15CS301 THEORY OF COMPUTATION

UNIT-1 FINITE AUTOMATA

- Introduction : Basic Mathematical Notation and techniques
- Finite state systems, Basic definitions, Finite automaton : DFA
- Finite Automaton : NDFA, Finite Automaton with ϵ- moves
- Regular Languages - Regular Expressions
- Equivalence of NFA and DFA
- Equivalence of NDFA's with and without ϵ-moves
- Equivalence of finite automaton and regular expressions
- Minimization of DFA .
- pumping lemma for regular sets, problems based on pumping lemma.

Introduction:

⇒ Theory of computation is the branch that deals with how efficiently problems can be solved on a model of computation using ~~using~~ an algorithm.

⇒ The field is divided into three major branches

1. Automata theory
2. Computability theory
3. Computational Complexity Theory.

Introduction to formal proof

The formal proof can be using deductive proof and inductive proof.

1) Deductive proof ⇒ Sequence of statements given with logical reasoning in order to prove the first or initial statement.

Initial statement - Hypothesis.

2) Inductive proof ⇒ Recursive kind of proof which consists of sequence of parameterized statements that use the statement itself with lower value of its parameter.

Example: Statement B is true because statement A is true

Here Statement A ⇒ Hypothesis

Statement B ⇒ Conclusion

Various forms of proof:

1) proofs about sets ⇒ Set is a collection of elements or items.

Thus proof is of the kind "if and only if" that means an elemen
X is in A if and only if it is in B.

Example: Let us prove $P \cup Q = Q \cup P$

L.H.S ⇒

Statement	Justification
X is in $P \cup Q$	Given
X is in P or X is in Q	1) And by definition of union
X is in Q or X is in P	2) And by definition of union
X is in $Q \cup P$	3) Rule 2) And by definition of union

RHS ⇒

Statement	Justification
X is in $Q \cup P$	Given
X is in Q or X is in P	1) And by definition of union
X is in P or X is in Q	2) And by definition of union
X is in $P \cup Q$	3) Rule 2) And by definition of union

Hence $P \cup Q = Q \cup P$. Thus A=B is true as element X is in B if and only if X is in A.

2) Proof by Contradiction ⇒ The statement of the form if A and B we start with statement A is not true and thus by assuming false A we try to get the conclusion of statement B.

Example: prove $P \cup Q = Q \cup P$

Proof:

Assume $P \cup Q = Q \cup P$ is not true

(ie) $P \cup Q \neq Q \cup P$

Consider X is in Q or X is in P ⇒ X is in PUQ then

X is in QUP according to definition of Union

Hence, The assumption which made initially is false.

Thus, PUQ = QUP is proved.

3) Proof by counter example

"All possible conditions" in which the statement remains true

Ex: There is no such pair of integers such that

$a \bmod b = b \bmod a$

Proof: Consider a=2 and b=3, then

$2 \bmod 3 \neq 3 \bmod 2$, thus the given pair is true for any pair of integers but if a=b then naturally $\boxed{a \bmod b = b \bmod a,\ a = b}$

Inductive proof:

⇒ Inductive proofs are special proof based on some observations.

⇒ It is used to prove recursively defined objects.

The proof by mathematical induction can be carried out using following steps.

1. Basis ⇒ In this step we assume the lowest possible value. This is an initial step in the proof of mathematical induction.

   (ie) prove the result is true for n=0 or n=1

2. Induction hypothesis ⇒ In this step we assign value of n to some other value k.

   (ie) Check whether the result is true for n=k or not.

3. Inductive step ⇒ In this step, if n=k is true, then we check whether the result is true for n=k+1 or not.

   If we get the same result at n=k+1 then we can state that given proof is true by principle of mathematical Induction.

Problem1 : prove by induction on 'n' that $\sum_{i=0} i = \frac{n(n+1)}{2}$

Solution:

1) Basis of induction

Assume n=1, Then $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$

$$L.H.S = \sum_{i=0}^{1} 1 = 1$$

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

2) Induction hypothesis

Assume n=k, then $1+2+3+\ldots+k = \frac{k(k+1)}{2}$ is true.

3) Inductive step

Assume n=k+1,

N.K.T $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$

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

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

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

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

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

$$LHS = \frac{k^2 + 3k + 2}{2} \quad \text{——} (1)$$

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

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

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

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

$$RHS = \frac{k^2 + 3k + 2}{2} \quad \text{——} (2)$$

From (1) and (2) LHS = RHS

$$\therefore \sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

Hence, the given hypothesis is proved.

Problem 2: Prove that $1^2+2^2+3^2+\cdots+n^2=\sum_{i=1}^{n} i^2=\frac{n(n+1)(2n+1)}{6}$ using mathematical induction.

Solution:

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

Let $n=1$, $LHS=1^2=1$

$$RHS=\frac{n(n+1)(2n+1)}{6}=\frac{1(1+1)(2(1)+1)}{6}=\frac{1\times2\times3}{6}=1$$

$LHS=RHS$

2) **Induction hypothesis**

Let $n=k$, $1^2+2^2+3^2+\cdots+k^2=\sum_{i=1}^{k} i^2=\frac{k(k+1)(2k+1)}{6}$ is true.

3) **Inductive step**

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

$$LHS=\sum_{i=1}^{k+1} i^2$$

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

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

$$=\frac{(k^2+k)(2k+1)+6k^2+12k+6}{6}$$

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

$$LHS=\frac{2k^3+9k^2+13k+6}{6} \quad \text{——①}$$

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

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

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

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

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

$$RHS=\frac{2k^3+9k^2+13k+6}{6} \quad \text{——②}$$

From ① and ② $LHS=RHS$

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

Hence, the given statement is proved using mathematical induction.

Problem 3: Prove that for every integer $n \geq 0$ the number $4^{2n+1} + 3^{n+2}$ is multiple of 13.

Solution:

1) Basis of induction, Let $n=0$

$$4^{2n+1} + 3^{n+2} = 4^{2(0)+1} + 3^{0+2}$$
$$= 4 + 3^2$$
$$= 4 + 9$$
$$= 13$$

$4^{2n+1} + 3^{n+2}$ = Multiple of 13

2) Induction hypothesis

Let $n=k$, $4^{2k+1} + 3^{k+2}$, for all $k \geq 0$ it is multiple of 13.

3) Inductive step.

Let $n=k+1$, $4^{2n+1} \cdot 3^{n+2} = 4^{2(k+1)+1} \cdot 3^{k+1+2}$

$$= 4^{2k+3} + 3^{k+3}$$
$$= 4^{2k+1} \cdot 4^2 + 3^{k+2} \cdot 3^1$$
$$= 16\left(4^{2k+1}\right) + 3\left(3^{k+2}\right)$$
$$= 13\left(4^{2k+1}\right) + 3\left(4^{2k+1}\right) + 3\left(3^{k+2}\right)$$
$$= 13\left(4^{2k+1}\right) + 3\left(4^{2k+1} + 3^{k+2}\right)$$
$$= 13\left(4^{2k+1}\right) + 3\,(13k)$$
$$= 13\left(4^{2k+1} + 3k\right)$$

$4^{2n+1} + 3^{n+2}$ = Multiple of 13

$\therefore$ For every integer $n \geq 0$, the number $4^{2n+1} + 3^{n+2}$ is multiple of 13.

Problem 4: Prove that for every integer $n \geq 0$ the number $n^4 - 4n^2$ is Divisible by 3.

Problem 5: Prove that every tree has 'e' edges and 'e+1' nodes.

Basic concept of Automata theory

- Automata theory has a basic fundamental unit called set.
- Set is used to represent the mathematical model.

Set => It is defined as collection of objects enclosed within { }.

Ex: A set of element which are less than 5,

$A = \{0, 1, 2, 3, 4\}$

Set of vowels $A = \{a, e, i, o, u\}$

Recursion => To define the elements of the set $A = \{x \mid x \text{ is a square of } n\}$ where $n \leq 10$

(ie) $A = \{0, 1, 4, 9, 16, \ldots\ldots 100\}$

Subset => Subset A is called subset of B if every element of set A is present in set B, denoted by $A \subseteq B$

Ex: $A = \{1, 2, 3\}$ and $B = \{1, 2, 3, 4, 5\}$ then $A \subseteq B$.

Empty set => Set having no element, $A = \{\}$ written as ϕ (phi).

Null String => Null element is denoted by ε

Power set => Set of all the subsets of its elements

Ex: $A = \{1, 2, 3\}$ power set $Q = \{\phi, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{3, 2\}, \{2, 3\}, \{1, 2, 3\}\}$

Operation on set =>

(i) **Union** => Combination of both sets.

$A = \{1, 2, 3\}$, $B = \{1, 2, 4\}$ then $A \cup B = \{1, 2, 3, 4\}$

(ii) **Intersection** => Common elements from both the sets

$A = \{1, 2, 3\}$, $B = \{1, 2, 4\}$ then $A \cap B = \{2\}$

(iii) **Difference** => Elements there in one set but not there in another

$A = \{1, 2, 3\}$, $B = \{1, 2, 4\}$ then $A - B = \{3\}$.

(iv) **Complement** => $\overline{A} = U - A$. $U = \{10, 20, 30, 40, 50\}$, $A = \{10, 20\}$

$\overline{A} = U - A = \{30, 40, 50\}$

Cartesian product => Cartesian product of two sets A and B is a set of all possible ordered pairs.

Ex: $A = \{a, b\}$ and $B = \{0, 1, 2\}$

$A \times B = \{\{a, b\} \mid a \in A, b \in B\}$ $A \times B = \{(a,0), (a,1), (a,2), (b,0), (b,1), (b,2)\}$

Relations => The relation R is a collection for the set S which represents the pair of elements.

Properties: If $A = \{a, b\}$ then,

Reflexive relation R can be $= \{(a,a), (b,b)\}$

Irreflexive relation R can be $= \{(a,b)\}$

Transitive relation R can be $= \{(a,b), (b,a), (a,a)\}$

Symmetric relation R can be $= \{(a,b), (b,a)\}$

Asymmetric relation R can be $= \{(a,b)\}$

Equivalence relation => A relation is said to be equivalence relation if it is, reflexive, symmetric and transitive.

Kleen closure => set of strings of any length including null string ξ.

Ex: let $\Sigma = \{a, b\}$ obtain Σ^*

$\Sigma^* = \{\xi, a, b, aa, bb, ab, ba, aaa, bbb, aba, \ldots\}$

Positive closure => consists of all the strings of any length except a null string.

Ex: let $\Sigma = \{a, b\}$ $\Sigma^+ = \{a, b, aa, bb, ab, ba, aaa, bbb, aba, \ldots\}$

Alphabets, Strings and Languages

Alphabet: An alphabet is a finite collection of symbols.

Ex: $S = \{a, b, c, \ldots z\}$, $W = \{0, 1\}$ Here 0, 1 are alphabets, denoted by W.

Strings: Finite collection of symbols from alphabet.

Ex: $\Sigma = \{a, b\} = \{ab, bb, aa, aaa, bbb, \ldots\}$

Language: It is a collection of appropriate strings. The language is defined using an input set.

Ex: $\Sigma = \{\xi, 0, 00, 000, \ldots\}$

Here, the language is defined as 'any number of zeros'.

Operations => Union $L_1 \cup L_2$, Intersection $L_1 \cap L_2$

Concatenation $L_1 L_2$, Difference $L_1 - L_2$.

Example: Describe the following language over the input set $A=\{a,b\}$

(i) $L_1=\{a, ab, ab^2\}$ ⇒ All the strings with letter a followed by any number of b's.

(ii) $L_2=\{a^n b^n \mid n \geq 1\}$ ⇒ Strings having equal number of a's followed by equal number of b's.

(iii) $L_3=\{a^m b^n \mid n>0\}$ ⇒ All the strings with any numbers of a's followed by atleast one b.

Application of Automata Theory

1) It is the base for the formal languages and useful for programming languag
2) It plays an important role in compiler Design
3) To prove the correctness of the program automata theory is used.
4) Switching theory and design and analysis of digital circuits automata Theory is applied.
5) Automata theory deals with the design finite state machine.

Finite State Systems

⇒ Finite state system represents a mathematical model of a system with certain input. The model finally gives certain output.

⇒ The input when is given to the machine it is processed by various states, These states are called as intermediate states.

Example: Control mechanism of elevator. This mechanism only remembers the current floor number pressed, it does not remember all the previously pressed numbers.

Finite Automata - Definition

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

Q - finite set of states

Σ - input alphabet

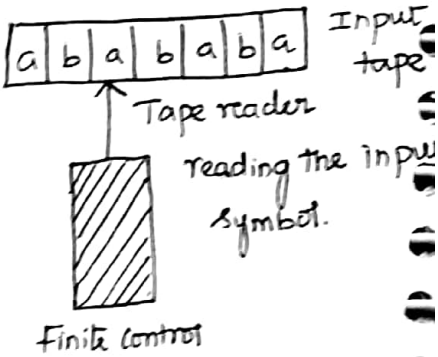
q_0 - initial state and q_0 is in Q (ie) $q_0 \in Q$

F - Final states

δ - Transition function - Used to determine next state.

Finite Automata Model.

The finite automata can be represented using



(i) Input tape => Linear tape having number of cells. Each i/p symbol is placed in each cell.

(ii) Finite control => It decides the next state on receiving particular i/p from input tape.

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

Acceptance of Strings and Languages

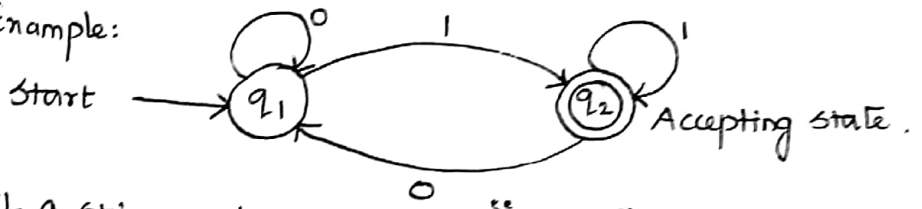
The strings and languages can be accepted by a finite automata, when it reaches to a final state.

Notations => (i) Transition diagram (ii) Transition table.

Transition diagram => Directed graph associated with the vertices of the graph correspond to the states of the finite automata.

Start State => $\rightarrow q_1$ Final state => (q_2)

Example:



If a string ends in a 0, it is "rejected" and "accepted" only if the string ends in 1 $\therefore$ The language $L(M) = \{ w \mid w \text{ ends in } 1 \}$.

Transition table => Tabular representation of finite automata.

Input / States	0	1
$\rightarrow q_1$	q_1	q_2
$*q_2$	q_1	q_2

Language Acceptance by FA:

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

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

DFA (Deterministic Finite Automata)

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

A Deterministic finite automata is a Quintuple

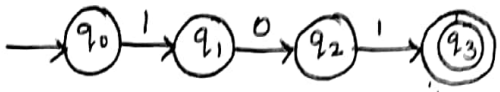
$$M = (Q, \Sigma, \delta, q_0, F)$$

Q - Finite set of states
Σ - Finite set of i/p symbols
δ - Transition function
q_0 - start state, $q_0 \in Q$
F - Final state, $F \subseteq Q$

Problems:

1) Design an FA which accepts the only input 101 over the input set $\Sigma = \{0,1\}$

Solution:



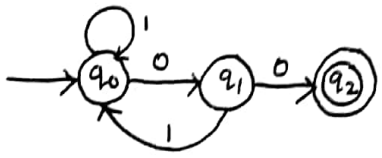
Here $M = (\{q_0, q_1, q_2, q_3\}, \{0,1\}, \delta, q_0, \{q_3\})$

δ: $\delta(q_0,1) = q_1$ $\quad \delta(q_2,1) = q_3$
$\delta(q_1,0) = q_2$

2) Design DFA for a) set of all strings ending in 00
b) Set of all strings with three consecutive Xeros.

Solution:

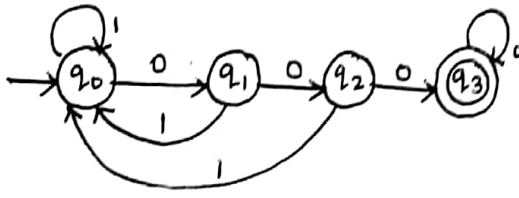
a)



Here $M = (\{q_0, q_1, q_2\}, \{0,1\}, \delta, q_0, \{q_2\})$

δ: $\delta(q_0,0) = q_1$ | $\delta(q_1,0) = q_2$
$\delta(q_0,1) = q_0$ | $\delta(q_1,1) = q_0$

b)

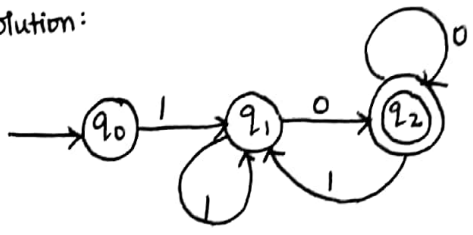


Here $M = (\{q_0, q_1, q_2, q_3\}, \{0,1\}, \delta, q_0, \{q_3\})$

δ: $\delta(q_0,0) = q_1$ | $\delta(q_1,0) = q_2$ | $\delta(q_2,0) = q_3$ | $\delta(q_3,0) = q_3$
$\delta(q_0,1) = q_0$ | $\delta(q_1,1) = q_0$ | $\delta(q_2,1) = q_0$ | $\delta(q_3,1) = q_3$

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

Solution:

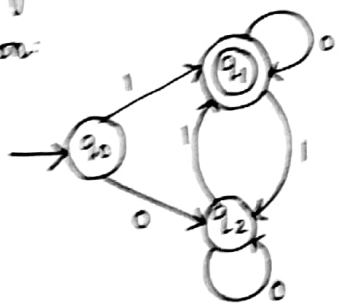


Here $M = (\{q_0, q_1, q_2\}, \{0,1\}, q_0, \delta, \{q_2\})$

δ: $\delta(q_0,1) = q_1$ | $\delta(q_2,0) = q_2$
$\delta(q_1,0) = q_2$ | $\delta(q_2,1) = q_1$
$\delta(q_1,1) = q_1$ |

4) Design FA which accepts odd number of 1's and any number of 0's.

Solution:



Here $M = (\{q_0, q_1, q_2\}, \{0,1\}, q_0, \delta, \{q_1\})$

δ: $\delta(q_0,0) = q_1$ | $\delta(q_1,0) = q_1$ | $\delta(q_2,0) = q_2$

$\delta(q_0,1) = q_2$ | $\delta(q_1,1) = q_2$ | $\delta(q_2,1) = q_1$

Testing: 10101 $\Rightarrow$ 1 0 1 0 1

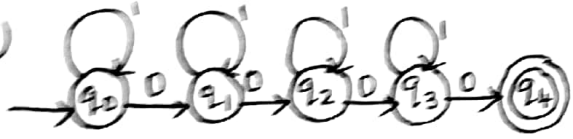
$$\begin{aligned}
\delta(q_0, 10101) &= \delta(\delta(q_0,1), 0101) \\
&= \delta(q_1, 0101) \\
&= \delta(\delta(q_1,0), 101) \\
&= \delta(q_1, 101) \\
&= \delta(\delta(q_1,1), 01) \\
&= \delta(q_2, 01) \\
&= \delta(\delta(q_2,0), 1) \\
&= \delta(q_2, 1) \\
&= q_1 \text{ Accepting state.}
\end{aligned}$$

5) Design DFA with a) All string that contain exactly 4 zeros.

b) All string that doesn't contain the substring 110.

Solution:

a)



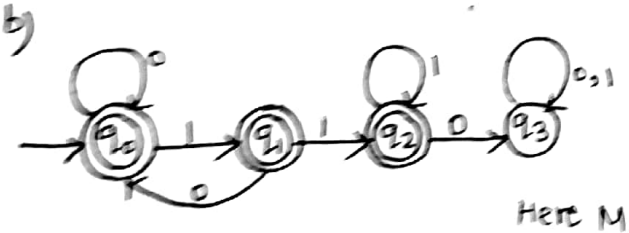
Here $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0,1\}, q_0, \delta, \{q_4\})$

δ: $\delta(q_0,0) = q_1$ | $\delta(q_1,0) = q_2$ | $\delta(q_2,0) = q_3$

$\delta(q_0,1) = q_0$ | $\delta(q_1,1) = q_1$ | $\delta(q_2,1) = q_2$

$\delta(q_3,0) = q_4$, $\delta(q_3,1) = q_3$

b)



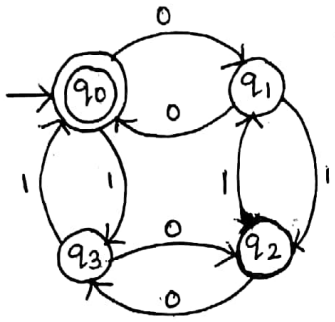
Here $M = (\{q_0, q_1, q_2, q_3\}, \{0,1\}, q_0, \delta, \{q_0, q_1, q_2\})$

δ: $\delta(q_0,0) = q_0$ | $\delta(q_1,0) = q_0$ | $\delta(q_2,0) = q_3$ | $\delta(q_3,0) = q_3$

$\delta(q_0,1) = q_1$ | $\delta(q_1,1) = q_2$ | $\delta(q_2,1) = q_2$ | $\delta(q_3,1) = q_3$

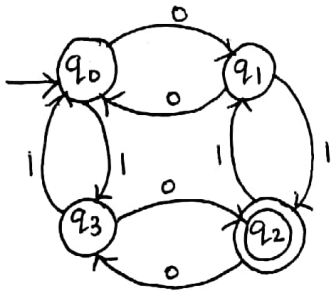
b) Design FA which accepts i) even number of 0's and even number of 1's.
ii) odd number of 0's and odd number of 1's.
iii) even number of 0's and odd number of 1's.
iv) odd number of 0's and even number of 1's.

(i)



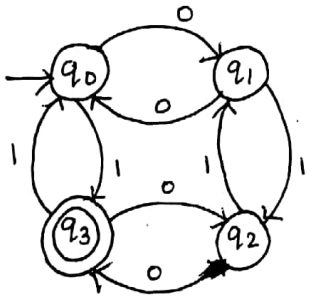
$\delta(q_0, 0011) = \delta(\delta(q_0,0), 011)$
$= \delta(q_1, 011)$
$= \delta(\delta(q_1,0), 11)$
$= \delta(q_0, 11)$
$= \delta(\delta(q_0,1), 1)$
$= \delta(q_3, 1) = q_0$ Accepted.

(ii)



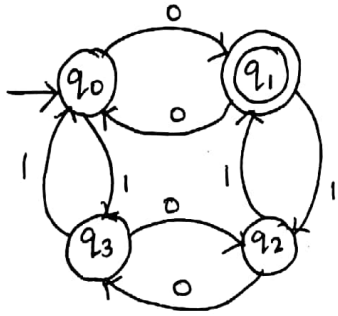
$\delta(q_0, 101010) = \delta(\delta(q_0,1), 01010)$
$= \delta(q_3, 01010)$
$= \delta(\delta(q_3,0), 1010)$
$= \delta(q_2, 1010)$
$= \delta(\delta(q_2,1), 010)$
$= \delta(q_1, 010)$
$= \delta(\delta(q_1,0), 10)$
$= \delta(q_0, 10) = \delta(\delta(q_0,1), 0) = \delta(q_3, 0)$
$= q_2$ Accepted.

(iii)



$\delta(q_0, 010) = \delta(\delta(q_0,0), 10)$
$= \delta(q_1, 10)$
$= \delta(\delta(q_1,1), 0)$
$= \delta(q_2, 0) = q_3$ Accepted.

(iv)



$\delta(q_0, 101) = \delta(\delta(q_0,1), 01)$
$= \delta(\delta(q_3,0), 1)$
$= \delta(q_2, 1)$
$= q_1$ Accepted.

7) Design FA for the following

a) Number of 1's in W is 3 mod 4

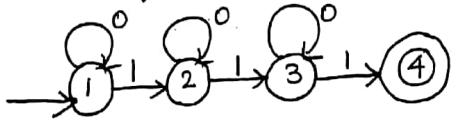
b) $L = \{0^n \mid n \bmod 3 = 2\}$

Solution:

a) Consider W = 010010001

No of 1's = 3 (ie) 3 mod 4 = 3

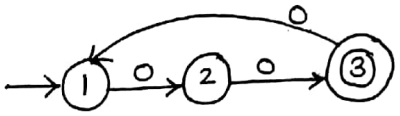
The no. of 1's should be exactly 3.



b) $L = \{0^n \mid n \bmod 3 = 2\}$

Here 1 mod 3 = 1
2 mod 3 = 2 → Accepted ← 5 mod 3 = 2
3 mod 3 = 0

$L = \{00, 0000, 00000000, \ldots\}$



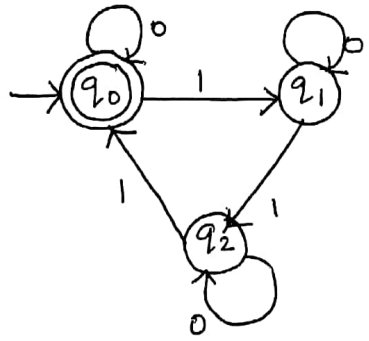
8) Construct DFA for the following

a) Number of 1's is a multiple of 3.

b) Number of 1's is not a multiple of 3.

Solution

a)



$\delta(q_0, 1110) = \delta(\delta(q_0, 1), 110)$

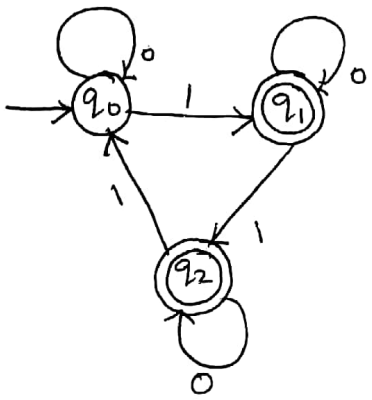
$= \delta(\delta(q_1, 1), 10)$

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

$= \delta(q_0, 0)$

$= q_0$ Accepted $\Rightarrow$ No of 1's is multiple of 3

b)



$\delta(q_0, 11110) = \delta(\delta(q_0, 1), 1110)$

$= \delta(\delta(q_1, 1), 110)$

$= \delta(\delta(q_2, 1), 10)$

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

$= \delta(q_1, 0)$

$= q_1$ Accepted.

