M_2 is "YES".

If at least one of M_1or M_2 is NO it rejects „w" and never halts.

Thus M_3 decides on $L_1 \cap L_2$ that halts if and only if $w \in L_1$ and $w \in L_2$. Else M_3 loops forever along with M_1 or M_2 or both

Hence the intersection of two recursively enumerable languages is recursively enumerable.

THE HALTING PROBLEM

- The halting problem is the problem of finding if the program/machine halts or loop forever.
- The halting problem is un-decidable over Turing machines.

Description

- Consider the Turing machine, M and a given string ω, the problem is to determine whether M halts by either accepting or rejecting ω, or run forever.
- **Example**

```
while (1)
{
    prinf("Halting problem");
}
```

- The above code goes to an infinite loop since the argument of while loop is true forever.
- Thus it doesn‟t halts.
- Hence Turing problem is the example for undecidability.
- This concept of solving the halting problem being proved as undecidable was done by Turing in 1936.
- The undecidability can be proved by reduction technique.

Representation of the halting set

The halting set is represented as,

$$h(M, \omega) = \begin{cases} 1 & \text{if M halts on input } \omega \\ 0 & \text{otherwise} \end{cases}$$

where,

M → Turing machine

ω → Input string

Theorem

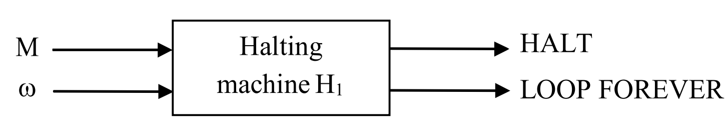
Halting problem of Turing machine is unsolvable / undecidable.

Proof

The theorem is proved by the method of proof by contradiction.

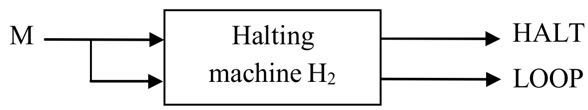
Let us assume that Turing machine is solvable / decidable.

Construction of H_1



- Consider, a string describing M and input string, ω for M.
- Let H_1 generates "halt" if H_1 determines that the turing machine, M stops after accepting the input, ω.
- Otherwise H_1 loops forever when, M doesn‟t stops on processing ω.

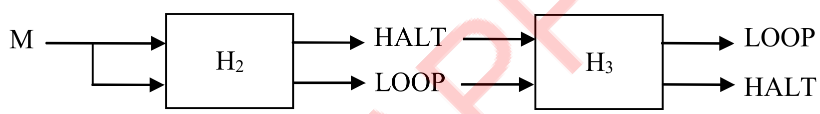
Construction of H_2



H_2 is constructed with both the inputs being M.

H_2 determines M and halts if M halts otherwise loops forever.

Construction of H_3

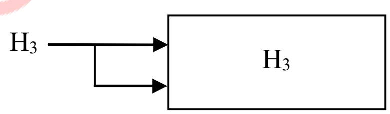


Let H_3 be constructed from the outputs of H_2.

If the outputs of H_2 are HALT, then H_3 loops forever.

Else, if the output of H_2 is loop forever, then H_3 halts.

Thus H_3 acts contractor to that of H_2.



- Let the output of H_3 be given as input to itself.
- If the input is loop forever, then H_3 acts contradictory to it, hence halts.
- And if the input is halt, then H_3 loops by the construction.
- Since the result is incorrect in both the cases, H_3 doesnot exist.
- Thus H_2 doesnot exist because of H_3.
- Similarly H_1 doesnot exist, because of H_2.

Thus halting problem is undecidable.

PARTIAL SOLVABILITY

Problem types

There are basically three types of problems namely

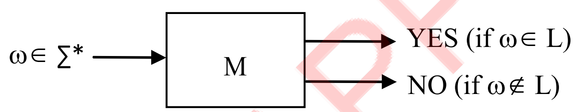
- Decidable / solvable / recursive
- Undecidable / unsolvable
- Semi decidable / partial solvable / recursively enumerable

Decidable / solvable problems

A problem, P is said to be decidable if there exists a turing machine, TM that decides P.