Here no. of 1's is not a multiple of 3.

NFA (Non-Deterministic Finite Automata)

The finite Automata is called NFA when there exists many paths for a specific input from current state to next state.

A non-deterministic finite Automata is defined by,

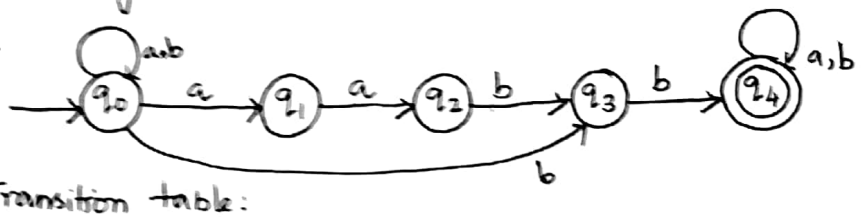
$$M = (Q, \Sigma, \delta, q_0, F)$$

Where

Q - Finite set of internal states
Σ - Finite set of I/p symbols
q_0 - Start state, $q_0 \in Q$
δ - Transition function
F - Finite set of final states $F \subseteq Q$.

Example: Draw state transition diagram for FA over $\{a, b\}$ containing substring aabb.

Solution:

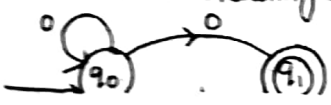


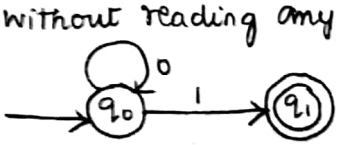
Transition table:

I/p / States	0	1
Start State → q_0	$\{q_0, q_1\}$	$\{q_0, q_3\}$
q_1	–	q_2
q_2	q_2	q_2
q_3	q_4	–
Final State ← q_4	q_4	q_4

DFA Vs NFA

	DFA	NFA
1.	Every i/p string leads to the unique state of finite automata.	For some i/p there can be more than one next states.
2.	Requires more memory for storing the state information.	Requires more computation time.
3.	It is not possible to move to next state without reading any symbol.	Possible to move to next state without reading any symbol.





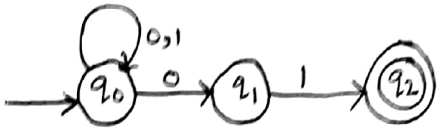
Extended Transition Function: $\hat{\delta}$

This is to represent transition functions with a string of input symbols 'w' and returns a set of states.

Suppose $w = xa$,

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

Example:



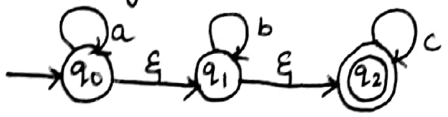
Process the input 001

$$\begin{aligned}
\hat{\delta}(q_0, 001) &= \delta(\hat{\delta}(q_0, 00), 1) \\
&= \delta(\delta(\hat{\delta}(q_0, 0), 0), 1) \\
&= \delta(\delta(\{q_0, q_1\}, 0), 1) \\
&= \delta((\delta(q_0, 0) \cup \delta(q_1, 0)), 1) \\
&= \delta(\{q_0, q_1\} \cup \phi), 1) \\
&= \delta(\{q_0 q_1\}, 1) \\
&= \delta(q_0, 1) \cup \delta(q_1, 1) \\
&= \{q_0\} \cup \{q_2\} \\
&= \{q_0, q_2\}
\end{aligned}$$

Finite Automaton with ε-moves

In the non-deterministic finite state machine, the transition that does not require input symbols for the state transition and is capable of transiting to zero or more states with ε is called epsilon transition.

Example: Find ε-closure for the following NFA.



ε-closure $(q_0) = \{q_0, q_1, q_2\}$

ε-closure $(q_1) = \{q_1, q_2\}$

ε-closure $(q_2) = \{q_2\}$

ε-closure

The ε-closure(p) is a set of all states which are reachable from state p on ε-moves such that

(i) ε-closure $(p) = p$ where $p \in Q$

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

Equivalence of NFA and DFA

Theorem: Let L be a set accepted by NFA, then there exists a DFA that accepts L

Proof:

This can be proved by subset construction method

Let $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$

$D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$

The states of D are all the subset of the set states of N.

$$Q_D = 2^{Q_N}$$

An element of Q_D will be denoted by $[q_1, q_2, \ldots q_i]$

where $q_1, q_2, \ldots q_i \in Q$

Define $\delta_D([q_1, q_2, \ldots q_i], a) = [P_1, P_2, \ldots P_j]$

if and only if

$\delta_N([q_1, q_2, \ldots q_i], a) = \{P_1, P_2, \ldots P_j\}$

δ_D is computed by applying δ_N to each states of δ_D represented by $[q_1, q_2, \ldots q_i]$

On applying to each of $q_1, q_2, \ldots q_i$ and taking the union.

we get, $[P_1, P_2, \ldots P_j]$ in Q_D

To show $\delta_D(\{q_0\}, x) = [q_1, q_2, \ldots q_j]$ if and only if

$\delta_N(q_0, x) = \{q_1, q_2, \ldots q_i\}$

This is proved by the method of induction.

F_D is the set of subset S of Q_N such that $S_n \cap F_n \neq \phi$.

Problem: Construct DFA equivalent to the NFA $M = (\{a, b, c, d\}, \{0, 1\}, \delta, a, \{b, d\})$ where δ is defined in the following table.

δ	0	1
a	{b, d}	b
b	c	{b, c}
c	d	a
d	ϕ	a

Solution:

(i) Find the states of $Q_N = \{a, b, c, d\}$

(ii) Construct subsets $Q_D = 2^{Q_N} = 2^4 = 16$

$$Q_D = \{ [a], [b], [c], [d], [a,b], [a,c], [a,d], [b,c], [b,d], [c,d], [a,b,c], [a,c,d], [b,c,d], [a,b,c,d], \phi \}$$

[a,b,d]

(iii) Find δ_D for all states

$\delta_D([a],0) = [b,d]$
$\delta_D([a],1) = [b]$
$\delta_D([b],0) = [c]$
$\delta_D([b],1) = [b,c]$
$\delta_D([c],0) = [d]$
$\delta_D([c],1) = [a]$
$\delta_D([d],0) = \phi$
$\delta_D([d],1) = [a]$

$\delta_D([a,b],0) = [b,c,d]$
$\delta_D([a,b],1) = [b,c]$

$\delta_D([a,c],0) = [b,d]$
$\delta_D([a,c],1) = [a,b]$
$\delta_D([a,d],0) = [b,d]$
$\delta_D([a,d],1) = [a,b]$
$\delta_D([b,c],0) = [c,d]$
$\delta_D([b,c],1) = [a,b,c]$
$\delta_D([b,d],0) = [c]$
$\delta_D([b,d],1) = [a,b,c]$
$\delta_D([c,d],0) = [d]$
$\delta_D([c,d],1) = [a]$

$\delta_D([a,b,c],0) = [b,c,d]$
$\delta_D([a,b,c],1) = [a,b,c]$
$\delta_D([a,c,d],0) = [b,d]$
$\delta_D([a,c,d],1) = [a,b]$
$\delta_D([b,c,d],0) = [c,d]$
$\delta_D([b,c,d],1) = [a,b,c]$
$\delta_D([a,b,c,d],0) = [b,c,d]$
$\delta_D([a,b,c,d],1) = [a,b,c]$

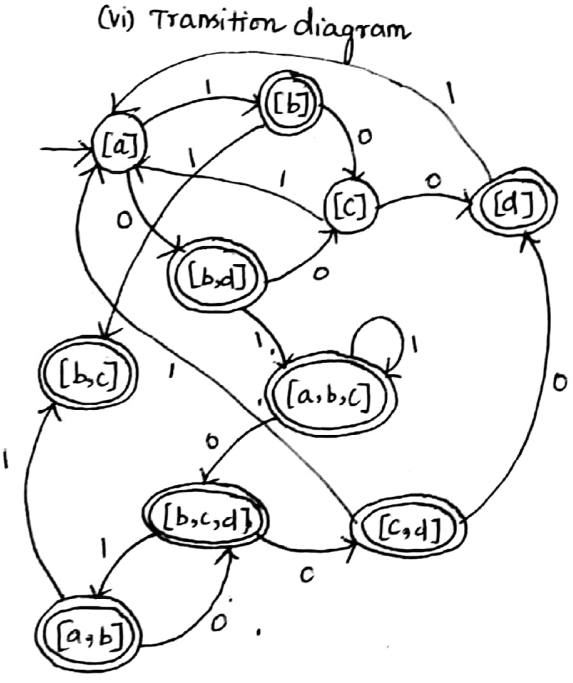
(iv) $F_N = \{b,d\}$ (Final state) $F_D \Rightarrow$ all set containing $\{b,d\}$

$$F_D = \{ [b], [d], [a,d], [b,d], [c,d], [a,b], [b,c], [a,b,c], [b,c,d], [a,b,d], [a,b,c,d] \}$$

(v) Transition table:

	0	1
[a]	[b,d]	[b]
* [b]	[c]	[b,c]
[c]	[d]	[a]
* [d]	ϕ	[a]
* [a,b]	[b,c,d]	[b,c]
[a,c]	[b,d]	[a,b]
* [a,d]	[b,d]	[a,b]
* [b,c]	[c,d]	[a,b,c]
* [b,d]	[c]	[a,b,c]
* [c,d]	[d]	[a]
* [a,b,c]	[b,c,d]	[a,b,c]
* [a,c,d]	[b,d]	[a,b]
* [b,c,d]	[c,d]	[a,b,c]
* [a,b,c,d]	[b,c,d]	[a,b,c]

(vi) Transition diagram



Equivalence of NDFA's with and without ϵ-moves

Theorem: If L is accepted by NFA with ϵ-transitions, then there exist L which is accepted by NFA without ϵ-transitions.

Proof:

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

Construct M' which is NFA without ϵ-transition.

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

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

By induction: $\begin{cases} \delta' \text{ and } \hat{\delta} \text{ are same} \\ \delta \text{ and } \hat{\delta} \text{ are different} \end{cases}$

Let x be any string $\delta'(q_0, x) = \hat{\delta}(q_0, x)$

This statement is not true if $x = \epsilon$ because,

$\delta'(q_0, \epsilon) = \{q_0\}$ and $\hat{\delta}(q_0, \epsilon) = \epsilon\text{-closure}(q_0)$.

Basis: $|x| = 1$, x is a symbol whose value is a $\delta'(q_0, a) = \hat{\delta}(q_0, a)$ [By definition of $\hat{\delta}$]

Inductive: Let $x = wa$ where a is in Σ

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

Now we must know that

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

But

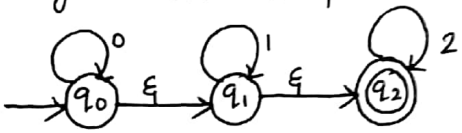
$$\delta'(P, a) = \bigcup_{q \text{ in } P} \delta'(q, a) = \bigcup_{q \text{ in } P} \hat{\delta}(q, a)$$

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

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

Hence proved.

Problem: Convert the given NFA with ε to NFA without ε.



Solution:

(i) Find ε-closure of all the states

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

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

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

(ii) Find δ' transitions for each state on each input symbol.

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

$= \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_0),0)$

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

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

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

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

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

$= \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_0),1)$

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

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

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

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

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

$= \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_1),0)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(iii) Summarize all the computed δ' transitions.

$\delta'(q_0,0) = \{q_0,q_1,q_2\}$ $\quad \delta'(q_1,2) = \{q_2\}$

$\delta'(q_0,1) = \{q_1,q_2\}$ $\quad \delta'(q_2,0) = \phi$

$\delta'(q_0,2) = \{q_2\}$ $\quad \delta'(q_2,1) = \phi$

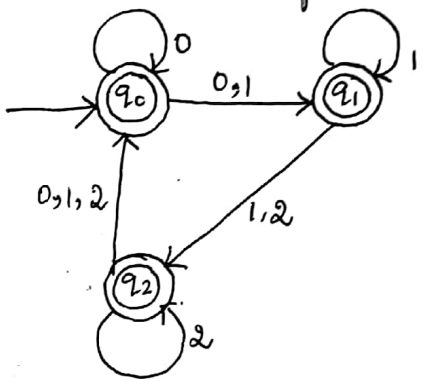
$\delta'(q_1,0) = \phi$ $\quad \delta'(q_2,2) = \{q_2\}$.

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

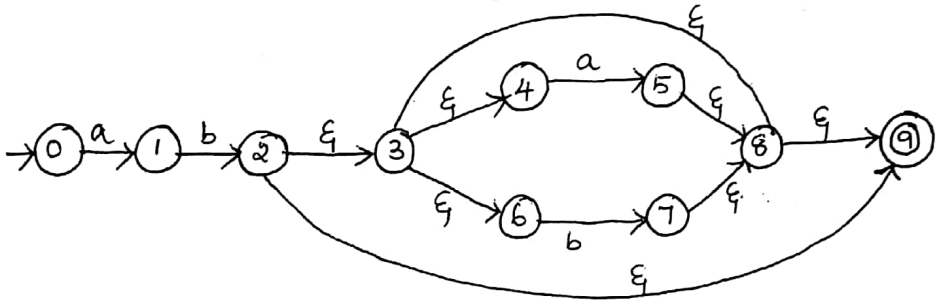
(iv) Transition table

Input / State	0	1	2
q_0	$\{q_0,q_1,q_2\}$	$\{q_1,q_2\}$	$\{q_2\}$
q_1	ϕ	$\{q_1,q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$

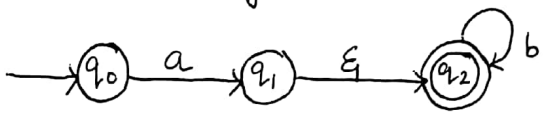
(v) Transition diagram.



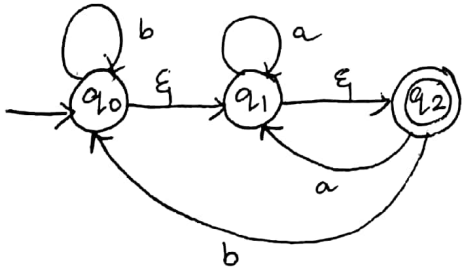
Problem2: Convert the following NFA with ε to NFA without ε.



Problem3: Convert the following NFA with ε to NFA without ε.



Problem4: Construct NFA with ε to ordinary NFA.



Regular Language - Regular Expression

⇒ Regular languages are those languages which are accepted by finite automaton

Ex: Set of strings of 0's and 1's $L = \{(0+1)^* \mid (0,1) \in \Sigma\}$

⇒ The languages accepted by finite automata are easily described by simple expression called regular expressions.

Properties: (closure)

1. ϕ is a regular expression which denotes the empty set.
2. ε is a RE and denotes the set $\{\varepsilon\}$ and it is a null string.
3. For each 'a' in Σ 'a' is a RE and denotes the set $\{a\}$
4. If r and s are regular expression denoting the language L_1 and L_2 respectively. Then
   - $r+s$ is equivalent to $L_1 \cup L_2$ (union)
   - rs is equivalent to $L_1 L_2$ (concatenation)
   - r^* is equivalent to L_1^* (closure)

Problem 1: Describe the following by regular expression.

a) L_1 = The set of all strings of 0's and 1's ending in 00

b) L_2 = The set of all strings of 0's and 1's beginning with 0 and ending with 1.

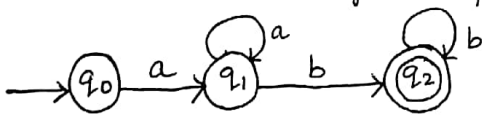
Solution

a) Regular Expression $L_1 = (0+1)^*00$

b) Regular expression $L_2 = 0(0+1)^*1$

Problem 2: Construct a DFA for the regular expression aa^*bb^*

Solution:



Problem 3: Find out the language generated by the regular expression $(0+1)^*$

Solution: $L = \{\varepsilon, 0, 1, 00, 11, 01, 10, 000, 111, \ldots\ldots\}$

$L = \{$Any number of 0's and 1's including null string$\}$

Problem 4: Give regular expression to describe an identifier and positive integer

Solution: identifier = (a-z A-Z)(a-z A-Z 0-9)*

Positive integer = $(0-9)^+$

Equivalence of finite Automaton and regular expressions

Every language defined by the regular expressions can be also be defined by the finite state automata.

Theorem: If $L = L(A)$ for some DFA, Then there is a regular expression R such that $L = L(R)$.

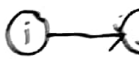
Proof: Let A be a DFA which defines the language L.

Let the states be $\{1, 2, \ldots n\}$

⇒ Construct RE of the form $R_{ij}^{(k)}$ – set of string w that takes the FA from state 'i' to state 'j' without going through any states labeled greater than k.

⇒ Let us prove this theorem by induction for $R_{ij}^{(k)}$.

Basis: If $k=0$, there will be no intermediate node between i and j

(i) 

An arc from node 'i' to 'j'

(ii) 

An arc from the node 'i' itself.

The transition may be (i) $R_{ij}^{(0)} = \phi$ (ii) $R_{ij}^{(0)} = a$ (iii) $R_{ij}^{(0)} = a_1 + a_2 + \cdots + a_\ell$

if $i=j$ then $R_{ij}^{(0)} = \varepsilon + a$

if $i \neq j$ then $R_{ij}^{(0)} = a_1 + a_2 + \cdots + a_\ell$

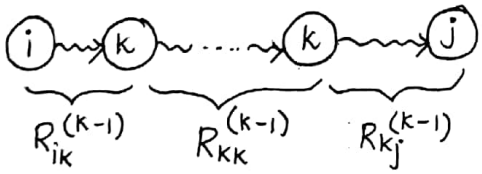
If there is no arc form i to j then, $R_{ij}^{(0)} = \phi$.

Induction: Let there is a path from state i to state j that goes through 'i' which is not greater than k.

Here it is assumed that we have already got $R_{ij}^{(k-1)}$

(i) Suppose if path does not goes through the state k, then $R_{ij}^{(k)} = R_{ij}^{(k-1)}$

(ii) If the path goes through the states k atleast one.



The set of lables for all path is represented by the RE

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

By combining the above two path going through k and not going through k, we have

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

∴ Regular expression $R_{ij}^{(k)}$ can be constructed for all values of i, j and k.

ARDEN'S THEOREM: (FA → RE)

Let P and Q be two regular expressions over Σ. If P does not contain ϵ, Then the equation in $R = Q + RP$ has a solution $R = QP^*$

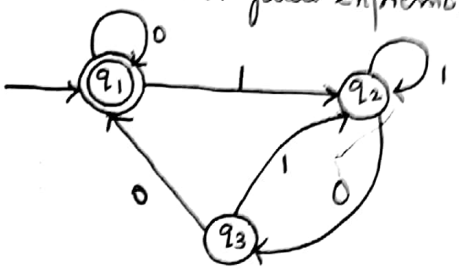
Using this theorem, it is easy to find the regular expression.

The condition to apply this theorem are,

(i) Finite automata does not have ϵ-moves.

(ii) It has only 1 start state.

Example: Construct the regular expression for the given finite automata



Solution: The transitions are defined by the equations,

$$q_1 = q_10 + q_30 + \epsilon \quad \text{——(1)}$$

$$q_2 = q_11 + q_21 + q_31 \quad \text{——(2)}$$

$$q_3 = q_20 \quad \text{——(3)}$$

Substitute (3) in (2)

$$q_2 = q_11 + q_21 + q_201$$

$$q_2 = q_11 + q_2(1+01)$$

$$q_2 = q_11 + \epsilon(1+01)^* \quad \text{——(4) [By theorem]}$$

Substitute (4) in (1)

$$q_1 = q_10 + q_20.0 + \epsilon$$

$$= q_10 + q_11(1+01)^*0.0 + \epsilon$$

$$= q_10 + q_1(1+(1+01)^*00) + \epsilon$$

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

$$= \epsilon(0+1(1+01)^*00)^* \quad \text{[By theorem]}$$

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

Since q_1 is the final state, the required regular expression is

$$= (0+1(1+01)^*00)^*$$

THAMSON'S CONSTRUCTION (RE → FA)

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

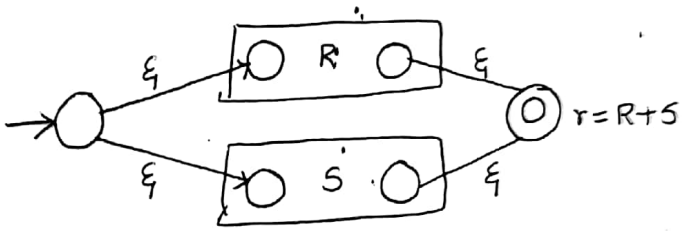
Proof: To prove L = L(E) for some ε-NFA.

Basis: (Zero operator)
(i) Exactly one accepting state → (q_0) $r = \varepsilon$
(ii) No arcs into the initial state → q_0 (q_1) $r = \phi$
(iii) No arcs out of the accepting state → q_0 —a→ (q_1) $r = a$

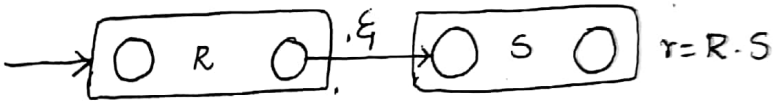
Induction: This theorem can be true for n number of operators.
In any type of regular expression there are only three cases possible.
1. Union 2. Concatenation 3. Closure

Case(i) Union. $r = R + S$

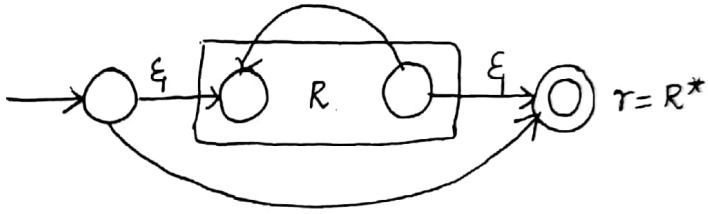
Let $r = R+S$, where r_1 and r_2 be the regular expressions.



Case(ii) $r = R \cdot S$ Concanetation

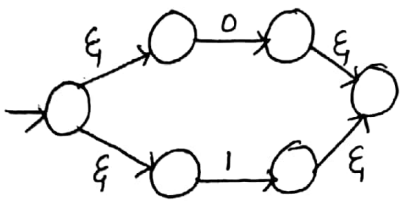


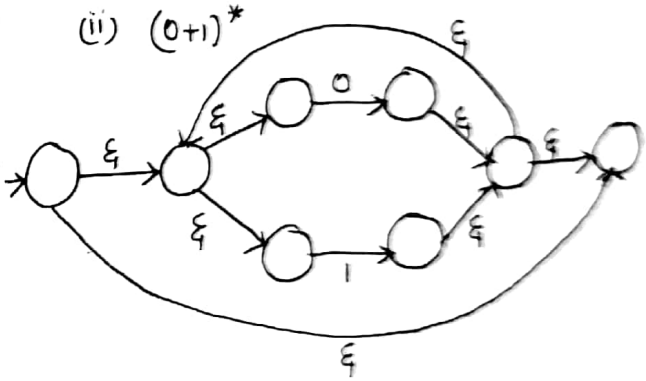
Case(iii) Closure



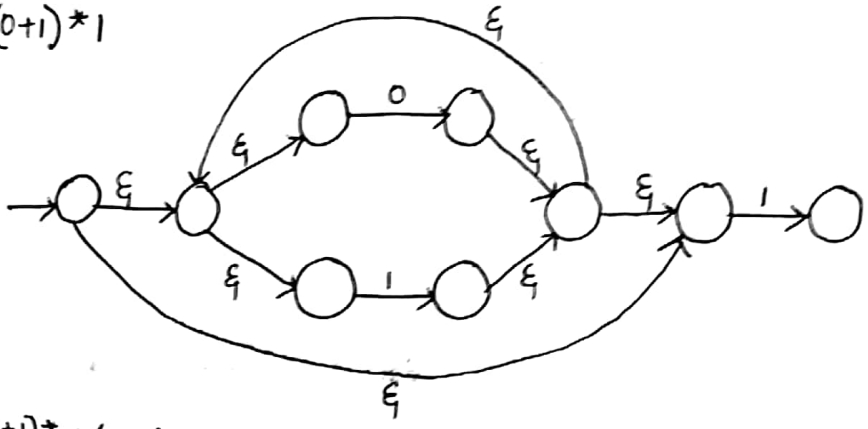
Problem 1: Convert the regular expression $(0+1)^*1(0+1)$ to an ε-NFA.

Solution: (i) Find the RE for 0+1 (ii) $(0+1)^*$

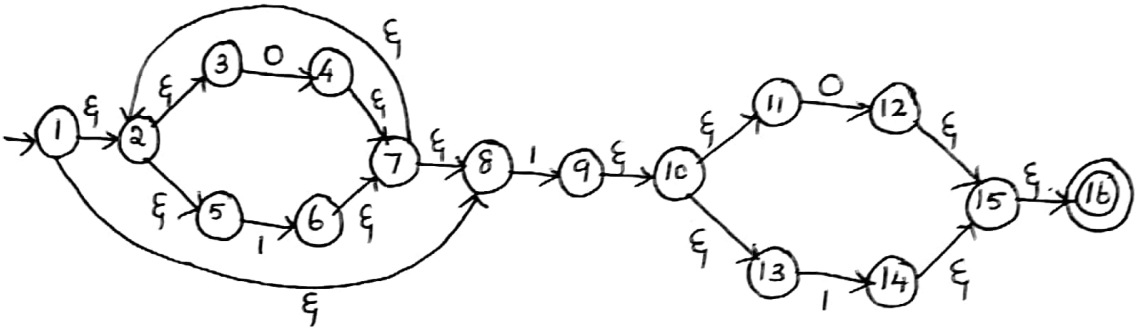




(iii) $(0+1)^*1$



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



Problem 2: Construct a NFA equivalent $(0+1)^*(00+11)$

Problem 3: Construct an NFA accepting L given by

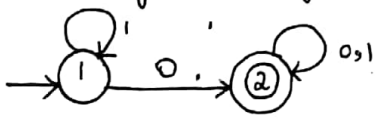
$L = \{ X \mid (a,b)^*, |X| \geq 3 \text{ and the third symbol of } x \text{ from right is 'b'} \}$

$[RE = (a+b)^* b (a+b)(a+b)]$

Problem 4: Construct an NFA to accept the language indicated by the following regular expression $((01+001)^*0^*)^*$

Problems: FA → RE

1) Convert the following to a regular expression.



Solution: Find $R_{ij}^{(k)}$ for all the values of i, j, k.

Let k=0, $R_{ij}^{(0)} = \begin{cases} \varepsilon + a_1 + a_2 + \dots + a_k & \text{if } i=j \\ a_1 + a_2 + \dots + a_r & \text{if } i \neq j \end{cases}$

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

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

$R_{ij}^{(1)} = R_{ij}^{(0)} + R_{i1}^{(0)} \left(R_{11}^{(0)}\right)^* R_{1j}^{(0)}$ ——— (2)

Find $R_{11}^{(1)}$, $R_{12}^{(1)}$, $R_{21}^{(1)}$, $R_{22}^{(1)}$

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

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

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

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

$\boxed{R_{11}^{(1)} = 1^*}$

(2) $\Rightarrow R_{21}^{(1)} = R_{21}^{(0)} + R_{21}^{(0)} \left(R_{11}^{(0)}\right)^* R_{11}^{(0)}$

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

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

$\boxed{R_{21}^{(1)} = \phi}$

(2) $\Rightarrow R_{12}^{(1)} = R_{12}^{(0)} + R_{11}^{(0)} \left(R_{11}^{(0)}\right)^* R_{12}^{(0)}$

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

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

$= 0 + 1^* 0$

$\boxed{R_{12}^{(1)} = 1^* 0}$

(2) $\Rightarrow R_{22}^{(1)} = R_{22}^{(0)} + R_{21}^{(0)} \left(R_{11}^{(0)}\right)^* R_{12}^{(0)}$

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

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

$\boxed{R_{22}^{(1)} = \varepsilon + 0 + 1}$

Let k=2

$R_{ij}^{(2)} = R_{ij}^{(1)} + R_{i2}^{(1)} \left(R_{22}^{(1)}\right)^* R_{2j}^{(1)}$ ——— (3)

Find $R_{11}^{(2)}$, $R_{12}^{(2)}$, $R_{21}^{(2)}$, $R_{22}^{(2)}$

③ $\Rightarrow R_{11}^{(2)} = R_{11}^{(1)} + R_{12}^{(1)} \left(R_{22}^{(1)}\right)^* R_{21}^{(1)}$

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

$= 1^* + \phi$

$\boxed{R_{11}^{(2)} = 1^*}$

③ $\Rightarrow R_{21}^{(2)} = R_{21}^{(1)} + R_{22}^{(1)} \left(R_{22}^{(1)}\right)^* R_{21}^{(1)}$

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

$= \phi + \phi$

$\boxed{R_{21}^{(2)} = \phi}$

③ $\Rightarrow R_{12}^{(2)} = R_{12}^{(1)} + R_{12}^{(1)} \left(R_{22}^{(1)}\right)^* R_{22}^{(1)}$

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

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

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

$\boxed{R_{12}^{(2)} = 1^* 0 \ (0+1)^*}$

③ $\Rightarrow R_{22}^{(2)} = R_{22}^{(1)} + R_{22}^{(1)} \left(R_{22}^{(1)}\right)^* R_{22}^{(1)}$

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

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

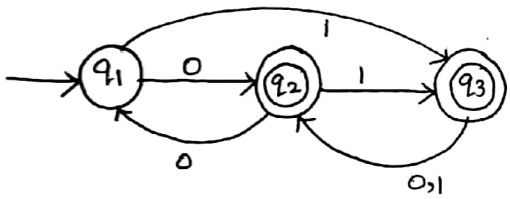
$= (\varepsilon+0+1) + (0+1)^*$

$\boxed{R_{22}^{(2)} = (0+1)^*}$

Regular expression for the accepting state

$\boxed{R_{12}^{(2)} = 1^* 0 \ (0+1)^*}$

Problem 2: Convert the following to a regular expression.



Solution: To find $R_{12}^{(3)} + R_{13}^{(3)}$

$$R_{ij}^{(0)} = \begin{cases} \varepsilon + a_1 + a_2 + \cdots + a_k & \text{if } i=j \\ a_1 + a_2 + \cdots + a_r & \text{if } i \neq j \end{cases}$$

k=0

$R_{11}^{(0)} = \varepsilon$ $\quad R_{21}^{(0)} = 0$ $\quad R_{31}^{(0)} = \phi$

$R_{12}^{(0)} = 0$ $\quad R_{22}^{(0)} = \varepsilon$ $\quad R_{32}^{(0)} = 0+1$

$R_{13}^{(0)} = 1$ $\quad R_{23}^{(0)} = 1$ $\quad R_{33}^{(0)} = \varepsilon$

k=1

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

$$R_{ij}^{(1)} = R_{ij}^{(0)} + R_{i1}^{(0)} \left(R_{11}^{(0)}\right)^* R_{1j}^{(0)}$$

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

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

$= \varepsilon + \varepsilon$

$\boxed{R_{11}^{(1)} = \varepsilon}$

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

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

$= 0 + 0$

$\boxed{R_{12}^{(1)} = 0}$

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

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

$\boxed{R_{13}^{(1)} = 1}$

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

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

$= 0 + 0 \cdot \varepsilon$

$= 0 + 0$

$\boxed{R_{21}^{(1)} = 0}$

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

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

$\boxed{R_{22}^{(1)} = \varepsilon + 00}$

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

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

$\boxed{R_{23}^{(1)} = 1 + 01}$

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

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

$\boxed{R_{31}^{(1)} = \phi}$

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

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

$= 0 + 1 + \phi$

$\boxed{R_{32}^{(1)} = 0 + 1}$

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

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

$\boxed{R_{33}^{(1)} = \varepsilon}$

$\underline{k=2} \qquad R_{ij}^{(2)} = R_{ij}^{(1)} + R_{i2}^{(1)}\left(R_{22}^{(1)}\right)^* R_{2j}^{(1)}$

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

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

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

$= \varepsilon + (00)^*$

$\boxed{R_{11}^{(2)} = (00)^*}$

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

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

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

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

$\boxed{R_{12}^{(2)} = 0\ (00)^*}$

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

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

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

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

$= 1 + 0^* 1$

$\boxed{R_{13}^{(2)} = 0^* 1}$

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

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

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

$= 0 + (00)^* 0$

$\boxed{R_{21}^{(2)} = (00)^* 0}$

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

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

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

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

$\boxed{R_{22}^{(2)} = (00)^*}$

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

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

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

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

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

$= (1 + 01) + 0^* 1$

$= 1 + 0^* 1$

$\boxed{R_{23}^{(2)} = 0^* 1}$

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

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

$\boxed{R_{31}^{(2)} = (0 + 1)(00)^* 0}$

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

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

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

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

$\boxed{R_{32}^{(2)} = (0 + 1)\ (00)^*}$

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

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

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

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

$\boxed{R_{33}^{(2)} = \varepsilon + (0 + 1)\ 0^* 1}$

Calculate $\gamma_{12}^{(3)} + \gamma_{13}^{(3)}$, to find regular expression

$\gamma_{12}^{(3)} = \gamma_{12}^{(2)} + \gamma_{13}^{(2)} \left(\gamma_{33}^{(2)}\right)^* \gamma_{32}^{(2)}$

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

$= 0\ (00)^* + 0^* 1\ \left((0 + 1)\ 0^* 1\right)^*\ (0 + 1)\ (00)^*$

$\gamma_{13}^{(3)} = \gamma_{13}^{(2)} + \gamma_{13}^{(2)} \left(\gamma_{33}^{(2)}\right)^* \gamma_{33}^{(2)}$

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

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

$\therefore \gamma_{12}^{(3)} + \gamma_{13}^{(3)} = 0^* 1\ \left((0 + 1) 0^* 1 + (\varepsilon + (0 + 1)\ 00)^*\right) + 0\ (00)^*$

Minimization of DFA.

The minimization of FSM means reducing the number of states from given FA. While minimizing FSM, first find out which two states are equivalent, we can represent those two states by one representative state.

Pbm: Give the minimized DFA for the following expression $(a/b)^* abb$

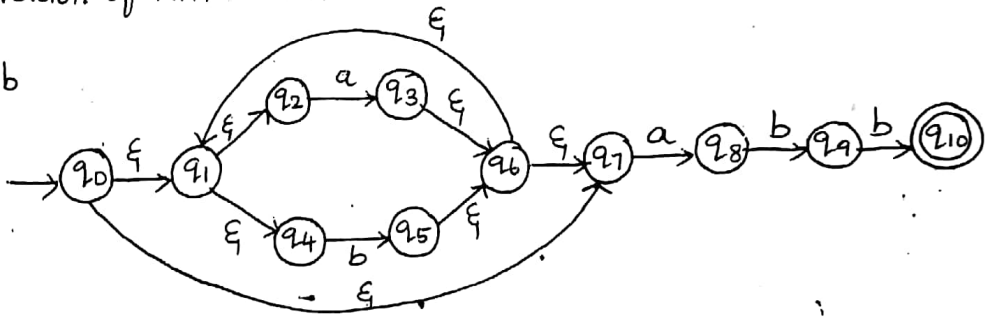
Solution: Steps:

(1) Design of NFA with ε for given regular expression

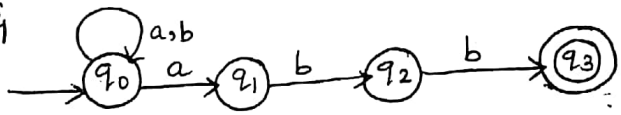
(2) Conversion of NFA with ε to without ε

(3) Conversion of NFA to DFA.

$(a/b)^* abb$



NFA without ε



Transition table:

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

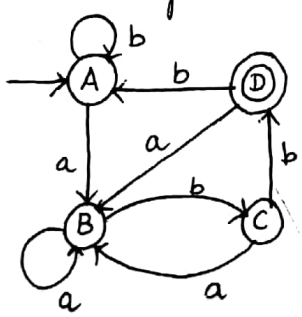
Equivalent DFA

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

Assume $q_0 = A$, $\{q_0, q_1\} = B$

$\{q_0, q_2\} = C$, $\{q_0, q_3\} = D$

Transition diagram,



Transition table,

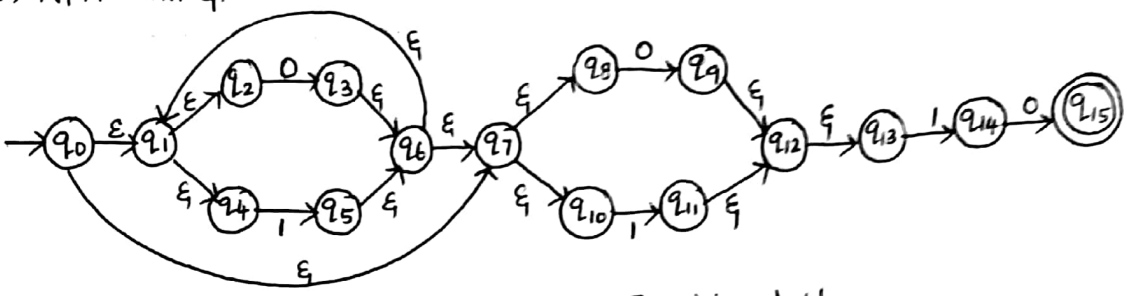
I/P / States	a	b
A	B	A
B	B	C
C	A	D
D		

Problem 2: Construct the minimized DFA for the regular expression

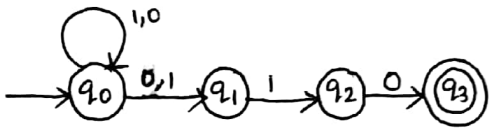
$(0+1)^*(0+1)10$

Solution:

(i) NFA with ε.



(ii) NFA without ε



Transition table

S \ I/P	0	1
q_0	$\{q_0, q_1\}$	$\{q_0, q_1\}$
q_1	ϕ	q_2
q_2	q_3	ϕ
q_3	ϕ	ϕ

Equivalent DFA

State \ I/P	0	1
→ q_0	$[q_0, q_1]$	$[q_0, q_1]$
q_1	ϕ	$[q_2]$
q_2	$[q_3]$	ϕ
q_3	ϕ	ϕ
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1, q_2]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_3]$	$[q_0, q_1, q_2]$
* $[q_0, q_1, q_3]$	$[q_0, q_1]$	$[q_0, q_1, q_2]$

Assume

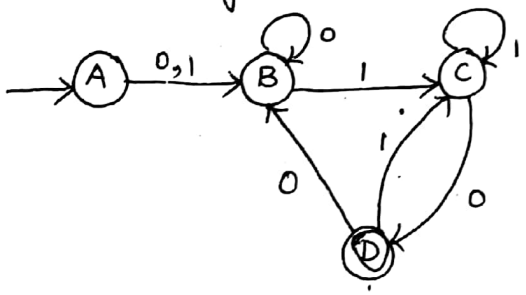
$q_0 = A$

$[q_0, q_1] = B$

$[q_0, q_1, q_2] = C$

$[q_0, q_1, q_3] = D$

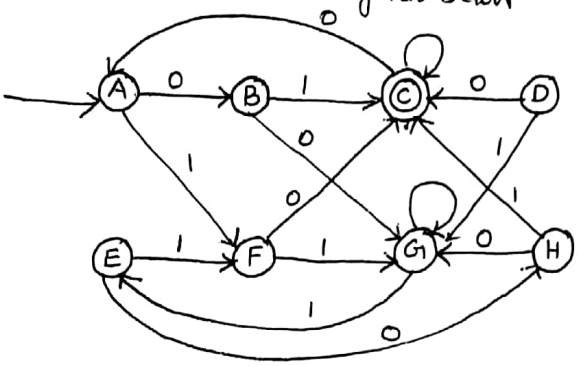
Transition Diagram (Min-DFA)



Transition table:

	0	1
→ A	B	B
B	B	C
C	D	C
D	B	C

Problem: 3 Minimize the DFA as given below



Solution:

Transition table

	0	1
→ A	B	F
B	G1	C
(C)	A	C
D	C	G1
E	H	F
F	C	G1
G1	G1	E
H	G1	C

Find the equivalent state

Consider pair (B, H)

$\delta(B,0) = G1$ | $\delta(B,1) = C$

$\delta(H,0) = G1$ | $\delta(H,1) = C$

∴ (B, H) are equivalent.

Since (B, H) are equivalent, consider the pair (A, E)

$\delta(A,0) = B$ (or) H | $\delta(A,1) = F$

$\delta(E,0) = H$ (or) B | $\delta(E,1) = F$

∴ (A, E) are equivalent.

Consider the pair (D, F)

$\delta(D,0) = C$ | $\delta(D,1) = G1$ ∴ (D, F) are equivalent.

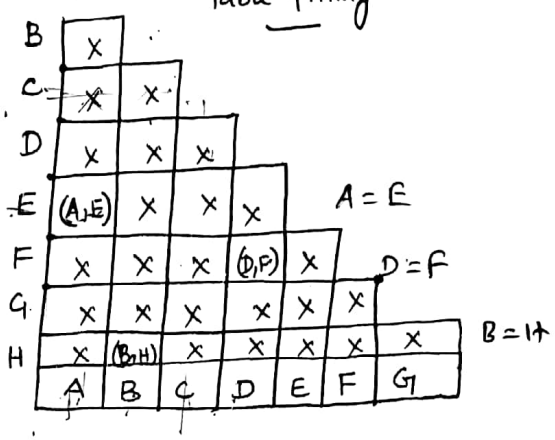
$\delta(F,0) = C$ | $\delta(F,1) = G1$

∴ The equivalent pairs are (A, E), (B, H) and (D, F)

∴ The minimized DFA will be

	0	1
→ A	B	D
B	G1	C
(C)	A	C
D	C	G1
G1	G1	A

Table filling



Problem 1: Define distinguishable and indistinguishable states. Using table filling method, minimize the following DFA. Draw the transition diagram for resulting DFA.

	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

Distinguishable states

↳ If for some input string w, $\hat{\delta}(p,w)$ gives an accepting state and $\hat{\delta}(q,w)$ gives a non-accepting state then states, p and q are called distinguishable or non-equivalent state.

Indistinguishable states

↳ If for some input string w, $\hat{\delta}(p,w)$ and $\hat{\delta}(q,w)$ both produces either accepting or non accepting states.

Solution:

Here Final states are C, F, I

Other states are A, B, D, E, G, H

Since C, F, I are final state

Consider the pair (E, H)

$\delta(E,0) = F$ | $\delta(E,1) = I$

$\delta(H,0) = I$ | $\delta(H,1) = C$

∴ (E, H) is equivalent state

Consider the pair (B, E)

$\delta(B,0) = C$ | $\delta(B,1) = F$

$\delta(E,0) = F$ | $\delta(E,1) = I$

∴ (B, E) is equivalent state.

Consider the pair (B, H)

$\delta(B,0) = C$ | $\delta(B,1) = F$

$\delta(H,0) = I$ | $\delta(H,1) = C$

∴ (B, H) is equivalent state.

Since (B, H) and (B, E) is equivalent,

Consider the pair (A, G)

$\delta(A,0) = B$ or H | $\delta(A,1) = E$ or B

$\delta(G,0) = H$ or B | $\delta(G,1) = B$ or E

∴ (A, G) is equivalent state.

Since (B,E) and (E,H) are equivalent

Consider the pair (A,D)

$\delta(A,0) = B$ or E | $\delta(A,1) = E$ or H

$\delta(D,0) = E$ or B | $\delta(D,1) = H$ or E

$\therefore$ (A,D) is equivalent state

Since (E,H) and (B,H) are equivalent

Consider the pair (D,G)

$\delta(D,0) = E$ or H | $\delta(D,1) = H$ or B

$\delta(G,0) = H$ or E | $\delta(G,1) = B$ or H

$\therefore$ (D,G) are equivalent.

Since (D,G) and (B,H) are equivalent

Consider the pair (C,F)

$\delta(C,0) = D$ or G | $\delta(C,1) = H$ or B

$\delta(F,0) = G$ or D | $\delta(F,1) = B$ or H

$\therefore$ (C,F) are equivalent.

Table filling:

	A	B	C	D	E	F	G	H
B	X							
C	X	X						
D	(A,D)	X	X					
E	X	(B,E)	X	X				
F	X	X	(C,F)	X	X			
G	X	X	X	(D,G)	X	X		
H	X	(B,H)	X	X	(E,H)	X	X	
I	X	X	(C,I)	X	X	(F,I)	X	X

Since (A,G) and (B,E) are equivalent

Consider the pair (F,I)

$\delta(F,0) = G$ or A | $\delta(F,1) = B$ or E

$\delta(I,0) = A$ or G | $\delta(I,1) = E$ or B

$\therefore$ (F,I) is equivalent state.

Since (A,D) and (E,H) are equivalent

Consider the pair (C,I)

$\delta(C,0) = D$ or A | $\delta(C,1) = H$ or E

$\delta(I,0) = A$ or D | $\delta(I,1) = E$ or H

$\therefore$ (C,I) is equivalent.

Here

State A = G = D

State B = H = E

State C = I = F

Minimized DFA is

I/P / State	0	1
A	B	B
B	C	C
©	A	B

Pumping lemma for Regular sets

⇒ Used for checking whether given string is accepted by regular expression or not.

⇒ Whether given language is regular or not.

Theorem: Let L be a regular set. Then there is a constant n such that if z is any word in L and $|z| \geq n$ we can write $z = uvw$ such that $|uv| \leq n$, $|v| \geq 1$ for all $i \geq 0$, uv^iw is in L. The n should not be greater than the number of states.

Proof: If the language L is regular it is accepted by a DFA.

$M = (Q, \Sigma, \delta, q_0, F)$ with some particular number of states say n.

Consider the input can be $a_1, a_2, \ldots a_m$, $m \geq n$. The mapping function δ could be written as $\delta(q_0, q_1, \ldots q_i) = q_i$.

If q_m is in F (ie) $q_1, q_2, \ldots q_m$ is in $L(M)$ then $a_1, a_2 \ldots a_j\ a_{k+1}\ a_{k+2} \ldots a_m$ is also in $L(M)$.

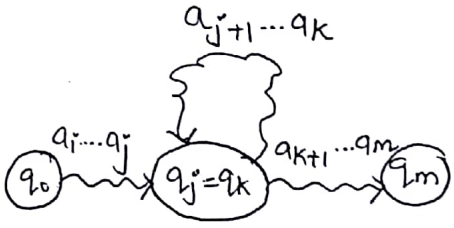


Fig: Pumping lemma

Since there is path from q_0 to q_m that goes through q_j but not around the loop labelled $a_{j+1} \ldots a_k$. Thus.

$$\begin{aligned}\delta(q_0, a_1, a_j\ a_{k+1} \ldots a_m) &= \delta(\delta(q_0, q_1, \ldots q_j), a_{k+1}, \ldots a_m) \\ &= \delta(q_j, q_{k+1} \ldots q_m) \\ &= \delta(q_k, q_{k+1} \ldots q_m) \\ &= q_m\end{aligned}$$

Thus $a_1 \ldots a_j (a_{j+1} \ldots a_k)^i a_{k+1} \ldots a_m$ is in $L(M)$ for any $i \geq 0$

Problem 1: Show that $L = \{0^{n^2} \mid n \geq 1\}$ is not regular.

Solution

(i) Assume L is regular.

(ii) No. of states $= n^2 > n$

(iii) Let $z = 0^{n^2}$, consider $n^2 = r$, $z = 0^r$

(iv) According to the pumping lemma,

z can be written as,

$z = uvw$ such that $|uv| \leq n$, $|v| \geq 1$

$\therefore z = 0^r$ can be expressed as,

Assume, $uv = 0^m$, where $m < r$

$v = 0^q$, where $q < m$

and $w = 0^{r-m}$

(v) check the condition.

(a) $|uv| \leq n$, $|0^m| \leq n$, $m \leq n$

(b) $|v| \geq 1$, $|0^q| \geq 1$, $q \geq 1$

(c) for all $i \geq 0$, the string uv^iw is in L

$$uv^iw = uv^{i-1}(v)w$$
$$= (uv)(v)^{i-1}w$$
$$= 0^m (0^q)^{i-1} 0^{r-m}$$
$$= 0^{m+q(i-1)+r-m}$$
$$\boxed{uv^iw = 0^{q(i-1)+r}}$$

For $i = 0$, $uv^0w = uw = 0^{r-q} \neq 0^r \notin L$

$i = 1$, $uvw = 0^r \in L$

$i = 2$ $uv^2z = 0^{q+r} \notin L$

Since for $i = 0, 2$, we have the string that does not belongs to the language L.

$\therefore L = \{0^{n^2} \mid n \geq 1\}$ is not regular.

Problem 2: Show that $L = \{a^n b^n \mid n \geq 1\}$ is not regular.

Solution:

i) Assume L is regular.

ii) No. of. states $L = \{a^n b^n \mid n \geq 1\}$ (ie) $n+n = 2n > n$

iii) Let $z = a^n b^n$

According to the pumping lemma, z can be written as,

$z = uvw$ such that $|uv| \leq n$, $|v| \geq 1$

iv) Consider $z = a^n b^n = a^m a^{n-m} b^n$

$z = a^m a^{n-m} b^n$ can be expressed as,

$$\boxed{uv = a^m,\ v = a^j \quad w = a^{n-m} b^n}$$

v) Check the conditions.

(a) $|uv| \leq n \Rightarrow |a^m| \leq n,\ m \leq n$

(b) $|v| \geq 1 \Rightarrow |a^j| \geq 1,\ j \geq 1$

(c) for all $i \geq 0$ the string uv^iw is in L.

$$uv^iw = u\,v^{i-1}(v)\,w$$
$$= (uv)\,(v)^{i-1} w$$
$$= a^m\,(a^j)^{i-1}\,a^{n-m} b^n$$
$$\boxed{uv^iw = a^{n+j(i-1)} b^n}$$

For $i=0$, $uw = a^{n-j} b^i \notin L$

$i=1$, $uvw = a^n b^n \in L$

$i=2$, $uv^2w = a^{n+j} b^n \notin L$

Since for $i = 0, 2$, we have the string that does not belongs to the language L.

$\therefore L = \{a^n b^n \mid n \geq 1\}$ is not regular.

Problem 3: Show that $L = \{a^m b^n a^{m+n} \mid m \geq 1, n \geq 1\}$ is not regular.

Solution:

i) Assume L is regular.

ii) Let $z = a^m b^n a^{m+n}$

iii) According to pumping lemma

$z = uvw, \quad |uv| \leq n, \quad |v| \geq 1$

$z = a^m b^n a^{m+n}$ is expressed as,

$$\boxed{uv = a^m, \quad v = a^q, \quad w = a^{P-m} b^n a^o} \quad [o = m+n]$$

iv) Check the condition.