Thus P is said to be recursive.

Consider a Turing machine, M that halts with either „yes‟ or „no‟ after computing the input.



The machine finally terminates after processing

It is given by the function,

$$F_p(\omega) = \begin{cases} 1 & \text{if } p(\omega) \\ 0 & \text{if } \neg p(\omega) \end{cases}$$

The machine that applies $F_p(\omega)$ is said to be turing computable.

Undecidable problem

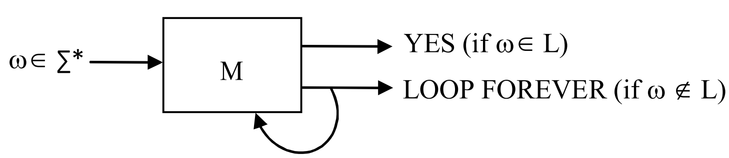
A problem, P is said to be undecidable if there is a Turing machine, TM that doesn‟t decides P.

Semi decidable / partial solvable / recursively enumerable

A problem, P is said to be semi decidable, if P is recursively enumerate.

A problem is RE if M terminates with „YES‟ if it accepts $\omega \in L$; and doesn‟t halt if $\omega \notin L$.

Then the problem is said to be partial solvable / Turing acceptable.



Partial solvability of a machine is defined as,

$$F_p(\omega) = \begin{cases} 1 & \text{if } p(\omega) \\ \text{undefined} & \text{if } \neg p(\omega) \end{cases}$$

Enumerating a language

Consider a k-tape turing machine. Then the machine M enumerates the language L (such that $L \subseteq \sum^*$) if

- The tape head never moves to the left on the first tape.
- No blank symbol (B) on the first tape is erased or modified.
- For all $\omega \in L$, where there exists a transition rule, δ_i on tape 1 with contents

$$\omega_1 \# \omega_2 \# \omega_3 \# \ldots \# \omega_n \quad \# \omega \# \quad (\text{for } n \geq 0)$$

Where $\omega_1, \omega_2, \omega_3, \ldots, \omega_n, \omega$ are distinct elements on L.

If L is finite, then nothing is printed after the # of the left symbol

That is,

- If L is a finite language then the TM, M either
  - Halts normally after all the elements appear on the first tape (elements are processed)

    or

  - Continue to process and make moves and state changes without scanning/printing other string on the first tape.

If the language, L is finite, the Turing machine runs forever.

Theorem

A language $L \subseteq \sum^*$ is recursively enumerable if and only if L can be enumerated by some TM.

Proof

Let M_1 be a Turing machine that enumerates L.

And let M_2 accepts L. M_2 can be constructed as a k-tape Turing machine [$k(M_2) > k(M_1)$].

M_2 simulates M_1 and M_1 pauses whenever M_2 scans the „#" symbol.

M_2 compares its input symbols to that of the symbols before „#“ while, M_1 is in pause.

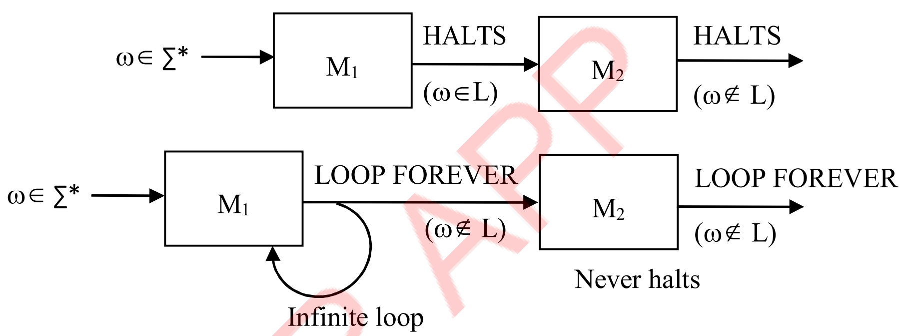
If the comparison finds a match of the string, M_2 accepts L.