(a) $|uv| \leq n, \quad |a^m| \leq n, \quad m \leq n$

(b) $|v| \geq 1, \quad |a^q| \geq 1, \quad q \geq 1$

(c) for all $i \geq 0$ the string $uv^i w$ is in L

$$uv^i w = uv^{i-1} vw$$
$$= (uv)(v)^{i-1} w$$
$$= a^m (a^q)^{i-1} a^{P-m} b^n a^o$$
$$\boxed{uv^i w = a^{q(i-1)+P} b^n a^o}$$

For $i=0$, $uw = a^{P-q} b^n a^o \notin L$

$i=1$, $uvw = a^P b^n a^o \in L$

$i=2$, $uv^2 w = a^{P+q} b^n a^o$

Take some random value for P, q, n and o

$P=4, \quad n=3, \quad o=10, \quad q=1$

$a^{4+1} b^3 a^{10} = a^5 b^3 a^{10} \notin L.$ [Here $m=5, n=3$, $m+n = 8 \neq 10$]

∴ Since for $i=0$ and $i=2$, string does not belongs to L.

∴ $L = \{a^m b^n a^{m+n} \mid m \geq 1, n \geq 1\}$ is not regular.

Problem 4: Is the following language regular? Justify your answer.

$$L = \{0^{2n} \mid n \geq 1\}$$

Solution:

This is a language length of string is always even.

(ie) $n = 1$, $L = 00$

$n = 2$, $L = 0000$ and so on.

Let $L = uvw$

$L = 0^{2n}$

$|z| = 2n = uv^i w$

If we add $2n$ to this string length

$|x| = 4n = uv.vw$ = even length of string.

Thus even after pumping $2n$ to the string we get the even length. So the language $L = \{0^{2n} \mid n \geq 1\}$ is regular language.

Problem 5: prove $L = \{a^p \mid p$ is a prime$\}$ is not regular.

Solution: Let, assume L is regular and p is a prime number.

$$L = a^p$$

$|x| = uvw$, $i = 1$

Consider $L = uv^i w$ where $i = 2$

$= uv.vw$

Adding 1 to p, we get

$$p < |uvvw|$$

$$p < p+1$$

But $p+1$ is not prime number.

Assumption is contradictory. Thus $L = \{a^p \mid p$ is a prime$\}$ behaves as it is not a regular language.

UNIT-II GRAMMAR

Grammar Introduction:

A grammar of a language (G_1) is defined as,

$$G_1 = (V, T, P, S)$$

V - finite set of objects called Variables (Non-terminal)

T - finite set of objects called Terminals

$S \in V$ - start symbol

P - finite set of productions.

Types of Grammar:

- Type 0 grammars / unrestricted grammars (Recursively Enumerable Language)
- Type 1 grammars / context sensitive grammars
- Type 2 grammars / context free grammars
- Type 3 grammars / Regular grammar.

1) Type 0 grammars

- No restrictions on the production rules
- Production rule is of the form

$$\alpha \rightarrow \beta \mid \alpha \neq \beta$$

Where $\alpha, \beta \rightarrow$ can be strings composed by terminals and non-terminals.

- This grammar can be modeled using Turing Machine.

2) Type 1 grammars

- Context sensitive grammar (CSG) :
- Production rule is of the form

$$\alpha A \beta \Rightarrow \alpha \gamma \beta$$

Here $A \rightarrow$ Non-terminal symbol

$\alpha, \beta, \gamma \rightarrow$ Combination of terminals and non-terminals.

- This grammar can be modeled using linear bounded automata.

3) Type 2 grammar:

- Context Free Grammar (CFG).
- production rule is of the form $A \rightarrow \alpha$

  A - Non terminal Symbol

  α - Terminal or non-terminal symbol.

- This grammar can be modeled using push down automata.

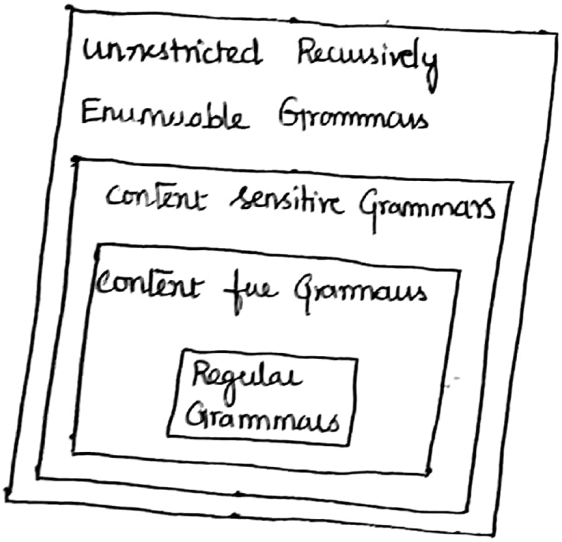
4) Type 3 grammar:

- Regular grammar that describe regular / formal languages.
- production rule consist of,
  - only one Non terminal at the left hand side.
  - Right hand side having a single terminal and may or may not be followed by non terminals.

    $A \rightarrow a$, $A \rightarrow aB$

- This grammar can be modeled using finite automata.

Chomsky hierarchy.



$\Rightarrow$ Regular grammars $\subseteq$ Context free grammars $\subseteq$ Context sensitive grammars $\subseteq$ Unrestricted grammars

Context free Language and Grammars.

Definition: The context free grammar can be formally defined as a set denoted by $G = (V, T, P, S)$ where V and T are set of non-terminals and terminals respectively

P is set of production rules. $NT \rightarrow NT$

$NT \rightarrow T$

S is a start symbol.

Example:

$P = \{ S \to S+S$
$S \to S*S$
$S \to (S)$
$S \to A \}$

Syntax of any English statement is,

SENTENCE → NOUN VERB
NOUN → Rama / Seeta / Gopal
VERB → goes / Writes / sings.

Dervied strings: "Rama sings"

Problems:

1) Contruct the CFG for the regular expression $(0+1)^*$.

solution: $CFG = (V, T, P, S)$

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

where $P = \{ S \to 0S \mid 1S$
$S \to \varepsilon \}$

Example:

$(0+1)^* = \{\varepsilon, 0, 1, 01, 10, 00, 11, \ldots\}$

2) Construct CFG for the language L which has all the strings which are all palindrome over $\Sigma = \{a, b\}$

Solution: $G_1 = (\{S\}, \{a,b\}, P, S)$

$S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$

Example: abaaba

$S \to a\underline{S}a$
$\to ab\underline{S}ba$
$\to abaSaba$
$\to aba\varepsilon aba$
$\to abaaba$

3) Construct CFG for $\{0^m 1^n \mid 1 \le m \le n\}$

Solution: $G_1 = (V, T, P, S)$

$V = \{S, A\}, \quad T = \{0,1\}$

$P = \{ S \to 0S1 \mid 0A \mid 01$
$A \to 1A \mid 1 \}$

Example: 00111

$S \to 0\underline{S}1$
$\to 00\underline{S}1$
$\to 001\underline{A}1$
$\to 00111$

4) Construct CFG for $L = \{a^m b^n c^p \mid m+n = p \text{ and } p \ge 1\}$

Solution: $G_1 = (V, T, P, S)$

$V = \{S, A\} \quad T = \{a, b, c\}$

$P = \{ S \to aSc \mid bAc \mid ac \mid bc$
$A \to bc \mid bc$

5) Consider the alphabet $\Sigma = \{a, b, (,), +, *, \cdot, /, \ldots \varepsilon\}$. Construct a CFG that generates all strings in Σ^* that are RE over $\{a, b\}$.

Solution:

$E \to E+E$

$E \to E*E$

$E \to E \cdot E$

$E \to E/E$

$E \to a \mid b \mid \varepsilon$.

Problems: Grammar to Language

1) If $S \to aSb \mid aAb$, $A \to bAa$, $A \to ba$ is a CFG then determine CFL.

Solution:

$S \to a\underline{S}b$

$\to aa\underline{S}bb$

$\to aaa\underline{A}bbb$

$\to \underbrace{aaa}_{a^n}\underbrace{ba}_{\cdot}\underbrace{bbb}_{b^n}$

$L(G) = \{a^n b^m a^m b^n \mid m, n \geq 1\}$

2) If $S \to aSa \mid bSb \mid \varepsilon$ is CFG. Find $L(G)$.

Solution:

$S \to a\underline{S}a$

$\to aa\underline{S}aa$

$\to aab\underline{S}baa$

$\to \underbrace{aab}_{W}\underbrace{baa}_{W^R}$

$\therefore L(G) = \{WW^R \mid W \in (a, b)^*\}$

3) Find the context free language for the following grammars.

(i) $S \to aSbS \mid bSaS \mid \varepsilon$

(ii) $S \to aSb \mid ab$.

Solution

(i) $S \to a\underline{S}bS$

$\to ab\underline{S}aSbS$

$\to aba\underline{S}bS$

$\to ababS$

$\to abab$

$S \to b\underline{S}aS$

$\to ba\underline{S}$

$\to ba$

$\therefore L$ containing equal number of a's and b's

(ii) $S \to aSb \mid ab$

$S \to a\underline{S}b$

$\to aa\underline{S}bb$

$\to aaabbb$

L containing equal number of a's followed by equal number of b's

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

Derivations: Use the productions from head to body (ie) from start symbol expanding till reaches the given string.

Two types of derivations are,

(i) Left Most Derivation (LMD)

(ii) Right Most Derivation (RMD)

⇒ LMD is a derivation in which the leftmost non-terminal is replaced first from the sentential form.

(ie) $S \underset{lm}{\overset{*}{\Rightarrow}} \alpha$, then α is left sentential form.

⇒ RMD is a derivation in which rightmost non-terminal is replaced first from the sentential form.

(ie) $S \underset{rm}{\overset{*}{\Rightarrow}} \alpha$, then α is right sentential form.

Derivation tree (parse tree)

- It is a graphical representation for the derivation of the given production rules for a given CFG.

Properties

i) Root node is always a node indicating start symbol.

ii) Derivation is read from left to right

iii) Leaf nodes are always terminal nodes.

iv) Interior nodes are always non-terminal nodes.

Example:

Consider the grammar G has the production

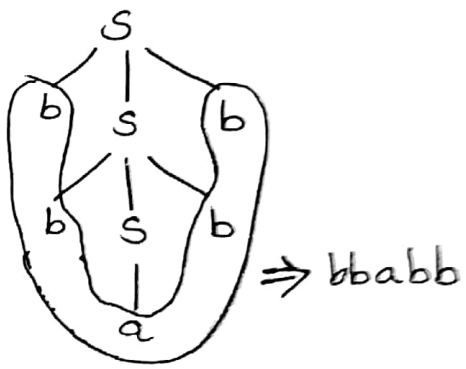
$S \rightarrow bSb \mid a \mid b$ and string "bbabb"

Derivation:

$S \rightarrow bSb$

$\rightarrow bbSbb$

$\rightarrow bbabb$



⇒ bbabb

Problem 1: Construct the derivation tree for the string "aaabbabbba" using LMD and RMD. using $S \to aB \mid bA$, $A \to a \mid aS \mid bAA$, $B \to b \mid bS \mid aBB$

Solution:

LMD:

$S \Rightarrow aB$

$\Rightarrow aaBB \quad (B \to aBB)$

$\Rightarrow aaaBBB \quad (B \to aBB)$

$\Rightarrow aaabBB \quad (B \to b)$

$\Rightarrow aaabbB \quad (B \to b)$

$\Rightarrow aaabbaBB \quad (B \to aBB)$

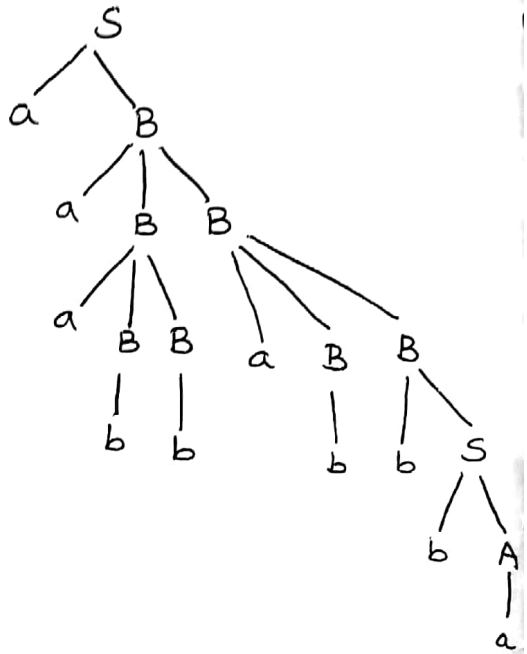
$\Rightarrow aaabbabB \quad (B \to b)$

$\Rightarrow aaabbabbS \quad (B \to bS)$

$\Rightarrow aaabbabbbA \quad (S \to bA)$

$\Rightarrow aaabbabbbba \quad (A \to a)$

Parse tree:



RMD:

$S \Rightarrow aB$

$\Rightarrow aaBB \quad (B \to aBB)$

$\Rightarrow aaBbS \quad (B \to bS)$

$\Rightarrow aaBbbA \quad (S \to bA)$

$\Rightarrow aaBbbba \quad (A \to a)$

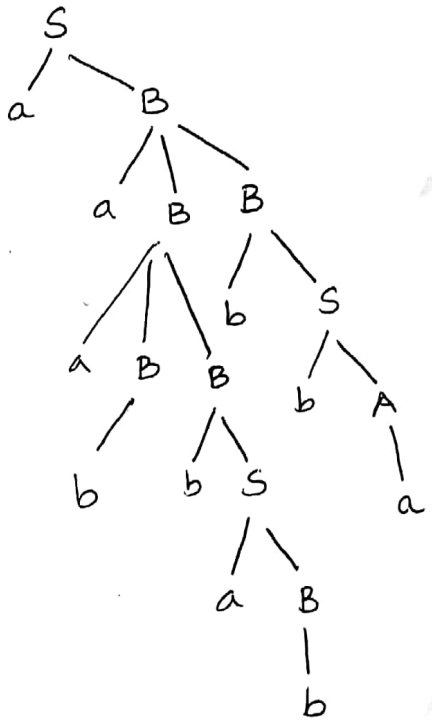
$\Rightarrow aaaBBbbba \quad (B \to aBB)$

$\Rightarrow aaaBbsbba \quad (B \to bS)$

$\Rightarrow aaaBbaBbba \quad (S \to aB)$

$\Rightarrow aaaBbabbba \quad (B \to b)$

$\Rightarrow aaabbabbba \quad (B \to b)$



Prob 2: Write a grammar G_1 to recognize all prefix expressions involving all binary arithmetic operators construct the parse tree for the sentence "-*+abc/de"

Solution:

$G_1 = (V, T, P, S)$ where

$V = \{S, A, B, C, D, E\}$

$T = \{+, -, *, /, a, b, c, d, e\}$

S is a start symbol

Productions are,

$S \rightarrow A|B|C|D|E$

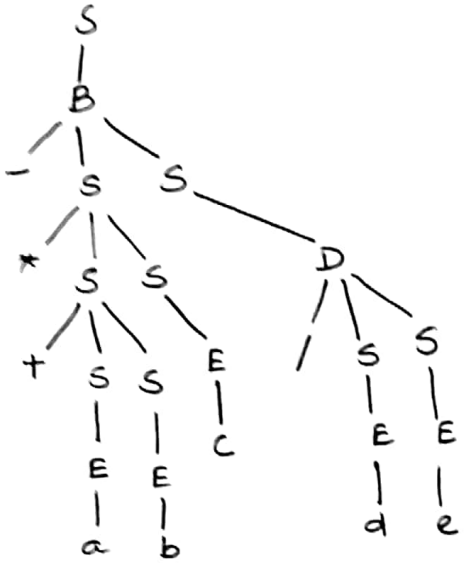
$A \rightarrow +SS$

$B \rightarrow -SS$

$C \rightarrow *SS$

$D \rightarrow /SS$

$E \rightarrow a|b|c|d|e$



Ambiguity:

If there exists more than one parse trees for a given grammar, that means there could be more than one leftmost or rightmost derivation possible and then that grammar is said to be ambiguous grammar.

Problems:

1) The CFG is given by $G_1 = (V, T, P, S)$ where $V = \{E\}$, $T = \{id\}$, $S = \{E\}$

$P = \{E \rightarrow E+E, E \rightarrow E*E, E \rightarrow id\}$. Is the grammar ambiguous?

Solution:

Consider the string id * id + id.

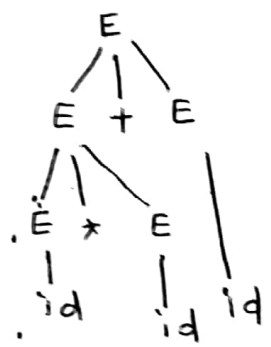
$E \Rightarrow E+E$

$\Rightarrow E*E+E$

$\Rightarrow id*E+E$

$\Rightarrow id*id+E$

$\Rightarrow id*id+id$



$E \Rightarrow E*E$

$\Rightarrow E*E+E$

$\Rightarrow id*E+E$

$\Rightarrow id*id+E$

$\Rightarrow id*id+id$

Here, we obtain two different parse tree for the string id * id + id.

∴ The given grammar is ambiguous.

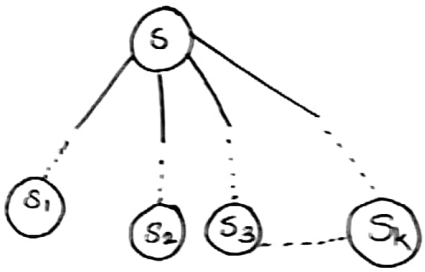
Relationship between derivation and derivation trees.

Theorem: Let $G = (V, T, P, S)$ be a context free grammar. Then $S \overset{*}{\Rightarrow} \alpha$ if and only if there is a derivation tree in grammar G which gives the string α.

Proof: For a non-terminal S there exists $S \overset{*}{\Rightarrow} w$ if and only if there is a derivation tree starting from root S and yielding w.

Basis of induction:

Assume that there is only one interior node S.
The derivation tree yielding $S_1, S_2, S_3 \ldots S_n$.
From S is that means $S \Rightarrow S_1, S_2 \ldots S_n \overset{*}{\Rightarrow} a$ is input string.



Induction hypothesis:

⇒ We assume that for k-1 nodes the derivation tree can be drawn. We then prove that for k vertices also we can have a derivation tree.

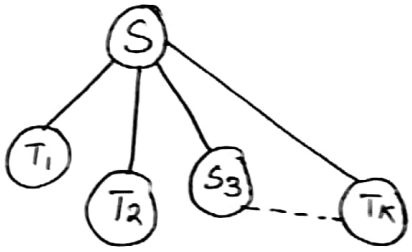
⇒ That means the input string can be derived as $S \rightarrow S_1 S_2 S_3 \ldots S_k$.

⇒ There are two cases,

(i) S_i may be a leaf variable

(ii) S_i may be an interior node yielding a.

⇒ The S derives α by fewer number of k steps then $a \in S_1 S_2 S_3 S_4 \ldots S_k$.



If $a_i = S_i$ then S_i is leaf node (terminal) and if $S_i \overset{*}{\Rightarrow} a_i$ then S_i is an interior node.

This proves that $S \overset{*}{\Rightarrow} S_1, S_2, S_3, \ldots S_n \overset{*}{\Rightarrow} a$ can be obtained.

Simplification of CFG

Simplification of grammar means reduction of grammar by removing useless symbols.

- Elimination of useless symbols
- Elimination of unit productions.
- Elimination of Null production (ε)

Elimination of useless symbols

Any symbol is useful when it appears on the right hand side in the production rule and generates some terminal string. If no such derivation exists then it is supposed to be an useless symbol.

Example: Eliminate the useless symbol from the following grammar.

$S \rightarrow aS \mid A \mid C$

$A \rightarrow a$

$B \rightarrow aa$

$C \rightarrow aCb$

Solution:

Production with terminal symbols are

$A \rightarrow a$

$B \rightarrow aa$

Start symbol $S \rightarrow aS \mid A \mid C$

Here, there is no production for B

$\therefore$ B is useless symbol.

IIIly $S \rightarrow C \rightarrow aCb \rightarrow aaCbb \rightarrow aaaCbbb \rightarrow \ldots$

Here, no terminating symbol for C

$\therefore$ C is useless symbol.

$\therefore$ Eliminate B and C, we get

$S \rightarrow aS \mid A$

$A \rightarrow a$

Elimination of ε production:

If there is ε production, remove it, without changing the meaning of the grammar.

Example: Eliminate ε-productions from the CFG

$A \to 0B1 \mid 1B1$

$B \to 0B \mid 1B \mid \varepsilon$

Solution:

$A \to 0B1 \mid 1B1$

$B \to \varepsilon$, $A \to 01 \mid 11$

$B \to 0B \mid 1B$

$B \to \varepsilon$, $B \to 0 \mid 1$

∴ After Elimination,

$A \to 0B1 \mid 1B1 \mid 01 \mid 11$

$B \to 0B \mid 1B \mid 0 \mid 1$

Removing Unit productions

The unit productions are the productions in which one non-terminal gives another non-terminal.

$X \to Y$

Example: Eliminate the unit production from following grammar.

$S \to AB$

$A \to a$

$B \to C \mid b$

$C \to D$

$D \to E \mid bC$

$E \to d \mid Ab$

Here, $B \to C$, $C \to D$, $D \to E$ are unit productions.

∴ $D \to E \mid bc$ can be written as $D \to d \mid Ab \mid bC$

Similarly $C \to E \mid bc$, B becomes $B \to d \mid Ab \mid bc \mid b$.

After removing unit productions.

$S \to AB$

$A \to a$

$B \to d \mid Ab \mid bC \mid b$

$C \to d \mid Ab \mid bC$

$D \to d \mid Ab \mid bC$ ✗

$E \to d \mid Ab$ ✗

Chomsky normal form (CNF)

A context free grammar $G_1 = (V, T, P, S)$ is said to be in CNF if each production in G_1 is of the form

[NT → NT. NT
NT → T]

$X \rightarrow YZ$

$X \rightarrow \alpha$, where $X, Y, Z \in V$, and $\alpha \in T$

Problems:

1) Convert the given CFG1 to CNF $S \rightarrow aSa \mid bSb \mid a \mid b$

Solution:

Productions are

$S \rightarrow aSa$
$S \rightarrow bSb$
$S \rightarrow a$
$S \rightarrow b$

Here, the productions which are already in CNF is

$S \rightarrow a$
$S \rightarrow b$

Apply CNF rule to other productions,

$S \rightarrow aSa \quad \mid C_a \rightarrow a$

$S \rightarrow C_a S C_a$

$S \rightarrow C_a A$
$A \rightarrow S C_a$

$S \rightarrow bSb \quad \mid C_b \rightarrow b$

$S \rightarrow C_b S C_b$

$S \rightarrow C_b B$
$B \rightarrow S C_b$

The resultant productions are,

$S \rightarrow C_a A \mid C_b B \mid a \mid b$
$A \rightarrow S C_a$
$B \rightarrow S C_b$
$C_a \rightarrow a$
$C_b \rightarrow b$

2) Reduce the following grammar to Chomsky normal form.

$S \to a \mid AAB$

$A \to ab \mid aB \mid \epsilon$

$B \to aba \mid \epsilon$

Solution:

Productions are,

$S \to a$

$S \to AAB$

$A \to ab$

$A \to aB$

$A \to \epsilon$

$B \to aba$

$B \to \epsilon$

Here, Eliminate ϵ production.

$A \to \epsilon$ and $B \to \epsilon$

$\Rightarrow$ After elimination, productions are

$S \to a \mid AAB \mid AA \mid AB \mid B$

$A \to ab \mid aB \mid a$

$B \to aba$

$\Rightarrow$ Eliminate unit production $S \to B$

$S \to a \mid AAB \mid AA \mid AB \mid aba$

$A \to ab \mid aB \mid a$

$B \to aba$

Productions already in CNF is

$S \to a$

$S \to AA$

$S \to AB$

$A \to a$

Apply CNF rules to other productions,

$S \to \underbrace{AA}_{S}B$

$\boxed{S \to SB}$

$S \to aba$

$S \to \underbrace{C_a C_b}_{C_1} C_a$

$C_a \to a$

$C_b \to b$

$\boxed{\begin{array}{l} S \to C_a C_1 \\ C_1 \to C_b C_a \end{array}}$

$\boxed{\begin{array}{l} A \to C_a C_b \\ A \to C_a B \end{array}}$

$B \to \underbrace{C_a C_b}_{C_1} C_a$

$\boxed{B \to C_a C_1}$

$\therefore$ The resultant productions are,

$S \to SB \mid C_a C_1 \mid AA \mid AB \mid a$

$C_1 \to C_b C_a$

$A \to C_a C_b \mid a \mid C_a B \mid C_a \epsilon_1$

$B \to C_a C_1$

3) Convert the given CFG to CNF.

$S \to aB$ | $A \to bAA$
$S \to bA$ | $B \to b$
$A \to a$ | $B \to bS$
$A \to aS$ | $B \to aBB$

Solution:

Productions already in CNF is,

$A \to a$
$B \to b$

Apply CNF rules to other productions.

$C_a \to a$, $C_b \to b$

$S \to C_aB$

$S \to C_bA$

$A \to C_aS$

$A \to C_bAA$ ($AA = C_1$)

$A \to C_bC_1$

$C_1 \to AA$

$B \to C_bS$

$B \to C_aBB$ ($BB = C_2$)

$B \to C_aC_2$

$C_2 \to BB$

The resultant productions are,

$S \to C_aB$
$S \to C_bA$
$A \to C_aS$
$A \to a$
$A \to C_bC_1$
$C_1 \to AA$
$B \to b$
$B \to C_bS$
$B \to C_aC_2$
$C_2 \to BB$

4) Convert the grammar with productions into CNF

$A \to bAB \mid \lambda$, $B \to BAa \mid \lambda$.

5) Convert the grammar $S \to AB \mid aB$, $A \to aab \mid \epsilon$, $B \to bbA$ into CNF

6) Convert to chomsky Normal Form.

$S \to A \mid CB$ | $B \to 1B \mid 1$ | $D \to 2D \mid 2$
$A \to C \mid D$ | $C \to 0C \mid 0$

Greibach Normal Form (GNF)

A grammar $G=(V,T,P,S)$ is said to be in GNF if every production rule is of the form

$$X \rightarrow a\alpha$$

where $a \in T$, $X \in V$

$\alpha \in V^*$

Right hand side of every productions starts with a terminal, followed by a string of variables of zero/more length.

Problems:

1) convert the given CFG to GNF

$S \rightarrow ABA$

$A \rightarrow aA \mid \epsilon$

$B \rightarrow bB \mid \epsilon$

Solution:

Simplify the CFG, Eliminate ϵ production $A \rightarrow \epsilon$, $B \rightarrow \epsilon$

$S \rightarrow ABA \mid AB \mid BA \mid AA \mid A \mid B$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

Eliminate unit productions,

$S \rightarrow ABA \mid AB \mid BA \mid AA \mid aA \mid a \mid bB \mid b$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

Apply GNF rules,

$S \rightarrow \underline{A}BA$

$S \rightarrow aABA \mid aBA$

$S \rightarrow \underline{A}B$

$S \rightarrow aAB \mid aB$

$S \rightarrow BA$

$S \rightarrow bBA \mid bA$

$S \rightarrow AA$

$S \rightarrow aAA \mid aA$

∴ The resultant productions are,

$S \rightarrow aABA \mid aBA \mid aAB \mid aB \mid bBA \mid bA$

$S \rightarrow aAA \mid aA \mid a \mid b$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

2) convert given CFG to GNF where $V = \{S, A\}$, $T = \{0, 1\}$ and P is

$S \to AA \mid 0$

$A \to SS \mid 1$

Solution: Replace S as A_1 and A as A_2

CFG becomes, $A_1 \to A_2 A_2 \mid 0$

$A_2 \to A_1 A_1 \mid 1$

start with $A_2 \to A_1 A_1 \mid 1$

$A_2 \to A_2 A_2 A_1 \mid 0A_1 \mid 1$

left recursion.

Introduce B_2 to eliminate left recursion $\Rightarrow$ $A_2 \to A_2 A_2 A_1$, $A_2 \to 0A_1 \mid 1$

$B_2 \to A_2 A_1 \mid A_2 A_1 B_2$, $A_2 \to 0A_1 B_2 \mid 1B_2$

The productions are

$A_2 \to 0A_1 \mid 1$

$A_2 \to 0A_1 B_2 \mid 1B_2$ $\Rightarrow$ $A_1 \to A_2 A_2 \mid 0$

$A_1 \to 0A_1 A_2 \mid 1A_2 \mid 0A_1 B_2 A_2 \mid 1B_2 A_2 \mid 0$ (GNF)

$B_2 \to A_2 A_1$

$B_2 \to 0A_1 \mid 1A_1 \mid 0A_1 B_2 A_1 \mid 1B_2 A_1$ (GNF)

$B_2 \to A_2 A_1 B_2$

$B_2 \to 0A_1 A_1 B_2 \mid 1A_1 B_2 \mid 0A_1 B_2 A_1 B_2 \mid 1B_2 A_1 B_2$ (GNF)

Converting Back $A_1 = S$ and $A_2 = A$

$S \to 0SA \mid 1A \mid 0SB_2 A \mid 1B_2 A \mid 0$

$A \to 0S \mid 1 \mid 0SB_2 \mid 1B_2$

$B_2 \to 0SS \mid 1S \mid 0SB_2 S \mid 1B_2 S$

$B_2 \to 0SSB_2 \mid 1SB_2 \mid 0SB_2 SB_2 \mid 1B_2 SB_2$

3) Convert the grammar $S \to AB$, $A \to BS|b$, $B \to SA|a$ into Greibach Normal form.

Solution:

Consider the grammar,

$S \to AB$, $A \to BS|b$, $B \to SA|a$

Assume $S=A_1$, $A=A_2$, and $B=A_3$.

$A_1 \to A_2A_3$

$A_2 \to A_3A_1|b$

$A_3 \to A_1A_2|a$

Consider, $A_3 \to A_1A_2|a$

$A_3 \to A_2A_3A_2|a$

$A_3 \to A_3A_1A_3A_2|bA_3A_2|a$

Now, $A_3 \to A_3A_1A_3A_2$, $A_3 \to bA_3A_2|a$

Introduce B_3 to eliminate left recursion $\Rightarrow$ $B_3 \to A_1A_3A_2|A_1A_3A_2B_3$

$A_3 \to bA_3A_2|a|bA_3A_2B_3|aB_3$. ($G_{NF}$)

$A_2 \to A_3A_1|b$

$A_2 \to bA_3A_2A_1|aA_1|bA_3A_2B_3A_1|aB_3A_1|b$ (G_{NF})

$A_1 \to A_2A_3$

$A_1 \to bA_3A_2A_1A_3|aA_1A_3|bA_3A_2B_3A_1A_3|aB_3A_1A_3|bA_3$

$B_3 \to A_1A_3A_2$

$\to bA_3A_2A_1A_3A_3A_2|aA_1A_3A_3A_2|bA_3A_2B_3A_1A_3A_3A_2|$

$aB_3A_1A_3A_3A_2|bA_3A_3A_2$.

$B_3 \rightarrow A_1A_3A_2B_3$

$\rightarrow bA_3A_2A_1A_3A_3A_2B_3 \mid aA_1A_3A_3A_2B_3 \mid bA_3A_2B_3A_1A_3A_2B_3 \mid aB_3A_1A_3A_2B_3 \mid bA_3A_3A_2B_3$.

∴ The resultant productions are,

$A_1 \rightarrow bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_2B_3A_1A_3 \mid bA_3$

$A_2 \rightarrow bA_3A_2A_1 \mid aA_1 \mid bA_3A_2B_3A_1 \mid aB_3A_1 \mid b$

$A_3 \rightarrow bA_3A_2 \mid a \mid bA_3A_2B_3 \mid aB_3$

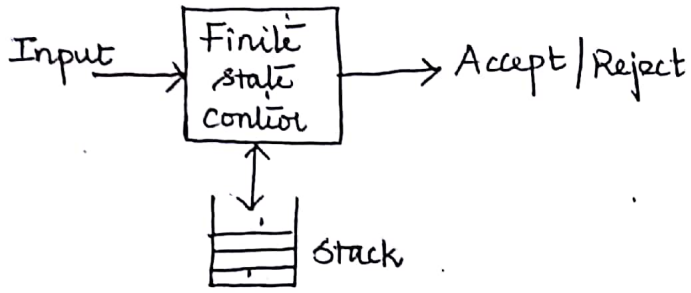
$B_3 \rightarrow bA_3A_3A_2bA_3A_2A_1A_3A_2 \mid aA_1A_3A_3A_2 \mid$
$bA_3A_2B_3A_1A_3A_3A_2 \mid aB_3A_1A_3A_3A_2 \mid$
$bA_3A_2A_1A_3A_3A_2B_3 \mid aA_1A_3A_3A_2B_3 \mid$
$bA_3A_2B_3A_1A_3A_3A_2B_3 \mid aB_3A_1A_3A_2B_3 \mid bA_3A_3A_2B_3$.

UNIT-III PUSHDOWN AUTOMATA

Pushdown Automata:

⇒ The pushdown automaton is essentially a finite automaton with control of both an input tape and a stack on which it can store a string of stack symbols.

⇒ With the help of a stack, the pushdown automaton can remember an infinite amount of information.



⇒ PDA consists of a finite set of states, a finite set of input symbols and a finite set of pushdown symbols.

⇒ The finite control has control of both the input tape and the pushdown store.

⇒ In one transition of the pushdown automaton,

- The control head reads the input symbol, then goto the new state.
- Replaces the symbol at the top of the stack by any string.

Definition of PDA:

A pushdown automaton consists of seven tuples

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

Where,

Q - A finite non empty set of states

Σ - A finite set of input symbols.

$\vdash$ - A finite non empty set of stack symbols.

q_0 - q_0 in Q is the start state

z_0 - Initial start symbol of the stack.

F - $F \subseteq Q$, set of accepting states or final states

δ - Transition function $Q \times (\Sigma \cup \{\varepsilon\}) \times \vdash \rightarrow Q \times \vdash^*$

Moves: The interpretation of

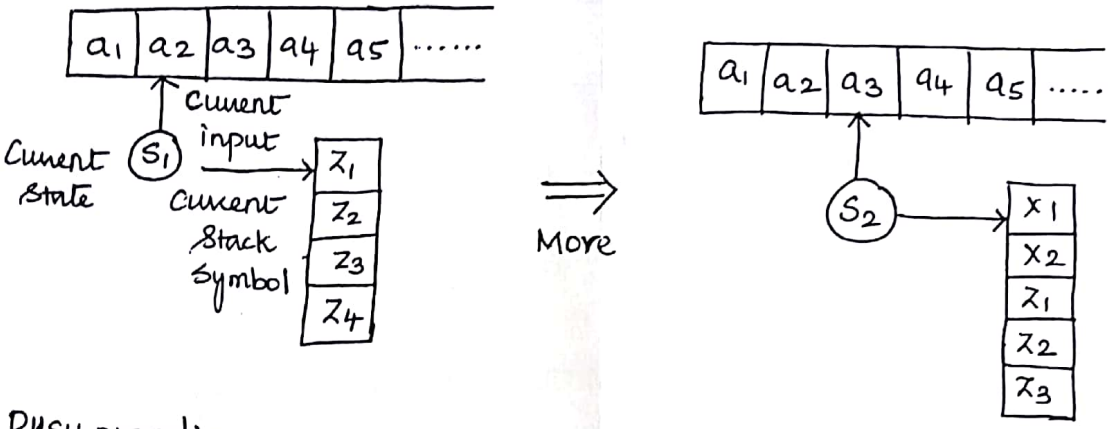
$$\delta(q, a, Z) = \{(P_1, \gamma_1), (P_2, \gamma_2) \ldots (P_M, \gamma_M)\}$$

Where q, P_i – states a – input symbol Z – stack symbol

γ_i – a symbol in Γ^*

PDA enters state P_i, replaces the symbol Z by the string γ_i and advances the input head one symbol.

Instantaneous Descriptions (ID)



PUSH operation:

Current stack top

$$\delta(q_0, X, Z_0) = \delta(q_1, XZ_0)$$

→ push X onto the stack.

Read input on the tape

Change the state from q_0 to q_1

POP operation:

$$\delta(q_0, x, y) = \delta(q_1, \xi)$$

→ Pop the stack.

Read input on tape

Current stack top

Change the state from q_0 to q_1

Problem: PDA Constructions

1) Design a PDA for accepting a language $L = \{a^n b^n \mid n \geq 1\}$

Solution:

Logic: First we will push all a's onto the stack. Then reading every single b each a is popped from the stack.

If we read all b and remove all a's and if we get stack empty then that string will be accepted.

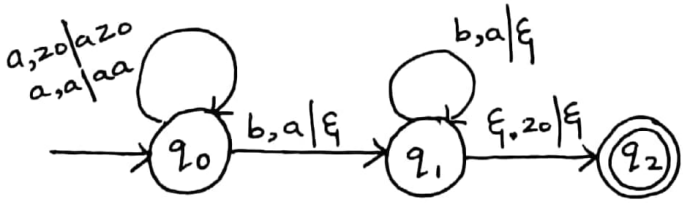
Instantaneous Description:

$\delta(q_0, a, z_0) = \{(q_0, az_0)\}$
$\delta(q_0, a, a) = \{(q_0, aa)\}$ } pushing the elements onto stack

$\delta(q_0, b, a) = \{(q_1, \epsilon)\}$
$\delta(q_1, b, a) = \{(q_1, \epsilon)\}$
$\delta(q_1, \epsilon, z_0) = \{(q_2, \epsilon)\}$ } popping the elements

PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, \{q_2\})$

Where $Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b\}$
$\Gamma = \{a, z_0\}$



Example: Let $n=2$, String $w = a^2 b^2 = aabb$

$\delta(q_0, aabb, z_0) \vdash (q_0, aabb, z_0)$
$\vdash (q_0, abb, az_0)$
$\vdash (q_0, bb, aaz_0)$
$\vdash (q_1, b, az_0)$
$\vdash (q_1, \epsilon, z_0)$
$\vdash (q_2, \epsilon)$ Accept state.

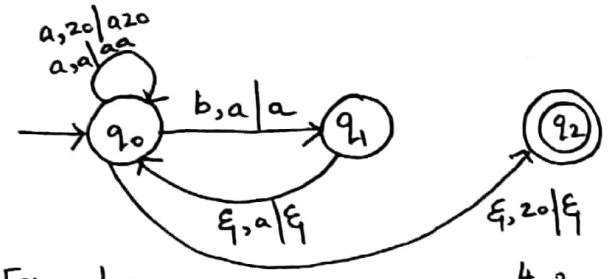
4) Construct the PDA for the language $L=\{a^{2n}b^n \mid n \geq 1\}$. Trace your PDA for the input with $n=2$.

Solution:

Logic: When we read single 'b', single 'a' popped from the stack.
for reading ε also single 'b' popped from the stack.

Instantaneous description:

$\delta(q_0,a,z_0)=\{(q_0,az_0)\}$ ↑ PUSH
$\delta(q_0,a,a)=\{(q_0,aa)\}$ ↓
$\delta(q_0,b,a)=\{(q_1,a)\}$ ↑ POP
$\delta(q_1,\varepsilon,a)=\{(q_0,\varepsilon)\}$ ↓
$\delta(q_0,\varepsilon,z_0)=\{(q_2,\varepsilon)\}$ ↓



PDA $P=(Q,\Sigma,\Gamma,\delta,q_0,z_0,\{q_2\})$

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

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

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

Example: Let $n=2$ string $w=a^4b^2=aaaabb$

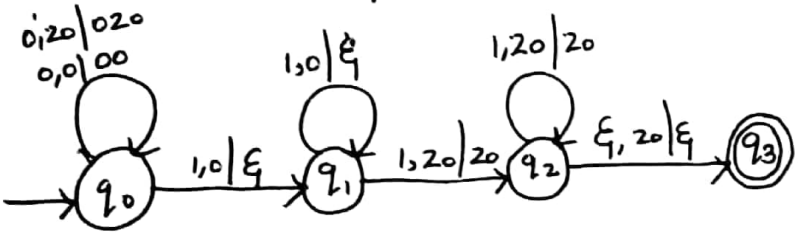
$\delta(q_0,aaaabb,z_0) \vdash (q_0,aaaabb,z_0)$
$\vdash (q_0,aaabb,az_0)$
$\vdash (q_0,aabb,aaz_0)$
$\vdash (q_0,abb,aaaz_0)$
$\vdash (q_0,bb,aaaaz_0)$
$\vdash (q_1,b,aaaz_0)$
$\vdash (q_0,b,aaz_0)$
$\vdash (q_1,\varepsilon,az_0)$
$\vdash (q_0,\varepsilon,z_0)$
$\vdash (q_2,\varepsilon)$ Accept state.

5) Construct the DPDA for the language $L=\{0^n1^m \mid n<m \text{ and } n,m\geq 1\}$

Solution:

ID:

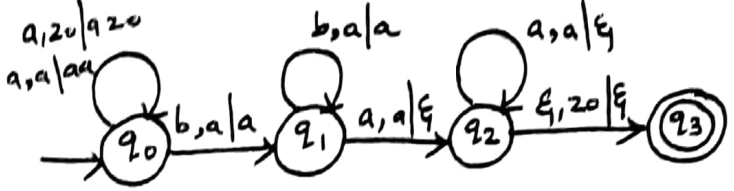
$\delta(q_0,0,z_0)=\{(q_0,0z_0)\}$
$\delta(q_0,0,0)=\{(q_0,00)\}$
$\delta(q_0,1,0)=\{(q_1,\varepsilon)\}$
$\delta(q_1,1,0)=\{(q_1,\varepsilon)\}$
$\delta(q_1,1,z_0)=\{(q_2,z_0)\}$
$\delta(q_2,1,z_0)=\{(q_2,z_0)\}$
$\delta(q_2,\varepsilon,z_0)=\{(q_3,\varepsilon)\}$



Example: $\delta(q_0,00111,z_0) \vdash (q_0,00111,z_0)$
$\vdash (q_0,0111,0z_0)$
$\vdash (q_0,111,00z_0)$
$\vdash (q_1,11,0z_0)$
$\vdash (q_1,1,z_0)$
$\vdash (q_2,\varepsilon,z_0)$
$\vdash (q_3,\varepsilon)$ Accept state

(6) Construct the PDA for the language $L=\{a^n b^m a^n \mid m,n \geq 1\}$ (4)

Solution:



ID:

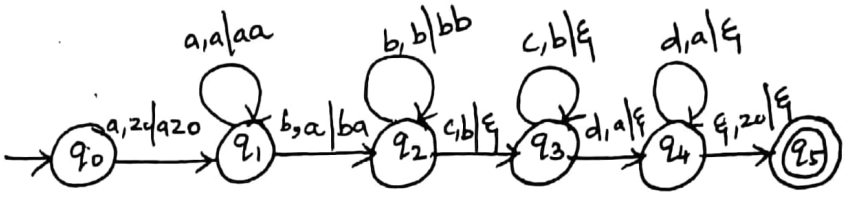
$\delta(q_0,a,z_0)=\{(q_0,az_0)\}$
$\delta(q_0,a,a)=\{(q_0,aa)\}$
$\delta(q_0,b,a)=\{(q_1,a)\}$
$\delta(q_1,b,a)=\{(q_1,a)\}$
$\delta(q_1,a,a)=\{(q_2,\varepsilon)\}$
$\delta(q_2,a,a)=\{(q_2,\varepsilon)\}$
$\delta(q_2,\varepsilon,z_0)=\{(q_3,\varepsilon)\}$

Example: $n=2, m=2$ string $w=a^2b^2a^2=aabbaa$

$\delta(q_0,aabbaa,z_0) \vdash (q_0,abbaa,az_0)$
$\vdash (q_0,bbaa,aaz_0)$
$\vdash (q_1,baa,aaz_0)$
$\vdash (q_1,aa,aaz_0)$
$\vdash (q_2,a,az_0)$
$\vdash (q_2,\varepsilon,z_0) \vdash (q_3,\varepsilon)$ Accept state.

(7) Construct the PDA for the language $L=\{a^n b^m c^m d^n \mid m,n \geq 1\}$

Solution



ID:

$\delta(q_0,a,z_0)=\{(q_1,az_0)\}$
$\delta(q_1,a,a)=\{(q_1,aa)\}$
$\delta(q_1,b,a)=\{(q_2,ba)\}$
$\delta(q_2,b,b)=\{(q_2,bb)\}$
$\delta(q_2,c,b)=\{(q_3,\varepsilon)\}$
$\delta(q_3,c,b)=\{(q_3,\varepsilon)\}$
$\delta(q_3,d,a)=\{(q_4,\varepsilon)\}$
$\delta(q_4,d,a)=\{(q_4,\varepsilon)\}$
$\delta(q_4,\varepsilon,z_0)=\{(q_5,\varepsilon)\}$

Example:

$\delta(q_0,aabbccdd,z_0) \vdash (q_0,abbccdd,az_0)$
$\vdash (q_1,bbccdd,aaz_0)$
$\vdash (q_1,bccdd,baaz_0)$
$\vdash (q_2,ccdd,bbaaz_0)$
$\vdash (q_3,cdd,baaz_0)$
$\vdash (q_3,dd,aaz_0)$
$\vdash (q_4,d,az_0)$
$\vdash (q_4,\varepsilon,z_0)$
$\vdash (q_5,\varepsilon)$ Accept state.

Deterministic pushdown automata

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ is deterministic if and only if it satisfies the following condition.

(i) $\delta(q, a, x)$ has at most one element

(ii) If $\delta(q, a, x)$ is nonempty for some $a \in \Sigma$ then $\delta(q, \epsilon, x)$ must be empty.

Non-Deterministic pushdown Automata

The non-deterministic pushdown automata is very much similar to NFA. The CFG which accepts deterministic PDA accepts non-deterministic PDAs as well.

Similarly there are some CFG's which can be accepted only by NDPA and not by DPDA. Thus NDPA is more powerful than DPDA.

Compare NFA and PDA.

NFA	PDA
1. NFA stands for non-deterministic finite automata.	PDA stands for pushdown automata.
2. This model does not have memory to remember input symbols.	This model has stack memory to remember input symbols.
3. It is always non-deterministic. It has two versions. (i) NFA with ϵ (ii) NFA without ϵ.	It has two versions. (i) Deterministic PDA (ii) Non-deterministic PDA.

Equivalence : pushdown automata to CFL.

Let, $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, q_n)$ is a PDA there exists CFG G_1 which is accepted by PDA P. The G_1 can be defined as,

$$G_1 = (V, T, P, S)$$

Where S is a start symbol, T – Terminals V – Non-terminals

For getting production rules P, we follow the following algorithm.

Algorithm for getting production rules of CFG

1. If q_0 is start state in PDA and q_n is final state of PDA then $[q_0\ z\ q_n]$ becomes start state of CFG.
2. The production rule for the ID of the form $\delta(q_i, a, z_0) = (q_{i+1}, z_1 z_2)$ can be obtained as,

$$\delta(q_i\ z_0\ q_{i+k}) \rightarrow a\,(q_{i+1}\ z_1\ q_m)\,(q_m\ z_2\ q_{i+k})$$

Where q_{i+k}, q_m represents the intermediate states, z_0, z_1, z_2 are stack symbols and a is input symbol.

3. The production rule for the ID of the form:

$\delta(q_i, a, z_0) = (q_{i+1}, \epsilon)$ can be converted as $(q_i\ z_0\ q_{i+1}) \rightarrow a$

Problems: PDA to CFG

1) Let $M = (\{q_0, q_1\}, \{0, 1\}, \{X, z_0\}, \delta, q_0, z_0, \phi)$ where δ is given by

$$\delta(q_0, 0, z_0) = \{(q_0, X z_0)\}$$
$$\delta(q_0, 0, X) = \{(q_0, XX)\}$$
$$\delta(q_0, 1, X) = \{(q_1, \epsilon)\}$$
$$\delta(q_1, 1, X) = \{(q_1, \epsilon)\}$$
$$\delta(q_1, \epsilon, X) = \{(q_1, \epsilon)\}$$
$$\delta(q_1, \epsilon, z_0) = \{(q_1, \epsilon)\}$$

Construct CFG $G_1 = (V, T, P, S)$ generating $N(M)$.

Solution: Given $M = (\{q_0, q_1\}, \{0, 1\}, \{X, Z_0\}, \delta, q_0, Z_0, \phi)$

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

$T = \{0, 1\}$

$V = (S, [q_0, X, q_0], [q_0, X, q_1], [q_1, X, q_0], [q_1, X, q_1]$
$[q_0, Z_0, q_0], [q_0, Z_0, q_1], [q_1, Z_0, q_0], [q_1, Z_0, q_1])$

Start state production S

$S \to [q_0, Z_0, q_0]$
$S \to [q_0, Z_0, q_1]$

Now productions for $[q_0, Z_0, q_0]$ and $[q_0, Z_0, q_1]$.

(1) $\delta(q_0, 0, Z_0) = \{(q_0, X Z_0)\}$

$[q_0, Z_0, q_0] \to 0\,[q_0, X, q_0]\,[q_0, Z_0, q_0]$
$[q_0, Z_0, q_0] \to 0\,[q_0, X, q_1]\,[q_1, Z_0, q_0]$
$[q_0, Z_0, q_1] \to 0\,[q_0, X, q_0]\,[q_0, Z_0, q_1]$
$[q_0, Z_0, q_1] \to 0\,[q_0, X, q_1]\,[q_1, Z_0, q_1]$

(2) $\delta(q_0, 0, X) = \{(q_0, XX)\}$

$[q_0, X, q_0] \to 0\,[q_0, X, q_0]\,[q_0, X, q_0]$
$[q_0, X, q_0] \to 0\,[q_0, X, q_1]\,[q_1, X, q_0]$
$[q_0, X, q_1] \to 0\,[q_0, X, q_0]\,[q_0, X, q_1]$
$[q_0, X, q_1] \to 0\,[q_0, X, q_1]\,[q_1, X, q_1]$

(iii) $\delta(q_1, 1, X) = (q_1, \epsilon)$

$[q_1, X, q_1] \to 1$

(iv) $\delta(q_1, \epsilon, X) = (q_1, \epsilon)$

$[q_1, X, q_1] \to \epsilon$

(v) $\delta(q_0, 1, X) = (q_1, \epsilon)$

$[q_0, X, q_1] \to 1$

(vi) $\delta(q_1, \epsilon, Z_0) = (q_1, \epsilon)$

$[q_1, Z_0, q_1] \to \epsilon$

After analysing all the productions

useless production $\Rightarrow [q_0, Z_0, q_0]\ [q_0, X, q_0]$

Unknown productions $\Rightarrow [q_1, Z_0, q_0]\ [q_1, X, q_0]$

Deleting all these productions, final productions are.