Here M_2 is a semi acceptor TM for L

- Scans the input string, ω
- Runs the transition rules of M_1
- If M_1 outputs ω, then ω is accepted and M_1 hats

If $\omega \in L$, M_1 will output ω and M_2 will eventually accept „ω“ and halts.

If $\omega \notin L$, then M_1 will never provide an output ω and so M_2 will never halt.



Thus M_2 is partially solvable / Turing acceptable for L.

POST CORRESPONDENCE PROBLEM (PCP)

Post correspondence problem, known as PCP is an unsolvable combinatorial problem. This Undecidable problem was formulated by Emil Post in 1946.

A PCP consists of two lists of string over some alphabet Σ; the two lists must be of equal length. Generally A=w1,w2, w3, …….wk and B= x1, x2 ,x3, ……xk for some integer k. For each i , the pair (wi , xi) is said to be a corresponding pair.

We say this instances of PCP has a solution, if there is a sequence of one or more integers i1, i2,……., im that, when interpreted as indexes for strings in the A and B lists, yield the same string.

wi1 wi2 ……. wim = xi1 xi2 ……. xim . We say the sequence i1, i2,……., im is a solution to this instance of PCP

EXAMPLE

1. **For $\Sigma = \{a, b\}$ with $A = \{a, aba^3, ab\}$ and $B = \{a^3, ab, b\}$, Does the PCP with A and B have a solution?**

Solution:

The sequence obtained from A and B = (2, 1, 1, 3) as,

A_2	A_1	A_1	A_3
aba^3	a	a	ab
B_2	B_1	B_1	B_3
ab	a^3	a^3	b

Thus $A_2A_1A_1A_3 = B_2B_1B_1B_3 = aba^3a^3b = aba^6b$

The PCP given has a solution (2,1,1,3) with the two lists of elements.

2. **Let $\Sigma = \{0, 1\}$. Let A and B be the lists of three strings defined as**

	A	B
I	w_i	x_i
1	1	111
2	10111	10
3	10	0

Solution:

Consider the sequence (2, 1, 1, 3)

$A_2A_1A_1A_3 \Rightarrow w_2w_1w_1w_3 = 101111110$

$B_2B_1B_1B_3 \Rightarrow X_2X_1X_1X_3 = 101111110$

Thus the PCP has (2, 1, 1, 3) sequences as solution

The Diagonalization Language *L_d*

We define Ld, the diagonalization language, as follows:

Let w1, w2, w3, . . . be an enumeration of all binary strings.

Let M1, M2, M3, . . . be an enumeration of all Turing machines.

Let Ld = { Wi | Mi does not accept Wi }.

The language Ld, the diagonalization language, is the set of strings Wi such that Wi is not in L(Mi). That is, Ld consists of all strings w such that the TM (M) does not accept when given w as input.

Theorem: Ld is not a recursively enumerable language.

Proof:

Suppose Ld = L(Mi) for some TM Mi.

This gives rise to a contradiction. Consider what Mi will do on an input string wi.

If Mi accepts wi, then by definition wi cannot be in Ld.

If Mi does not accepts wi, then by definition wi is in Ld.

we must conclude there is no Turing machine that can define Ld.

Hence Ld is not a recursively enumerable language.

UNIVERSAL TURING MACHINE

Motive of UTM

A single Turing machine has a capability of performing a function such as addition, multiplication etc.

For computing another function, other appropriate Turing machine is used. To do so, the machine has to be re-written accordingly.

Hence Turing proposed “Stored Program Computer” concept in 1936 that executes the program/instructions using the inputs, stored in the memory.

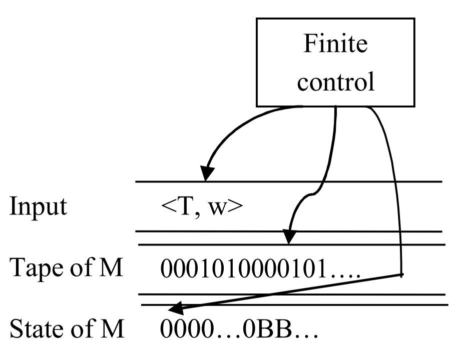
The instructions and inputs are stored on one or more tapes.