$S \to [q_0, Z_0, q_1]$

$[q_0, Z_0, q_1] \to 0\,[q_0, X, q_1]\,[q_1, Z_0, q_1]$
$[q_0, X, q_1] \to 0\,[q_0, X, q_1]\,[q_1, X, q_1]$
$[q_0, X, q_1] \to 1$
$[q_1, Z_0, q_1] \to \epsilon$
$[q_1, X, q_1] \to \epsilon$
$[q_1, X, q_1] \to 1$

After Renaming

$S \to A$
$A \to 0BC$
$B \to 0BD \mid 1$
$C \to \epsilon$
$D \to 1 \mid \epsilon$

2) Let $M = (\{q_0, q_1\}, \{a, b\}, \{z, z_0\}, \delta, q_0, z_0, \phi)$ where δ is given by

$\delta(q_0, b, z_0) = \{(q_0, zz_0)\}$ $\quad$ $\delta(q_0, a, z) = \{(q_1, z)\}$

$\delta(q_0, \epsilon, z_0) = \{(q_0, \epsilon)\}$ $\quad$ $\delta(q_1, b, z) = \{(q_1, \epsilon)\}$

$\delta(q_0, b, z) = \{(q_0, zz)\}$ $\quad$ $\delta(q_1, a, z_0) = \{(q_0, z_0)\}$

Construct CFG $G = (V, T, P, S)$ generating $N(M)$.

Solution:

Given $M = (\{q_0, q_1\}, \{a, b\}, \{z, z_0\}, \delta, q_0, z_0, \phi)$

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

$T = \{a, b\}$

$V = S, [q_0, z, q_0], [q_0, z, q_1], [q_1, z, q_0], [q_1, z, q_1]$
$[q_0, z_0, q_0], [q_0, z_0, q_1], [q_1, z_0, q_0], [q_1, z_0, q_1])$

Start state production S

$S \to [q_0, z_0, q_0]$

$S \to [q_0, z_0, q_1]$

Now productions for $[q_0, z_0, q_0]$ and $[q_0, z_0, q_1]$

(i) $\delta(q_0, b, z_0) = \{(q_0, zz_0)\}$

$[q_0, z_0, q_0] \to b\,[q_0, z, q_0]\,[q_0, z_0, q_0]$

$[q_0, z_0, q_0] \to b\,[q_0, z, q_1]\,[q_1, z_0, q_0]$

$[q_0, z_0, q_1] \to b\,[q_0, z, q_0]\,[q_0, z_0, q_1]$

$[q_0, z_0, q_1] \to b\,[q_0, z, q_1]\,[q_1, z_0, q_1]$

(ii) $\delta(q_0, \epsilon, z_0) = \{(q_0, \epsilon)\}$

$[q_0, z_0, q_0] \to \epsilon$

(iii) $\delta(q_0, b, z) = \{(q_0, zz)\}$

$[q_0, z, q_0] \to b\,[q_0, z, q_0]\,[q_0, z, q_0]$

$[q_0, z, q_0] \to b\,[q_0, z, q_1]\,[q_1, z, q_0]$

$[q_0, z, q_1] \to b\,[q_0, z, q_0]\,[q_0, z, q_1]$

$[q_0, z, q_1] \to b\,[q_0, z, q_1]\,[q_1, z, q_1]$

(iv) $\delta(q_0, a, z) = (q_1, z)$

$[q_0, z, q_0] \to a\,[q_1, z, q_0]$

$[q_0, z, q_1] \to a\,[q_1, z, q_1]$

(v) $\delta(q_1, b, z) = \{(q_1, \epsilon)\}$

$[q_1, z, q_1] \to b$

(vi) $\delta(q_1, a, z_0) = \{(q_0, z_0)\}$

$[q_1, z_0, q_0] \to a\,[q_0, z_0, q_0]$

$[q_1, z_0, q_1] \to a\,[q_0, z_0, q_1]$

After analysing all the productions,

Useless productions $\Rightarrow [q_0, z, q_0]\,[q_0, z_0, q_1]$

Unknown productions $\Rightarrow [q_1, z_0, q_1]\,[q_1, z, q_0]$

Deleting all these productions.

$S \to [q_0, z_0, q_0]$

$[q_0, z_0, q_0] \to b\,[q_0, z, q_1]\,[q_1, z_0, q_0]$

$[q_0, z, q_1] \to b\,[q_0, z, q_1]\,[q_1, z, q_1]$

$[q_0, z_0, q_0] \to \epsilon$

$[q_0, z, q_1] \to a\,[q_1, z, q_1]$

$[q_1, z, q_1] \to b$

$[q_1, z_0, q_0] \to a\,[q_0, z_0, q_0]$

After Renaming

$S \to A$

$A \to bBC \mid \epsilon$

$B \to bBD \mid aD$

$D \to b$

$C \to aA$

3) Construct a PDA accepting $\{a^n b^m a^n \mid m,n \geq 1\}$ by empty stack. Also construct the corresponding context free grammar accepting the same set

4) Let $M = (\{P,q\}, \{0,1\}, \{X, z_0\}, \delta, q, z_0)$ where δ is given by

$\delta(q,1,z_0) = \{(q, Xz_0)\}$ $\quad$ $\delta(q,\varepsilon,z_0) = \{(q,\varepsilon)\}$

$\delta(q,1,X) = \{(q, XX)\}$ $\quad$ $\delta(P,1,X) = \{(P,\varepsilon)\}$

$\delta(q,0,X) = \{(P,X)\}$ $\quad$ $\delta(P,0,z_0) = \{(q,z_0)\}$

Construct CFG $G = (V,T,P,S)$ generating $N(M)$.

Equivalence : CFL to pushdown automata

Algorithm:

(1) Convert the CFG to Greibach Normal form.

(2) The δ function is to be developed for the grammar of the form.

$$A \to aB \text{ as } \delta(q_i, a, A) \to \delta(q_i, B)$$

(3) Finally add the rule

$$\delta(q_i, \varepsilon, z_0) \to (q_i, \varepsilon)$$

Where z_0 – Stack symbol (Accepting state)

Problem 1: Construct PDA for the following grammar.

$$S \to AB,\ B \to b,\ A \to CD,\ C \to a,\ D \to a$$

Solution:

GNF form:

$S \to AB$
$\to CDB$
$\to aDB$

$A \to CD$
$\to aD$

$B \to b$

$C \to a$

$D \to a$

Equivalent PDA is

$\delta(q_1, a, S) \to (q_1, DB)$
$\delta(q_1, a, A) \to (q_1, D)$
$\delta(q_1, b, B) \to (q_1, \varepsilon)$
$\delta(q_1, a, C) \to (q_1, \varepsilon)$
$\delta(q_1, a, D) \to (q_1, \varepsilon)$

Example: $\delta(q_1, aab, S) \vdash \delta(q_1, ab, DB)$
$\vdash \delta(q_1, b, B)$
$\vdash \delta(q_1, \varepsilon, z_0)$
$\vdash \delta(q_f, \varepsilon)$ Accepting state.

Problem 2: Construct an unrestricted PDA equivalent of the grammar given below (

$S \to aAA$, $A \to aS \mid bS \mid a$

Solution: The given grammar is already in GNF. Hence the PDA can be-

$\delta(q_1, a, S) \to (q_1, AA)$

$\delta(q_1, a, A) \to (q_1, S)$

$\delta(q_1, b, A) \to (q_1, S)$

$\delta(q_1, a, A) \to (q_1, \epsilon)$

$\delta(q_1, \epsilon, z_0) \to (q_1, \epsilon)$ Accept.

The simulation of abaaaa is,

$\delta(q_1, abaaaa, S) \vdash \delta(q_1, baaaa, AA)$

$\vdash \delta(q_1, aaaa, SA)$

$\vdash \delta(q_1, aaa, AAA)$

$\vdash \delta(q_1, aa, AA)$

$\vdash \delta(q_1, a, A)$

$\vdash \delta(q_1, \epsilon, z_0)$

$\vdash \delta(q_1, \epsilon)$ Accept.

Problem 3: Consider GNF $G = (\{S, T, C, D\}, \{a, b, c, d\}, S, P)$ where P is.

$S \to cCD \mid dTC \mid \epsilon$ $\quad C \to aTD \mid c$

$T \to cDC \mid cST \mid a$ $\quad D \to d\epsilon \mid d$

Present a PDA that accepts the language generated by this grammar.

Solution: Let PDA $M = \{\{q\}, \{c, a, d\}, \{S, T, C, D, c, d, a\}, \delta, q, S, \phi\}$

The production rules δ is given by

$\delta(q, \epsilon, S) = \{(q, CCD), (q, dTC), (q, \epsilon)\}$

$\delta(q, \epsilon, C) = \{(q, aTD), (q, c)\}$

$\delta(q, \epsilon, T) = \{(q, cDC), (q, cST), (q, a)\}$

$\delta(q, \epsilon, D) = \{(q, d\epsilon), (q, d)\}$

$\delta(q, c, c) = \{(q, \epsilon)\}$

$\delta(q, d, d) = \{(q, \epsilon)\}$

$\delta(q, a, a) = \{(q, \epsilon)\}$

} Acceptance by Empty stack

Simulation for String "caadd"

$\delta(q, \epsilon, S) \vdash \delta(q, caadd, S)$

$\vdash \delta(q, aadd, CD)$

$\vdash \delta(q, add, TDD)$

$\vdash \delta(q, dd, DD)$

$\vdash \delta(q, d, D)$

$\vdash (q, \epsilon)$ Accept.

Problem 4: Find the PDA equivalent to given CFG with following productions.

$S \to A$; $A \to BC$, $B \to ba$, $C \to dC$

Problem 5: convert the grammar $S \to aSb \mid A$, $A \to bSa \mid S \mid \epsilon$ to a PDA that accepts the same language by empty stack.

Problem 6: convert the grammar $S \to 0S1 \mid A$, $A \to 1A0 \mid S \mid \epsilon$ into PDA that accepts the same language by empty stack. check whether 0101 belongs to $N(M)$.

Problem 7: construct CFL for the grammar $S \to aSbb \mid a$ and also construct its corresponding PDA.

Sln: $\{L = a^n b^m \mid m > n\} \longrightarrow$ PDA

Problem 8: construct CFL for the grammar $S \to aSa \mid bSb \mid \epsilon$ and also construct its corresponding PDA.

Sln: $L = \{WW^R \mid W \text{ is in } (a+b)^*\} \longrightarrow$ PDA.

Problem 9: construct CFL for the grammar $S \to aSb \mid A$, $A \to bSa \mid s \mid \epsilon$ and also construct its corresponding PDA.

Sln: $L = \{a^n b^n \mid n \geq 1\} \longrightarrow$ PDA.

Problem 10: convert the grammar $E \to E+E$, $E \to id$ into PDA and trace the string "id+id+id".

Pumping lemma for CFL

Lemma: Let L be any CFL. Then there is a constant n, depending only on L, such that z is in L and $|z| \geq n$, then we can write $z = uvxyz$ such that

(i) $|vxy| \leq n$

(ii) $|vy| \geq 1$ (or) $|vy| \neq \epsilon$

(iii) for all $i \geq 0$, $uv^ixy^iz \in L$.

Problems:

1) Prove that $L = \{a^ib^ic^i \mid i \geq 1\}$ is not context free language.

Solution:

(i) Let us assume that L is regular / CFL

(ii) Let $w = a^ib^ic^i$ where i is constant

(iii) W can be written as $uvxyz$ where

(a) $|vxy| \leq n$

(b) $|vy| \neq \epsilon$

(c) for all $i \geq 0$, $uv^ixy^iz \in L$

Since $vy \neq \epsilon$, either $v = ab|bc|ca$ (or)
$y = ab|bc|ca$.

If $i=2$, $uv^ixy^iz = uv^2xy^2z$ becomes

Case (i) If $v=ab$ and $y=c$

$$uv^2xy^2z = (ab)^2c^2 \Rightarrow uv^ixy^iz \notin L$$

Case (ii) If $v=a$ and $y=bc$

$$uv^2xy^2z = a^2(bc)^2 \Rightarrow uv^ixy^iz \notin L$$

Hence L is not a CFL.

3) Prove that $L=\{a^i b^m c^p \mid 0 \le n < m < p\}$ is not CFL.

Solution:

(i) Let $L=\{a^n b^m c^p \mid 0 \le n < m < p\}$

(ii) Let $w = a^n b^{n+1} c^{n+2}$, where n is constant
[since $n < m < p$]

(iii) w can be rewritten as uvxyz where

(a) $|vxy| \le n$

(b) $vy \neq \epsilon$

(c) for all $i \ge 0$, $uv^i xy^i z \in L$.

Case(i) If $v=ab$, and $y=c$

$i=2$, $uv^2xy^2z = (ab)^2 c^2$ — ①

Case(ii) If $v=a$ and $y=bc$

$i=2$, $uv^2xy^2z = a^2 (bc)^2$ — ②

From ① and ② $uv^i xy^i z \notin L$.

Hence L is not a CFL.

4) Prove that $L=\{0^i 1^j 2^i 3^j \mid i \ge 1, j \ge 1\}$ is not CFL.

Solution:

(i) Let us assume that L is CFL

(ii) Let $w = 0^n 1^n 2^n 3^n$ where n is constant.

(iii) Let w can be rewritten as, uvxyz where,

(a) $|vxy| \le n$

(b) $|vy| \neq \epsilon$

(c) for all $i \ge 0$, $uv^i xy^i z \in L$.

Case(i) if $v=01$ and $y=2$

$i=2$, $uv^2xy^2z = (01)^2 2^2 3^1$ — ①

Case(ii) if $v=12$ and $y=3$

$i=2$, $uv^2xy^2z = (12)^2 (3)^2 0^1$ — ②

From ① and ② $uv^i xy^i z \notin L$

∴ Given L is not a CFL.

5) Prove that $L=\{a^i b^m c^p \mid 0 \le n < m < p\}$ is not CFL.

Solution:

(i) Let us assume that L is CFL

(ii) let $w=a^i b^j c^j$, where i, j is a constant

(iii) w can be written as, uvxyz where

(a) $|vxy| \le n$

(b) $vy \neq \epsilon$

(c) for all $i \ge 0$, $uv^i xy^i z \in L$.

Case(i) If $v=ab$ and $y=c$

$i=2$, $uv^2xy^2z = (ab)^2 c^2 \notin L$

since, Power of c should be greater than ab

Case(ii) If $v=a$ and $y=bc$

$uv^2xy^2z = (a)^2 (bc)^2 \notin L$, $j > i$ is not true

Hence L is not a CFL.

5) Design a TM that perform multiplication operation using "copy" subroutine.

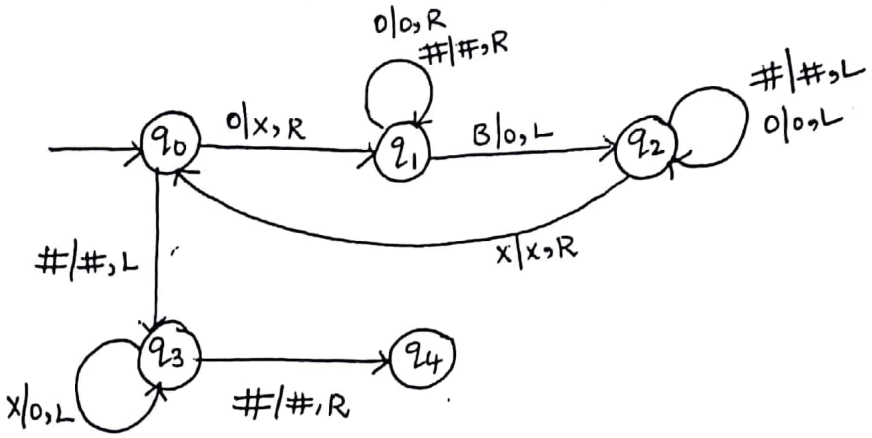
Solution:

Let the inputs be 0^x and 0^y

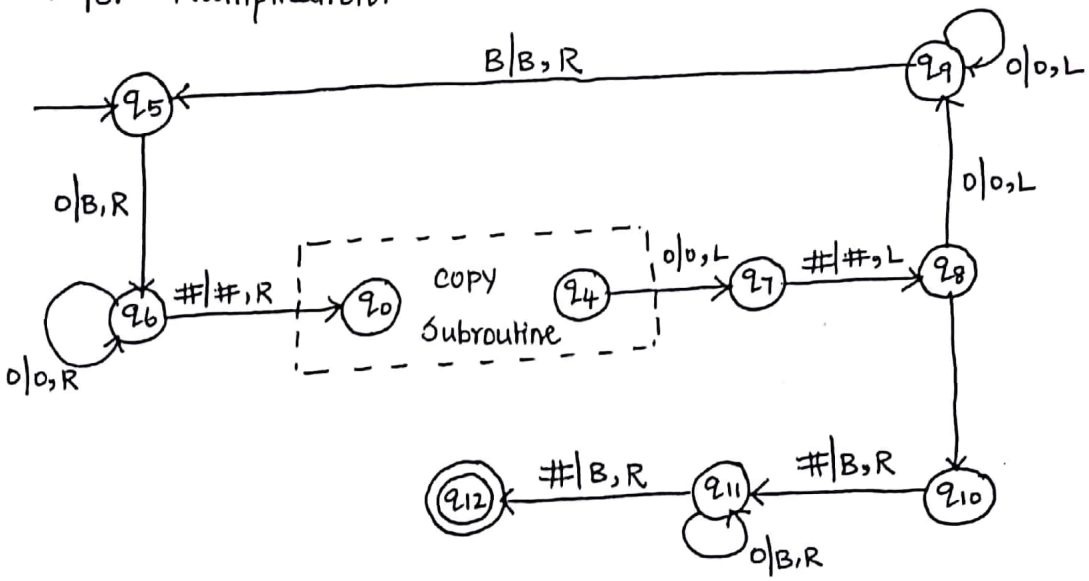
The inputs are stored as $B0^x\#0^y\#BB$.

Turing Machine:

Subroutine for copy operation.



TM for Multiplication:



Turing Machine constructions

A turing machine is also as powerful as a conventional computer. The following are the different technique of constructing a TM.

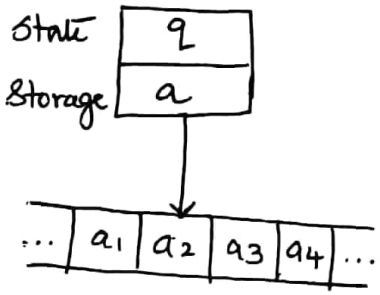
1. Storage in the finite control (or) State
2. Multiple tracks
3. Subroutines
4. Checking off symbols.

Storage in the finite control:

⇒ The states of turing machine are to remember / store a symbol. This is done by the finite control.

⇒ The finite control can also be used to hold a finite amount of information along with the task of representing a position in the program.

⇒ The state is written as a pair of elements, one for control and the other storing a symbol.



Multiple tracks:

⇒ It is also possible that a turing machine, input tape can be divided into several tracks.

⇒ Each track can hold one symbol, and the tape alphabet of the TM consists of tuples with one component for each track.

⊢	1	0	1	1	$	B	B		
B	B	B	B	1	0	B	B	B	B
	B	B	1	0	1	1	B	B	

Subroutines:

⇒ Subroutines are subfunctions that can be used to execute repeated tasks for number of times depending on the application.

UNIT-IV TURING MACHINE

Introduction:

⇒ A Turing machine is an "automatic machine" that manipulates the input strings according to the transition rules.

⇒ It was invented by Alan Turing in 1936. Turing machine became the most powerful general model of computation.

Description of a TM / Informal definition:

Turing machine is composed of

1. Input tape with tape head
2. Finite set of transition rules.
3. Output tape with tape head.

⇒ The input tape contains the input symbols. The tape is divided into a number of cells, each cell storing one symbol.

⇒ The input tape can also serve as output tape, that contains the resultant value of the operation done by TM.

⇒ The tape head is used to read a symbol from the input tape or write a symbol on the output tape.

⇒ The tape is capable of reading / writing one symbol at a time.

⇒ Several head can be used to perform parallel processes / operations on the head

⇒ The transition rules are responsible for performing the required operations using its rules and based on the input read.

Nature of TM:

The Turing machines are computing machines that are more powerful than the FA and PDA. It is given as,

$$\text{Finite Automata} \subseteq \text{Deterministic PDA} \subseteq \text{NPDA} \subseteq \text{TM}.$$

FA, DPDA, NPDA are less powerful than TM since these machines have no control over the i/p and cannot modify their own inputs.

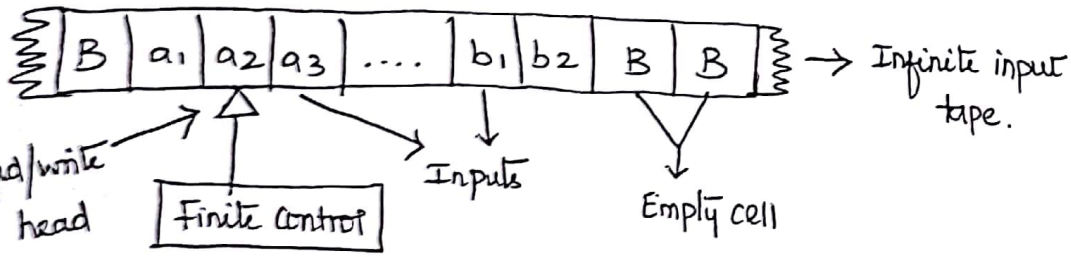
Model of Turing Machine

⇒ TM should be thought of as a finite state machine with a read/write head.

⇒ The input is stored on an input tape that is divided into a number of individual cells.

⇒ The cell contains blank symbol, 'B' when there is no input in it.

⇒ Each cell can store one symbol in it.



⇒ The read/write head of the system reads from and writes data to the i/p tapes.

⇒ The head performs its operation on one cell at a time.

⇒ The movement of the head can be,
- Left (moving backward)
- Right (moving forward)
- None (no movement)

⇒ The head is capable of
- Reading a symbol
- Modifying a symbol.
- Move previous (L) | Move next (R)
- Halt.

Formal Definition of TM.

A Turing machine, M is a 7-Tuple given by,

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

Where
- $Q \rightarrow$ Finite set of states
- $\Sigma \rightarrow$ Finite set of input symbols on input tape
- $\vdash \rightarrow$ Finite set of tape symbols $[\Sigma \cup B]$ [B → blank symbol]
- $\delta \rightarrow$ Transition function given by,

  $(q,a) = (q', b, M)$ M → Movement (left, Right, No movement)
- $q_0 \rightarrow$ Initial state $[q_0 \in Q]$
- $B \rightarrow$ Blank symbol representing empty cell $[B \in \vdash]$
- $F \rightarrow$ set of final states $[F \subseteq Q]$

Instantaneous description for TM.

Instantaneous description for TM is the snapshot of how the input string is processed by the Turing machine.

It describes,

- → The input string
- → position of the head
- → State of the machine

$(q, a) = (q', b, M)$ → Movement $(L/R/N)$.

- q → current state
- a → I/p
- q' → Next state
- b → replaced symbol

Problems:

1) Design a Turing machine that accepts all strings of the form $a^n b^n$ for $n \geq 1$ and rejects all other strings.

Solution:

Procedure:

(i) change the leftmost a to X

(ii) Move forward right side and change the rightmost b to y.

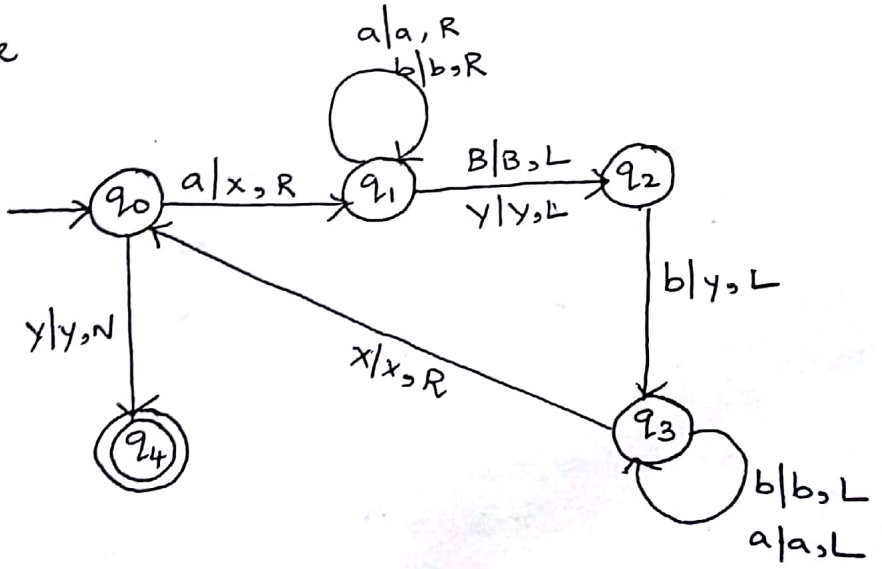
(iii) Move left and change the leftmost a, which is at position immediate right of X, to X.

(iv) Move right and change the rightmost b, which is at immediate left of y, to y

(v) Repeat step (iii) and (iv) until there are no more b's equal to a's.

(vi) Halt the TM.

Turing Machine



Description of the TM.

The turing Machine, M is given by,

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

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

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

$\vdash = \{a, b, B, X, Y\}$

$q_0 = \{q_0\}$

$B = \{B\}$

$F = \{q_4\}$

δ is given by,

$\Sigma \cup \vdash$ / Q	a	b	X	Y	B
$\rightarrow q_0$	(q_1, X, R)	–	–	(q_4, Y, N)	–
q_1	(q_1, a, R)	(q_1, b, R)	–	(q_2, Y, L)	(q_2, B, L)
q_2	–	(q_3, Y, L)	–	–	–
q_3	(q_3, a, L)	(q_3, b, L)	(q_0, X, R)	–	–
$* q_4$	ϕ	ϕ	ϕ	ϕ	ϕ

Instantaneous description for the string w = "aabb"

$\underset{q_0}{aabbB} \vdash X\underset{q_1}{abbB} \vdash X\underset{q_1}{abbB} \vdash X\underset{q_1}{abbB} \vdash X\underset{q_1}{abbB} \vdash X\underset{q_1}{abbB}$

$\vdash X\underset{q_2}{abbB} \vdash X\underset{q_3}{abbB} \vdash X\underset{q_3}{abyB} \vdash X\underset{q_3}{abyB} \vdash X\underset{q_0}{abyB}$

$\vdash Xx\underset{q_1}{byB} \vdash Xx\underset{q_1}{byB} \vdash Xx\underset{q_2}{byB} \vdash Xx\underset{q_3}{yyB} \vdash Xx\underset{q_0}{yyB}$

$\vdash Xxy\underset{q_4}{yB}$ [String Accepted]

2) Design a TM that perform right shift over $\Sigma = \{0,1\}$.

Solution:

To perform: shift the input string right by one place.

Procedure:

(i) Move right side to the last character from the initial character.

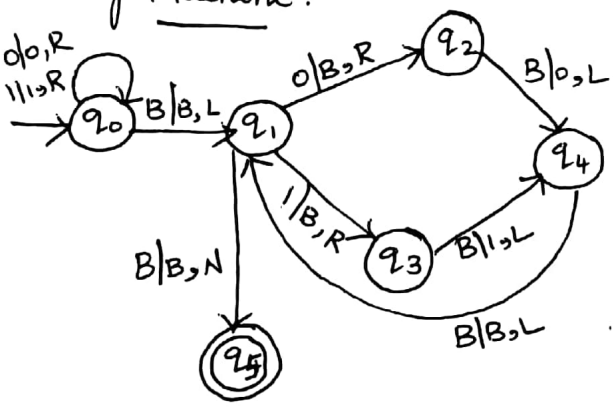
(ii) If the character is '0' replace it with 'B' and move one step right to replace the immediate 'B' to '0'.

(iii) If the character is '1' replace it with 'B' and move one step right to replace the immediate 'B' to 1

(iv) After processing as above, move left one step to perform step (ii) or (iii) on 11 next right-most unprocessed character.

(v) Perform step-(iv) until the all characters are processed.

Turing Machine:



Description of Turing Machine

The Turing Machine is given by,

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

Where,

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$ $q_0 = \{q_0\}$

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

$\Gamma = \{0,1,B\}$ $F = \{q_5\}$

Transition function δ is given by.

Q \ Σ∪Γ	0	1	B
→q_0	$(q_1,0,R)$	$(q_0,1,R)$	(q_1,B,L)
q_1	(q_2,B,R)	(q_3,B,R)	(q_5,B,N)
q_2	-	-	$(q_4,0,L)$
q_3	-	-	$(q_4,1,L)$
q_4	-	-	(q_1,B,L)
*q_5	ϕ	ϕ	ϕ

Instantaneous description for W="0101"

$0101B \vdash 0101B \vdash 0101B \vdash 0101B$
(q_0, q_0, q_0, q_0)

$\vdash 0101B \vdash 0101B \vdash 010BB \vdash 010B1$
(q_0, q_1, q_3, q_4)

$\vdash 010B1 \vdash 01BB1 \vdash 01B01 \vdash 01B01$
(q_1, q_2, q_4, q_1)

$\vdash 0BB01 \vdash 0B101 \vdash 0B101 \vdash BB101$
(q_3, q_4, q_1, q_2)

$\vdash B0101 \vdash BB0101 \vdash B0101$ Accept
(q_4, q_1, q_5)

3) Construct a Turing machine to make a copy of a string over $\Sigma = \{0,1\}$

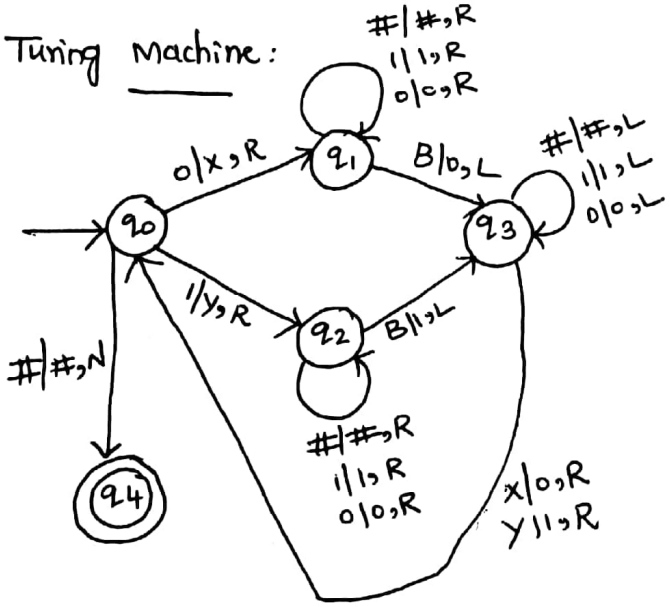
Solution:

Assume: A sequence of input symbols followed by '#' at the end.

Procedure:

1. If the symbol is '0', move right until 'B' is reached. Mark '0' as 'x' to indicate that it is processed.
2. Replace the symbol 'B' by '0'. Move left until 'x' is reached. Mark 'x' as '0' and move right.
3. If the symbol is '1', move right until 'B' is reached. Mark '1' as 'y' to indicate that it is processed.
4. Replace 'B' by 1 and move left until 'y' is reached. Mark 'y' as 1 and move right.
5. Process step 1 and 2 if '0' is the input symbol; else perform step 4 and 5 to process y.
6. Stop and reach state if # is reached.

Turing Machine:



Description of Turing machine

The Turing machine, M is given by

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

Where,

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

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

$\Gamma = \{0,1,x,y,\#,B\}$

$q_0 = \{q_0\}$

$B = \{B\}$

$F = \{q_4\}$

Transition function δ is given by

δ \ $\Sigma\cup\Gamma$	0	1	#	B	X	Y
$\rightarrow q_0$	(q_1, X, R)	(q_2, Y, R)	$(q_4, \#, N)$	–	–	–
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	$(q_1, \#, R)$	$(q_3, 0, L)$	–	–
q_2	$(q_2, 0, R)$	$(q_2, 1, R)$	$(q_2, \#, R)$	$(q_3, 1, L)$	–	–
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	$(q_3, \#, L)$	–	$(q_0, 0, R)$	$(q_0, 1, R)$
$*\ q_4$	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ

Instantaneous description for w = 101

$\vdash$ 101#B $\vdash$ y01#B $\vdash$ y01#B $\vdash$ y01#B $\vdash$ y01#B $\vdash$ y01#1B
$\vdash$ y01#1B $\vdash$ y01#1B $\vdash$ y01#1B $\vdash$ 101#1B $\vdash$ 1x1#1B $\vdash$ 1x1#1B
$\vdash$ 1x1#1B $\vdash$ 1x1#1B $\vdash$ 1x1#10B $\vdash$ 1x1#10B $\vdash$ 1x1#10B $\vdash$ 1x1#10B
$\vdash$ 101#10B $\vdash$ 10y#10B $\vdash$ 10y#10B $\vdash$ 10y#10B $\vdash$ 10y#10B
$\vdash$ 10y#101B $\vdash$ 10y#101B $\vdash$ 10y#101B $\vdash$ 10y#101B $\vdash$ 101#101B
$\vdash$ 101#101B $\Rightarrow$ Accept.

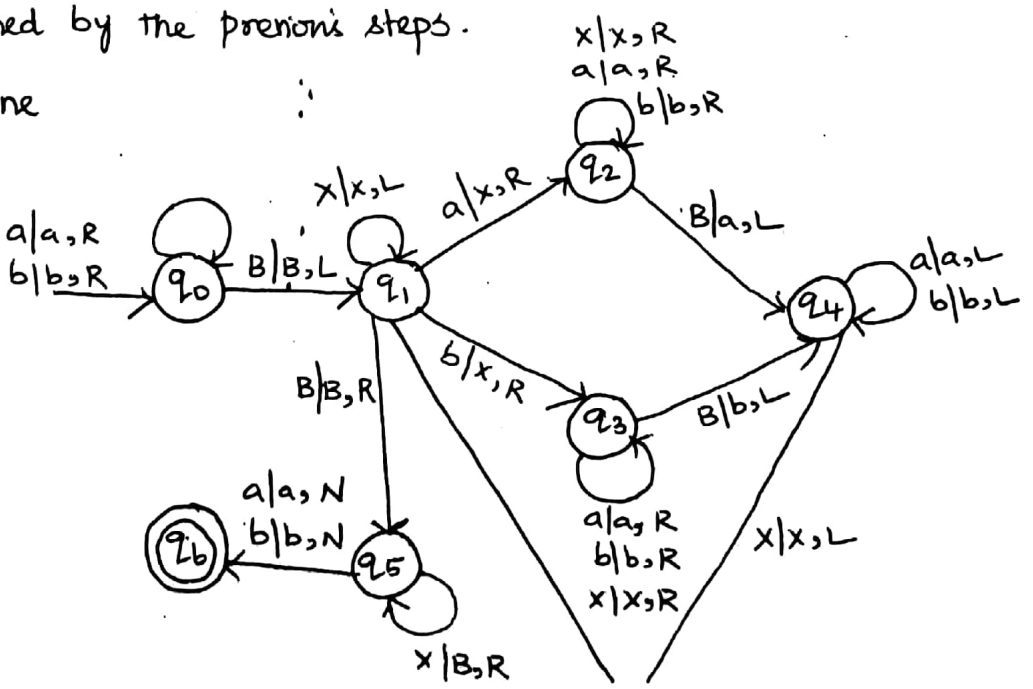
4) Design a TM to reverse a string over $\Sigma = \{a, b\}$

Solution:

Procedure:

1. Move to the last symbol, replace x for a or x for b and move right to convert the corresponding B to 'a' or 'b' accordingly.
2. Move left until the symbol left to x is reached.
3. Perform step 1 and 2 until 'B' is reached while traversing left.
4. Replace every x to B to make the cells empty since the reverse of the string performed by the previous steps.

Turing machine :



Description of TM:

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

where,

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

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

$\vdash = \{a, b, x, B\}$

$q_0 = \{q_0\}$

$B = \{B\}$

$F = \{q_6\}$

Transition function δ is given by.

$\Sigma \cup \vdash$ / Q	a	b	x	B
q_0	(q_0, a, R)	(q_0, b, R)	-	(q_1, B, L)
q_1	(q_2, x, R)	(q_3, x, R)	(q_1, x, L)	(q_5, B, R)
q_2	(q_2, a, R)	(q_2, b, R)	(q_2, x, R)	(q_4, a, L)
q_3	(q_3, a, R)	(q_3, b, R)	(q_3, x, R)	(q_4, b, L)
q_4	(q_4, a, L)	(q_4, b, L)	(q_1, x, L)	-
q_5	(q_6, a, N)	(q_6, b, N)	(q_5, B, R)	-
q_6	ϕ	ϕ	ϕ	ϕ

Instantaneous description for w = abb.

$q_0abbB \vdash aq_0bbB \vdash abq_0bB \vdash abbq_0B \vdash abbBq_1 \vdash abxBq_3 \vdash abxbBq_4$

$\vdash abxbBq_1 \vdash axxbBq_3 \vdash axxbBq_3 \vdash axxbBq_3 \vdash axxbbBq_4 \vdash axxbbBq_4$

$\vdash axxbbBq_1 \vdash axxbbBq_1 \vdash xxxbbBq_2 \vdash xxxbbBq_2 \vdash xxxbbBq_2$

$\vdash xxxbbBq_2 \vdash xxxbbBq_2 \vdash xxxbbaBq_4 \vdash xxxbbaBq_4 \vdash xxxbbaBq_4$

$\vdash xxxbbaBq_1 \vdash xxxbbaBq_1 \vdash Bxxxbbaq_1 \vdash Bxxxbbaq_5 \dots \vdash Bxxxbbaq_5 \vdash Bxxxbbaq_6$

7) Design a TM which recognizes palindrome over $\Sigma = \{a, b\}$

Solution:

Procedure:

1. If there is no input, reach the final state and Halt.
2. If the input = 'a', then traverse forward to process the last symb

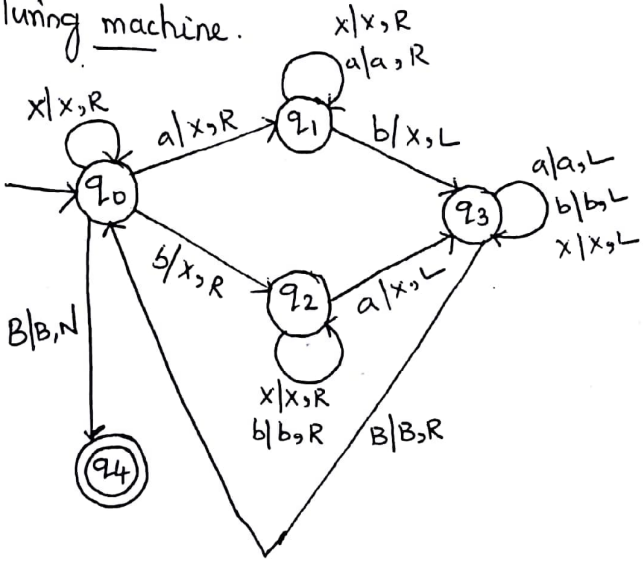
5) Design a Turing Machine to check if there are equal numb over $\{a, b\}$

Solution:

Procedure:

1. Process the first symbol.
2. If it is 'a', move right to locate the first 'b' and replace both by x
3. If 'b' occurs, replace it by x and move forward to reach the first
4. Perform step 2 and 3 until all inputs are processed.

Turing machine.



The turing machin

$M = (Q, \Sigma, \vdash$

Where,

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

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

$\vdash = \{a, b, x, B\}$

$q_0 = \{q_0\}$

$B = \{B\}$

$F = \{q_0\}$

Transition function δ is given by.

$\Sigma \cup \vdash$ / δ	a	b	x	B
$\rightarrow q_0$	(q_1, X, R)	(q_2, X, R)	(q_0, X, R)	(q_4, B, N)
q_1	(q_1, a, R)	(q_3, X, L)	(q_1, X, R)	–
q_2	(q_3, X, L)	(q_2, b, R)	(q_2, X, R)	–
q_3	(q_3, a, L)	(q_3, b, L)	(q_3, X, L)	(q_0, B, R)
$*q_4$	ϕ	ϕ	ϕ	ϕ

Instantaneous des

$\vdash Babba B$ $\vdash BXbba$
$\uparrow q_0$ $\uparrow q_1$

$\vdash BXXbaB$ $\vdash BXXba$
$\uparrow q_3$ $\uparrow q_0$

$\vdash BXXbaB$ $\vdash BXXX$
$\uparrow q_0$ $\uparrow q$

$\vdash BXXXXB$ $\vdash BXXXX$
$\uparrow$ $\uparrow$

uction - Squaring a number

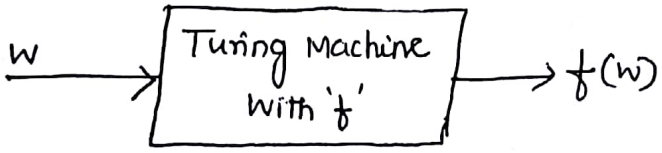
plication - GCD of numbers

ion - finding binary equivalent.

1 2's complementation.

achine M can compute a function, f from the subset of Σ^*.

ne starts the computation corresponding to the input string, w in the 'f' and halts with the output string f(w).



is an input string that cannot be computed by Turing machine, M
n f, then the TM should not accept w.

on:

unction f is Turing computable by the machine,

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

the transition function δ of the form,

$$(q_0, w) \overset{*}{\underset{M}{\vdash}} (q_f, f(w))$$

q_0 → initial state ($q_0 \in Q$)

w → Input string, where $w \in \Sigma$

q_f → Final state ($q_f \in F$)

f(w) → output string after computation.

ical function $w = 0^n$

$f(w) = 0^{f(n)}$

$\vdash$ Bxa$\underset{q_0}{\uparrow}$Yb2CB $\vdash$ BxxY$\underset{q_1}{\uparrow}$b2CB $\vdash$ BxxYb$\underset{q_1}{\uparrow}$2CB $\vdash$ BxxYY$\underset{q_2}{\uparrow}$2CB $\vdash$ BxxYY2$\underset{q_3}{\uparrow}$CB $\vdash$ BxxYY2C$\underset{q_3}{\uparrow}$B

$\vdash$ BxxYY2$\underset{q_3}{\uparrow}$2B $\vdash$ BxxYY2$\underset{q_0}{\uparrow}$2B $\vdash$ BxxYY2$\underset{q_4}{\uparrow}$2B $\vdash$ BxxYY2$\underset{q_4}{\uparrow}$2B $\vdash$ BxxYY2$\underset{q_5}{\uparrow}$2B

(Accept)

Problem:

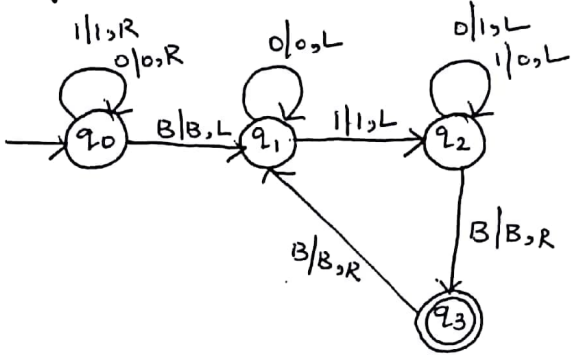
1) Design a TM to perform 2's complement of a number over $\Sigma = \{0,1\}$

Solution:

Procedure:

1. Traverse right and locate the rightmost bit.
2. If the bit = '0', perform no replacement and move left
3. If the bit = '1', perform no change and move left.
4. If the next bit symbol = '0', replace it by '1' and move left.
5. Else if the next bit = '1', replace it by '0' and move left.
6. perform step 4 and 5 until all the i/p symbols are processed.
7. Halt the machine.

Turing Machine:



Turing machine, M is given by,

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

Where,

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

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

$\Gamma = \{0, 1, B\}$

$q_0 = \{q_0\}$

$B = \{B\}$

$F = \{q_3\}$

Transition table, δ

Q \ Σ∪Γ	0	1	B
→ q_0	$(q_0, 0, R)$	$(q_0, 1, R)$	(q_1, B, L)
q_1	$(q_1, 0, L)$	$(q_2, 1, L)$	(q_3, B, R)
q_2	$(q_2, 1, L)$	$(q_2, 0, L)$	(q_3, B, R)
* q_3	ϕ	ϕ	ϕ

2's complement of w = 10100

10100

1's ⇒ 01011

+ 1

2's ⇒ 01100

Instantaneous description for w = 10100

$\vdash$ B 10100 B $\vdash$ B 10100 B $\vdash$ B 1010 0B

(q0) (q0) (q0)

$\vdash$ B 1010 0 B $\vdash$ B 1010 0B $\vdash$ B 101 00 B

(q0) (q0) (q0)

$\vdash$ B 10100 B $\vdash$ B 10100B $\vdash$ B 10100B

(q1) (q1) (q1)

$\vdash$ B 10100B $\vdash$ B 10100B $\vdash$ B 01100B

(q2) (q2) (q2)

$\vdash$ B 01100 B

(q3) (Halt).

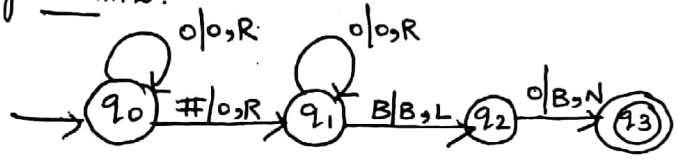
2) Design a TM to compute addition of two unary numbers.

Solution:

Procedure:

1. Read the symbols of the first input with no replacement and move right.
2. When the symbol = '#' replace it by '0' and move right.
3. Traverse right side until the rightmost '0'
4. Replace the rightmost '0' by 'B'.
5. Stop the machine.

Turing machine:



Turing machine, M is given by

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

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

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

$\Gamma = \{0, \#, B\}$

$q_0 = \{q_0\}$, $B = \{B\}$, $F = \{q_3\}$

Transition table δ,

Q \ Σ∪Γ	0	#	B
→q_0	$(q_0, 0, R)$	$(q_0, 0, R)$	–
q_1	$(q_1, 0, R)$	–	(q_2, B, L)
q_2	(q_3, B, N)	–	–
* q_3	ϕ	ϕ	ϕ

Instantaneous description W= 2,3, 2+3 =5

$\vdash B\,00\#000B \vdash B\,00\#000B \vdash B\,00\#000B \vdash B\,00\#000B$

$\vdash B\,00\#000B \vdash B\,000000B \vdash B\,000000B$

$\vdash B\,000000B \vdash B\,000000B \vdash B\,00000BB$

$[2+3 = 00+000 \Rightarrow 00000]$ (Halt)

3) Design a TM to compute subtraction of two unary numbers given by,

$$f(m,n) = \begin{cases} m-n, & \text{if } m>n \\ 0, & \text{otherwise} \end{cases}$$

Solution:

Procedure:

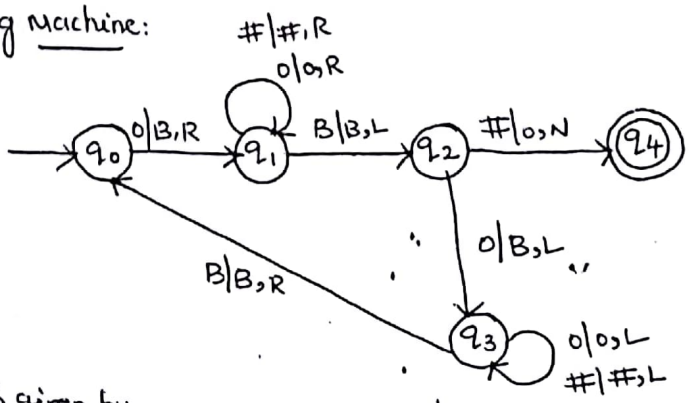
1. Replace the leftmost '0' by B and move right
2. Replace the rightmost '0' by B and move left
3. perform steps 1 and 2, (n-1) times
4. Halts the machine since remaining '0' by B and halt.

Solution:

Procedure:

1. Replace the leftmost '0' by B and move right.
2. Replace the rightmost '0' by B and move left.
3. Perform steps 1 and 2, (n-1) times.
4. Halt the machine since remaining '0' by B and halt.

Turing machine:



Turing machine, M is given by,

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

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

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

$\Gamma = \{0, \#, B\}$

$q_0 = \{q_0\}$

$B = \{B\}$

$F = \{q_4\}$

δ is given by,

$\Sigma \cup \Gamma$ / Q	0	#	B
$\rightarrow q_0$	(q_1, B, R)	-	-
q_1	$(q_1, 0, R)$	$(q_1, \#, R)$	(q_2, B, L)
q_2	(q_3, B, L)	$(q_4, 0, N)$	-
q_3	$(q_3, 0, L)$	$(q_3, \#, L)$	(q_0, B, R)
$*q_4$	ϕ	ϕ	ϕ

Instantaneous discription W = 000#0 [3-1]

$\vdash$ B000#0B $\vdash$ BB0#0B $\vdash$ BB00#0B

$\vdash$ BB00#0B $\vdash$ BB00#0B $\vdash$ BB00#0B

$\vdash$ BB00#0B $\vdash$ BB00#BB $\vdash$ BB00#BB

$\vdash$ BB00#BB $\vdash$ BB00#BB $\vdash$ BBB0#BB $\vdash$ BBB0#BB

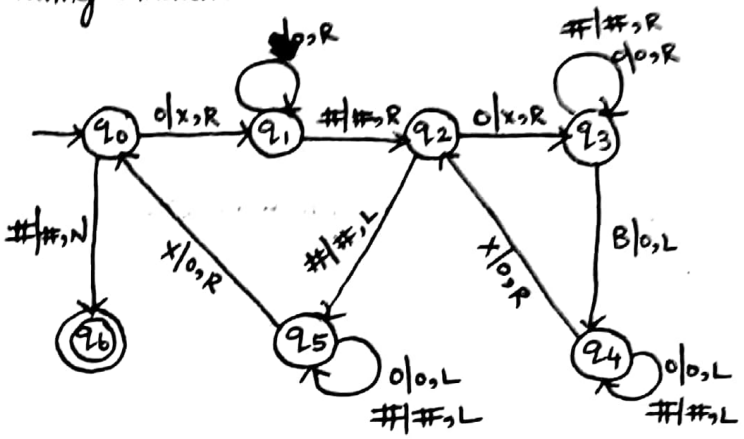
$\vdash$ BBB0#BB $\vdash$ BBB00BB Accept.

4) Compute a TM that performs multiplication of two unary numbers.

Solution: procedure:

1. Read the leftmost '0' replace it by 'x' and move right to process the immediate symbol after '#'
2. Replace the symbol '0' by x and move right reach the first 'B' after '#'.
3. Replace 'B' by '0' and move left until the nearest 'x' is reached.
4. Replace 'x' by '0' and move right to process the next symbol of the multiplicand.
5. Perform Step 2, 3 and 4 until all the symbols of the multiplicand are processed.
6. Move left to replace the symbol of the multiplier 'x' by '0'
7. Perform steps 1 to 6 until all the symbols of the multiplier are processed.

Turing Machine:



Turing machine M is given by

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

where,

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

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

$\Gamma = \{0, \#, x, B\}$

$q_0 = \{q_0\}$

$B = \{B\}$

$F = \{q_6\}$

Transition function δ is given by,

	0	#	X	B
q_0	(q_1, X, R)	$(q_6, \#, N)$	-	-
q_1	$(q_1, 0, R)$	$(q_2, \#, R)$	-	-
q_2	(q_3, X, R)	$(q_5, \#, L)$	-	-
q_3	$(q_3, 0, R)$	$(q_3, \#, R)$	-	$(q_4, 0, L)$
q_4	$(q_4, 0, L)$	$(q_4, \#, L)$	$(q_2, 0, R)$	-
q_5	$(q_5, 0, L)$	$(q_5, \#, L)$	$(q_0, 0, R)$	-
q_6	ϕ	ϕ	ϕ	ϕ

Example: Instantaneous description of $w = 2 \times 2 = 4 \Rightarrow B00\#00\#BB$.