Concept of UTM

The universal Turing machine, T_u takes over the program and the input set to process the program.

The program and the inputs are encoded and stored on different tapes of a multi-tapeTuring machine.

The T_u thus takes up T, w where T is the special purpose Turing machine that passesthe program in the form of binary string, w is the data set that is to be processed by T.



Input to the T_u

The universal Turing machine, T_u is always provided with the code for Transitions, e(T) and code for input, e(w) as

$$TM = e(T)e(w)$$

For example, if the input data, w=”baa”, then

$$e(w) = 10001001001$$

This e(w) will be appended to e(T) of T_u.

Construction of T_u

As in the figure for universal Turing machine, there are three tapes controlled by a finite control component through heads for each tape.

Tape -1 ⇒ Input tape and also serves as output tape. It contain e(T) e(w).

Tape-2 ⇒ Tape of the TM/Working tape during the simulation of TM

Tape -3 ⇒ State of the TM, current state of the T in encoded form.

Operation of UTM

Theorm :(Lu is Recursively enumerable)

(To prove this Theorem it is necessary to construct a turning machine that accepts Lu)

- UTM checks the input to verify whether the code for TM=<T,w> is a legitimate for some TM.
    - If the input is not accepted, UTM halts with rejecting, w
- Initialize the second tape to have e(w), that is to have the input, w in encoded form. Place the code of the initial state on the third tape and move the head of the finite state control on the first cell of second tape.
- To simulate a move of the Turing machine, UTM searches for the transition - $o^i 1o^j 1o^k 1o^l 1o^m$ on the first tape, with o^i (initial state/current state) on tape -3 and o^j (input symbol to be processed) on tape- 2.
- The state transition is done by changing the tape -3 content as o^k as in the transition.
- Replace o^j by o^l on tape-2 to indicate the input change.

- Depending on o^m [m=1 ⇒stop, m=2 ⇒Left, m=3 ⇒Right], move the head on tape-2 to the position of the next 1 to the left/right/stop accordingly
- If TM has no transition, matching the simulated state and tape symbol, then no transition will be found. This happens when the TM stops also.
- If the TM, T enters halt (accepting state), then UTM accepts the input, w

Thus for every coded pair <T, w>, UTM simulates T on w, if and only if T accepts the input string, w.

Thus U TM simulates M and accepts W.Thus Lu is recursively enumerable

Definition of Universal Language [L_u]

The universal language, L_u is the set of all binary strings$[\alpha]$, where α represents the ordered pair <T, w> where

T →Turing machine

w →any input string accepted by T

It can also be represented as α = e(T) e(w) **.**

Theorem

L_u is the recursively enumerable but not recursive .

Proof

From the definition and operations of UTM, we know that L_u is recursively enumerable.

L_u accepts the string w if it is processed by the TM,T. Else, rejects „w" and the machine doesn"t halts forever.

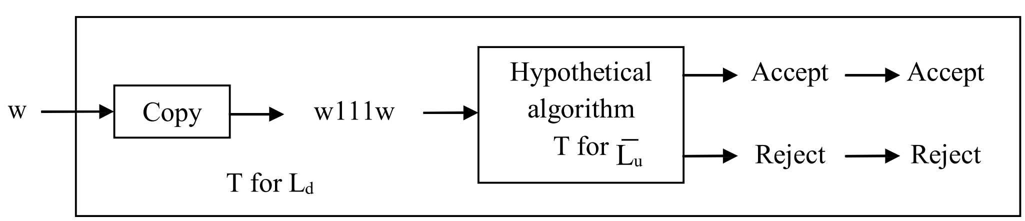
To prove that L_u is not recursive, the proof can be done by contradiction. Let L_u is Turing decidable [recursive], and then by definitionacceptable.

L_u (complement of L_u) is Turing

We can show that $\overline{L_u}$ is Turing acceptable, that leads to L_d to be Turing acceptable. But we know that L_d is not Turing acceptable.

Hence L_u is not Turing decidable by proof by contradiction.

Proof on $\overline{L_u}$ is during acceptable ⇒ L_d is Turing acceptable



Suppose "A" is the algorithm that recognizes L_u .

Then $\overline{L_d}$ is recognizes as follows. Given a string $w \in (0,1)^*$ determined easily, the value of I such that $w = w_i$.

Integer value, I in binary is the corresponding code for TM, T_i. Provide <T_i, w_i> to the algorithm A and accept, w if and only if T_i accepts w_i.

So the algorithm accepts w if and only if $w = w_i$ which is in $L(T_i)$.

This is the algorithm for L_d. Hence L_u is Recursively Enumerable but not recursive.

TRACTABLE AND INTERACTABLE PROBLEMS

Tractable Problems/Languages

The languages that can be recognized by a Turing machine in finite time and with reasonable space constraint is said to be tractable.

Example: If the language $L_1 \in$ Time (f), then L is tractable and is less complex in nature

Example: If $L_2 \notin$ Time (f), L2 is complex and cannot be tractable in limited time.

Tractable problems are those that can be solved in polynomial time period.

Intractable Problems

The languages that cannot be recognized by any Turing machine with reasonable space and time constraint is called intractable problems.

These problems cannot be solved in finite polynomial time. Even problems with moderate input size cannot achieve feasible solution

P AND NP PROBLEMS

These refer to how long it takes a program to run. Problems in class P can be solved with algorithms that run in **polynomial time.**

An algorithm that finds the smallest integer in an array. One way to do this is by iterating over all the integers of the array and keeping track of the smallest number you've seen up to that point. Every time you look at an element, you compare it to the current minimum, and if it's smaller, you update the minimum.

How long does this take? Let's say there are n elements in the array. For every element the algorithm has to perform a constant number of operations. Therefore we can say that the algorithm runs in O(n) time, or that the runtime is a linear function of how many elements are in the array. So this algorithm runs in **linear time.**

You can also have algorithms that run in **quadratic time** (O(n^2)), **exponential time** (O(2^n)), or even **logarithmic time** (O(log n)). Binary search (on a balanced tree) runs in logarithmic time because the height of the binary search tree is a logarithmic function of the number of elements in the tree.

If the running time is some polynomial function of the size of the input, for instance if the algorithm runs in linear time or quadratic time or cubic time, then we say the algorithm runs in **polynomial time** and the problem it solves is in class **P**.

NP

There are a lot of programs that don't (necessarily) run in polynomial time on a regular computer, but do run in polynomial time on a nondeterministic Turing machine. These programs solve problems in NP, which stands for nondeterministic polynomial time. A nondeterministic Turing machine can do everything a regular computer can and more. This means all problems in P are also in NP.

An equivalent way to define NP is by pointing to the problems that can be verified in polynomial time. This means there is not necessarily a polynomial-time way to find a solution, but once you have a solution it only takes polynomial time to verify that it is correct.

P = NP, which means any problem that can be verified in polynomial time can also be solved in polynomial time and vice versa. If they could prove this, it would revolutionize computer science because people would be able to construct faster algorithms for a lot of important problems.

NP-hard

Solve a problem by reducing it to a different problem. Reduce Problem B to Problem A if, given a solution to Problem A, It can easily construct a solution to Problem B. (In this case, "easily" means "in polynomial time.")

If a problem is **NP-hard**, this means , reduce any problem in NP to that problem. It can solve that problem, I can easily solve any problem in NP. If we could solve an NP-hard problem in polynomial time, this would prove P = NP.

NP-complete

A problem is **NP-complete** if the problem is both

- NP-hard, and
- in NP.

A technical point: O(n) actually means the algorithm runs in *asymptotically* linear time, which means the time complexity approaches a line as n gets very large. Also, O(n) is technically an upper bound, so if the algorithm ran in sublinear time you could still say it's O(n), even if that's not the best description of it.

** Note that if the input has many different parameters, like n and k, it might be polynomial in n and exponential in k