$\vdash$ B00#00#B $\vdash$ Bx0#00#B $\vdash$ Bx0#00#B $\vdash$ Bx0#00#B

(q_0, q_1, q_1, q_2)

$\vdash$ Bx0#x0#B $\vdash$ Bx0#x0#B $\vdash$ Bx0#x0#B $\vdash$ Bx0#x0#0B

(q_3, q_3, q_3, q_4)

$\vdash$ Bx0#x0#0B $\vdash$ Bx0#x0#0B $\vdash$ Bx0#x0#0B $\vdash$ Bx0#0x#0B

(q_4, q_4, q_2, q_3)

$\vdash$ Bx0#0x#0B $\vdash$ Bx0#x0#0B $\vdash$ Bx0#x0#00B $\vdash$ Bx0#0x#00B

(q_3, q_3, q_4, q_4)

$\vdots$

$\vdash$ B00#00#0000B

(q_6)

Result of 2x2.

⇒ In such case, the Turing machine has to be designed that handles subroutines.

⇒ The subroutine has two states (i) Initial State (ii) Return state.

⇒ When the main function is executed, the subroutine is called. the TM reaches the initial state and follows a series of execution using the transition rules of the subroutine.

⇒ After processing, the TM reaches the return state that returns back to the main function after a temporary halt.

Checking off symbols:

⇒ It is a useful technique for visualization how a TM recognizes languages defined by repeated string such as.

$$L = \{ww \mid w \text{ in } \Sigma^*\}$$

⇒ Here we use one track of the tape to mark the some symbols have been read without changing them.

Ex: W=abb

1st track	a	a	b	#	a	a	b	#	← Input symbol
2nd track	✓								← Status

↑ checking off symbol.

Definition: Multitape turning machine.

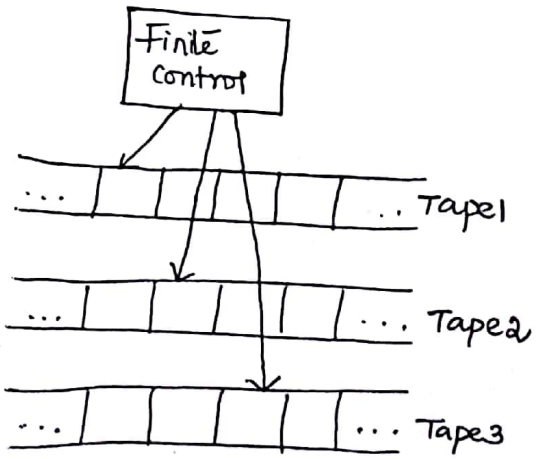
A multitape turning has a finite control with some finite number of ta Each tape is infinite in both directions. It has its own initial state and some accepting states.

Initially,

⇒ Finite set of input symbols is placed on the first tape.

⇒ All the other cells of all the tapes hold the blank.

⇒ The control head of 1st tape is at the left end of the input.



In one more, The multitape TM can

- change state
- print a new symbol on each of the cells scanned by its tape heads.
- Move each of its tape heads, independently, one cell to the left or right or keep it stationary.

Definition : Non-deterministic TM.

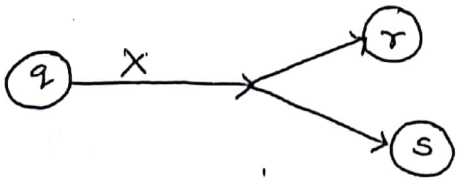
A non-deterministic turing machine is a device with a finite control and a single one-way infinite tape. For a given state and a tape symbol scanned by the tape head, the machine has a finite number of choices for the next move

For $\delta(q, x)$, there is a set of triples like.

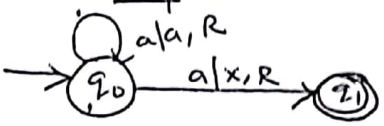
$$\{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \ldots (q_k, Y_k, D_k)\}$$

where k is an integer.

M accepts an input 'w' if there is any sequence of choices of move that leads from the initial ID with w as input, to an ID with an accepting state.



Example :



Problems : (Subroutine)

Design a TM to implement the function "multiplication". and TM will start with $0^m|0^n$ on its tape and will end with 0^{mn} on the tape.

Solution : [Similar problem of multiplication with copy Subroutine]

Problem (Multiple track)

Design the TM to check whether the given input is prime or not using multiple tracks.

Solution:

Procedure

1) The input greater than 2 is placed on the 1st track and also the same input is placed on the 3rd track.
2) The TM write a number 2 on the 2nd track.
3) The number on the 2nd track is subtracted from the 3rd track as many as possible till getting the remainder.
4) If the remainder is zero and the number on the 1st track is not a prime
5) If the remainder is non-zero then increase the number on the 2nd track by 1.
6) If the 2nd track equal to the 1st track then the number is prime, because it should be divided by itself.

Example: consider the number 8.

1st	8	8	8	8	8	...
2nd	2	2	2	2	2	
3rd	8	6	4	2	0	

$\Rightarrow$ The number is not prime.

Consider the number = 7.

Level1:

1st	7	7	7	7	
2nd	2	2	2	2	
3rd	7	5	3	1	

Level2:

1st	7	7	7
2nd	3	3	3
3rd	7	4	1

Level3:

1st	7	7
2nd	4	4
3rd	7	3

Level4:

1st	7	7
2nd	5	5
3rd	7	2

Level5:

1st	7	7	
2nd	6	6	
3rd	7	1	

Level6:

7
7
7

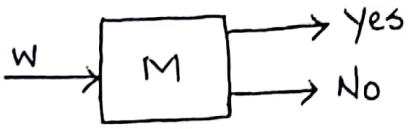
The given number 7 is prime.

UNIT-V COMPUTATIONAL COMPLEXITY

Undecidability:

Basic definitions:

Recursive Language => 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.



Decidable problems => A problem whose language is recursive is said to be decidable otherwise the problem which can be answered as "yes" are called solvable (or) decidable.

Example: sorting, searching etc..

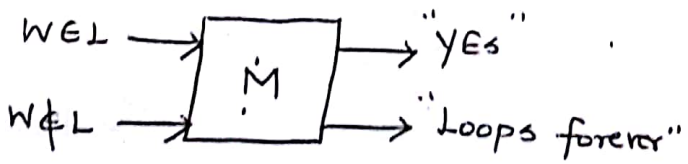
Undecidable problems => The problem which can be answered as "No" are called undecidable problems.

A problem is said to be undecidable if there is no algorithm that takes as input, an instance of the problem and determines whether the output to that instance is 'yes' or 'No'.

Example: post correspondence problems, Halting problem, etc,..

Recursively enumerable language =>

A language $L \subseteq \Sigma^*$ is recursively enumerable if there exists a TM, 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 answer "yes".
If the string is not an element of L, then M may not halt and enter into infinite loop.



⇒ A language is RE if there exists a TM that accepts every string of the language and does not accept string that are not in the language.

⇒ The long range goal of proving the undecidability consisting of pairs such that,

(i) M is a Turing machine encoded in binary
(ii) W is a string of 0's and 1's.
(iii) M accepts input W.

⇒ If the problem with the binary inputs is undecidable, then TM with any other alphabets is undecidable.

⇒ The inputs given to the Turing machine should be in a well-formed representation using binary values.

Codes for Turing Machine

To represent a TM $M = (Q, \{0,1\}, \vdash, \delta, q_1, B, F)$ as a binary string, first assign integer to states, type symbols and directions L and R.

(i) Assume the states are $q_1, q_2, \ldots q_k$ for some k. Using the integers available in the suffix of each state, the string can be represented as $q_1 \rightarrow 0$, $q_2 \rightarrow 00$, $q_3 \rightarrow 000$ etc...

(ii) Assume the tape symbols 0, 1, B are represented as, $X_1, X_2, X_3 \ldots X_m$

Where $X_1 \rightarrow 0$
$X_2 \rightarrow 1$
$X_3 \rightarrow B$

(iii) Assume the directions are represented as D_1 and D_2, where

$L \rightarrow D_1 \rightarrow 0$
$R \rightarrow D_2 \rightarrow 00$

After representing each state, symbol and direction using integers, we can encode the transition function.

$$\delta(q_i, X_j) = (q_k, X_l, D_m) \text{ for some integers } i, j, k, l, m.$$

The encoded string is given by $0^i 1 0^j 1 0^k 1 0^l 1 0^m$.

The code for entire TM consists of all strings, separated by pair of 1's.

$$C_1 11 C_2 11 C_3 11 \cdots C_{n-1} 11 C_n$$

C → Transition Code.

Problem: obtain the code for $\langle M, 1011 \rangle$, where $M = (\{q_1, q_2, q_3\}, \{0,1\}, \{0,1,B\}, \delta, q_1, B, \{q_2\})$

has moves $\delta(q_1,1) = (q_3,0,R)$, $\delta(q_3,0) = (q_1,1,R)$

$\delta(q_3,1) = (q_2,0,R)$, $\delta(q_3,B) = (q_3,1,L)$

Solution:

Assume the tape symbols be encoded as,

$0 - X_1 = 0$

$1 - X_2 = 00$

$B - X_3 = 000$

Directions can be encoded as, $L - D_1 = 0$, $R - D_2 = 00$

Transition are encoded as,

(i) $\delta(q_1,1) = (q_3,0,R)$ $\quad C_1 \Rightarrow 0^1 1 0^2 1 0^3 1 0^1 1 0^2$

(ii) $\delta(q_3,0) = (q_1,1,R)$ $\quad C_2 \Rightarrow 0^3 1 0^1 1 0^1 1 0^1 1 0^2$

(iii) $\delta(q_3,1) = (q_2,0,R)$ $\quad C_3 \Rightarrow 0^3 1 0^2 1 0^2 1 0^1 1 0^2$

(iv) $\delta(q_3,B) = (q_3,1,L)$ $\quad C_4 \Rightarrow 0^3 1 0^3 1 0^3 1 0^2 1 0^1$

The complete code is given by,

$C_1 11 C_2 11 C_3 11 C_4$

$M =$ 0100100010100 11 000101010100 11 000100100101 0011

0001000100010010

$\therefore$ Code for $\langle M, 1011 \rangle$ is given by

$= \langle$ 0100100010100 11 000101010100 11 000100100101 0011

000 1000 1000 10010, 1011 $\rangle$.

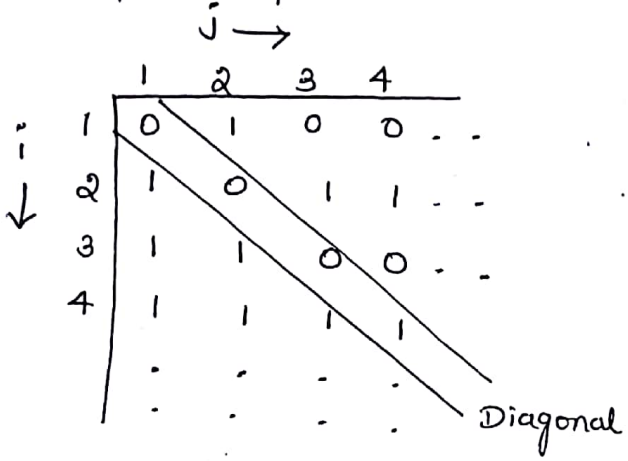
① Diagonalization Language (L_d).

It consist of all the strings w such that the TM represented by w does not accept the input w. L_d has no turing machine that accepts it.

⇒ Some integers do not belong to any TM. If W_i is not a valid TM code, then take the TM M_i to be the TM with one state and no transitions.

Definition ⇒ the diagonalization language L_d is the set of strings W_i, where W_i is not in $L(M_i)$.

L_d consists of all strings w such that the TM, M whose code is w does not accept the input w.



if (i,j) = 1, yes it is accepted

if (i,j) = 0, No it is rejected.

ith row ⇒ characteristic vector for the language $L(M_i)$

Where is in this row correspond to the member of this language.

Inorder to find L_d, complement the diagonal.

Here diagonal value = 0001, After complementing value becomes 1110

Diagonalization ⇒ process of complementing the diagonal to construct the characteristic vector of a language that cannot be language that appears in any row.

Property: L_d is not recursively Enumerable.

proof: Suppose L_d is accepted by some TM M defined by $L(M)$.

Since L_d is a language over alphabet {0,1}, there may be atleast one code for M, say i, (ie) $M = M_i$.

Check whether W_i is in L_d,

By definition $L_d = \{ W_i \mid M_i \text{ does not accept } W_i \}$

Two possibilities:

(i) $W_i \in L_d$ ⇒ (i,j) entry is '0' and M_i does not accept W_i. But our assumption here it is TM M_i which accepts W_i. there is a contradiction.

(ii) $W_i \notin L_d$ ⇒ (i,j) entry is '1' and M_i accept W_i. But by definition of L_d, M_i does not accept W_i. So there is a contradiction.

Hence proved.

Universal language (L_u)

Consider the binary string representation $\langle M, w \rangle$ such that M accepts w called as Universal language L_u.

$$L_u = \{ \langle M, w \rangle \mid M \text{ accepts } w \}$$

L_u is the set of strings representing a TM and an input accepted by that TM. So there is a TM U called as Universal Turing machine.

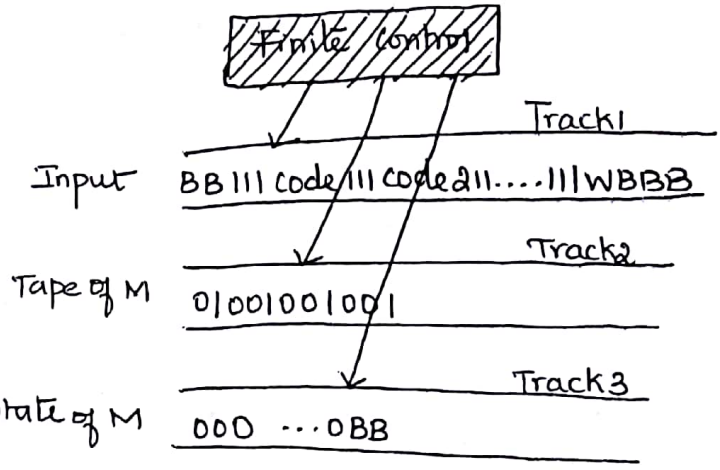
Theorem: L_u is recursively enumerable.

Proof: Inorder to prove this theorem, it is necessary to construct a TM U that accepts L_u. The turing machine U consists of a three track input tape.

1st track ⇒ Hold the input tape ($\langle M, w \rangle$)

2nd track ⇒ Tape symbols are written in unary form.

3rd track ⇒ represents states (unary)



Operation:

1. First make sure that the code for M, is a legitimate code for some TM M. Otherwise it halts without accepting.
2. Initialize the 2nd tape with input w, inf its encoded form. Keep 0 the start state of M on the third tape and move the head of U's second tape to the first simulated cell.
3. If 0^i is the current state with 0^j the current i/p 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 L_u is recursively enumerable.

∴

Rice Theorem :

All nontrivial properties of the RE languages are undecidable.
(ie) It is not possible to recognize the property by a TM.

A property is trivial if it is either empty which is satified by no language or is a all RE language, or else it is nontrivial.

Theorem: Every non-trivial property of the RE language is undecidable.

Proof:

Let $\mathcal{P}$ be a non-trivial property of the RE language.

Assume that ϕ is not in $\mathcal{P}$. Since $\mathcal{P}$ is nontrivial, There must be some non empty language L that is in $\mathcal{P}$.

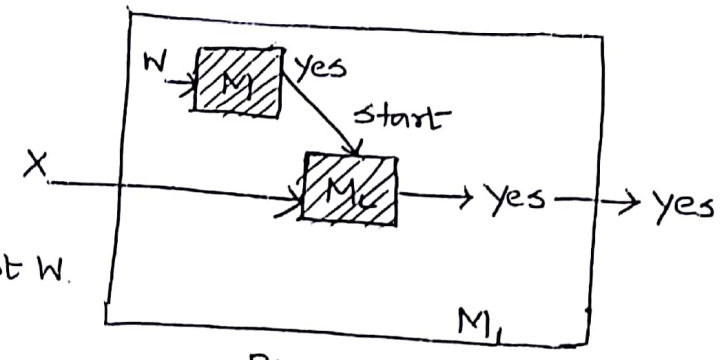
Here M_1 => TM accepting L

L_u => Undecidable.

Design of M_1 is

(i) if $L(M_1) \neq \phi$, if M does not accept W.

(ii) if $L(M_1) = L$, if M accepts W.



Rice theorem.

M_1 is a two-tape TM. one tape is used to simulate M on W and another tape is used to simulate M_L on input X to M_1.

TM, M_1 is constructed as follows,

=> M_1 ignores its input and simulates M on w, If M does not accept w, then M_1 does not accept X. If M accepts W, Then M_1 simulates M_L on x, accepting x if and only if M_L accepts x, thus M_1 either accepts ϕ or L depending on M.

=> It is possible to use the hypothetical $M_{\mathcal{P}}$ to determine if $L(M_1)$ is in $\mathcal{P}$. Since $L(M_1)$ is in $\mathcal{P}$ if and only if $\langle M, w \rangle$ is in L_u. we have an algorithm for L_u, a contradiction.

Thus $\mathcal{P}$ must be undecidable.

Undecidable problems about Turing Machine

Reduction:

P_1 reduces to P_2 => If there is an algorithm used to convert instance of a problem P_1 to instance of a problem P_2 with the same answer, then it is said that P_1 reduces P_2.

Thus, P_1 is not recursive => P_2 cannot be recursive

P_2 is non-RE => P_1 cannot be RE.

So, any instance of P_1 that has "yes" answer can be turned into an instance of P_2 with a "yes" answer and every instance of P_1 with answer "No" answer can be turned into an instance of P_2 with "No" answer.

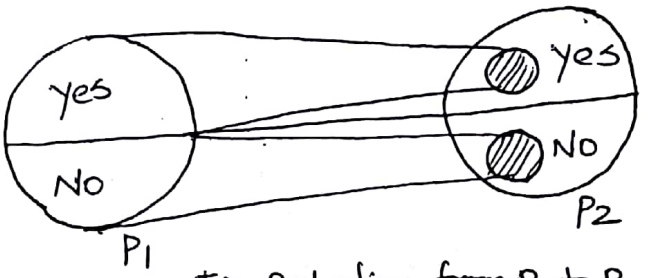


Fig: Reduction from P_1 to P_2.

A reduction can be defined as "A reduction from P_1 to P_2 is TM that accepts/takes an instance of P_1 written on its tape and halts with an instance of P_2 on its tape."

Theorem: If there is a reduction from P_1 to P_2 then.

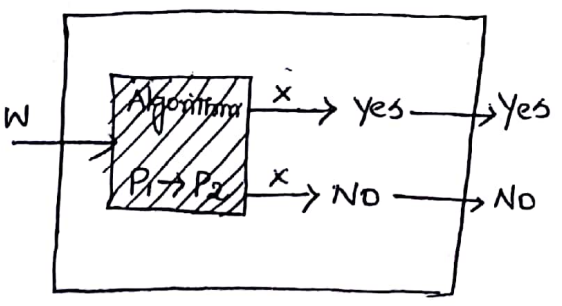
(i) If P_1 is undecidable, then so is P_2.

(ii) If P_1 is non RE, then so is P_2.

Proof:

(i) If P_1 is undecidable, then so is P_2.

Assume P_1 is undecidable. Let A be the algorithm which converts the instance of P_1 to instance of P_2.



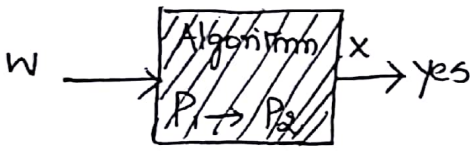
suppose W be the instance of P_1, given to the algorithm A that converts W into an instance X of P_2.

o/p	Inference
yes	X is in P_2
No	X is not in P_2

∴ 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 is RE, have an algorithm to reduce P_1 to P_2 that is there is a TM gives "yes" output if it is input is in P_2 but may not halt if it is input is not in P_2.



If W is in P_1, then X is in P_2, so the TM will accept W.

If W is in P_1, then X is not in P_2, so the TM may or may not halt and will ~~not~~ accept W.

This is a contradiction, thus if P_1 is non RE, then P_2 is non RE.

Turing machine that Accepts the Empty Language.

There are two languages involving TM called L_e and L_{ne},

L_e – Empty language

L_{ne} – Non-empty language.

If $L(M_i) = \phi$, M_i does not accept any input, then W is in L_e.

If $L(M_i) \neq \phi$, language is called L_{ne}.

$$\therefore \boxed{L_e = \{M \mid L(M) = \phi\}}$$

$$\boxed{L_{ne} = \{M \mid L(M) \neq \phi\}}$$

Post- Correspondence problem

An instance of post - Correspondence problem (pcp) consists of two lists of strings over Σ.

$A_B = W_1, W_2, \ldots W_k$

$B = X_1, X_2, \ldots X_k$ 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

$$\boxed{W_{i1}, W_{i2}, W_{i3}, \ldots W_{im} = X_1, X_2, \ldots X_k}$$

is a solution to this instance of pcp.

Pbm: let $\Sigma = \{0, 1\}$. let A and B be strings. Find the instance of PCP.

i	List A: w_i	List B: x_i
1	1	111
2	10111	10
3	10	0

Solution:

Given $W = (1, 10111, 10) = (w_1, w_2, w_3)$

$X = (111, 10, 0) = (x_1, x_2, x_3)$

Take $M = 4$, Take this combination 2, 1, 1, 3

By concatenating string in this series.

$w_2 w_1 w_1 w_3 = x_2 x_1 x_1 x_3$

$10111110 = 10111110$

∴ Instance of PCP = 2, 1, 1, 3.

Pbm: Does PCP with two lists $x = (b, bab^3, ba)$ and $y = (b^3, ba, a)$ have a solution?

Solution: Given $x = (b, bab^3, ba) = (x_1, x_2, x_3)$

$y = (b^3, ba, a) = (y_1, y_2, y_3)$

Take $M = 4$, Take the combination 2, 1, 1, 3

Check the string in this series.

$x_2 x_1 x_1 x_3 = y_2 y_1 y_1 y_3$

$bab^3 bb ba = bab^3 b^3 a$

$bab^3 b^3 a = bab^3 b^3 a$

List x = List y

∴ Hence PCP has a solution for the given lists.

Modified PCP:

Given lists A and B of k strings each from Σ^*

$$A = w_1, w_2, \ldots w_k$$

$$B = x_1, x_2, \ldots x_k$$

It has the solution $i_1, i_2, \ldots i_r$ such that

$$w_1, w_{i_1}, w_{i_2}, \ldots w_{i_r} = x_1, x_{i_1}, x_{i_2}, \ldots x_{i_r}$$

Difference b/w the MPCP and PCP is that in the MPCP, a solution is required to be stored with the first string on each list.

Ex: Let $\Sigma = \{0, 1\}$. let A and B the list of string defined as.

	List A	List B
i	w_i	x_i
1	1	10
2	110	0
3	0	11

Find the instance of MPCP.

Solution: MPCP is given by, $w_1, w_{i_1}, w_{i_2}, \ldots w_{i_r} = x_1, x_{i_1}, x_{i_2}, \ldots x_{i_r}$

$$10110 = 10110$$

Where the instance is given 1, 3, 2.

Problems: Construction of PCP from MPCP.

Consider the TM M and w = 01, where $M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_3\})$ and δ is given by.

	$\delta(q_i, 0)$	$\delta(q_i, 1)$	$\delta(q_i, B)$
q_1	$(q_2, 1, R)$	$(q_2, 0, L)$	$(q_2, 1, L)$
q_2	$(q_3, 0, L)$	$(q_1, 0, R)$	$(q_2, 0, R)$
q_3	-	-	-

Reduce the above problem to PCP and find the solution:

Solution: An instance of MPCP with two lists can be A and B

List A: #

List B: $\#q_101\#$

Rule	List A	List B	Source
1	#	$\#q_{1}01\#$	
2	0	0	
	1	1	
	#	#	
3.	$q_{1}0$	$1q_{2}$	from $\delta(q_1,0) = (q_2,1,R)$
	$0q_{1}1$	$q_{2}00$	from $\delta(q_1,1) = (q_2,0,L)$
	$1q_{1}1$	$q_{2}10$	
	$0q_{1}\#$	$q_{2}01\#$	from $\delta(q_1,B) = (q_2,1,L)$
	$1q_{1}\#$	$q_{2}11\#$	
	$0q_{2}0$	$q_{3}00\#$	from $\delta(q_2,0) = (q_3,0,L)$
	$1q_{2}0$	$q_{3}10\#$	
	$q_{2}1$	$0q_{1}$	from $\delta(q_2,1) = (q_1,0,R)$
	$q_{2}\#$	$0q_{2}\#$	from $\delta(q_2,B) = (q_2,0,R)$
4	$0q_{3}0$	q_{3}	
	$0q_{3}1$	q_{3}	
	$1q_{3}0$	q_{3}	
	$1q_{3}1$	q_{3}	
	$0q_{3}$	q_{3}	
	$1q_{3}$	q_{3}	
	$q_{3}0$	q_{3}	
	$q_{3}1$	q_{3}	
5.	$q_{3}\#\#$	#	

For the input w = 01, Computation is

$\vdash q_{1}01 \vdash 1q_{2}1 \vdash 10q_{1}B \vdash 1q_{2}01 \vdash q_{3}101 \vdash q_{3}01 \vdash q_{3}1 \vdash q_{3}$.

Since q_3 is final state, a solution to the instance of MPCP is written from list A and list B as follows,

$\#q_{1}01\#1q_{2}1\#10q_{1}\#1q_{2}01\#q_{3}101\#q_{3}01\#q_{3}1\#q_{3}\#\#$.

Properties of Recursive and Recursively enumerable languages

Property 1: The union of two recursively enumerable language is recursively enumerable.

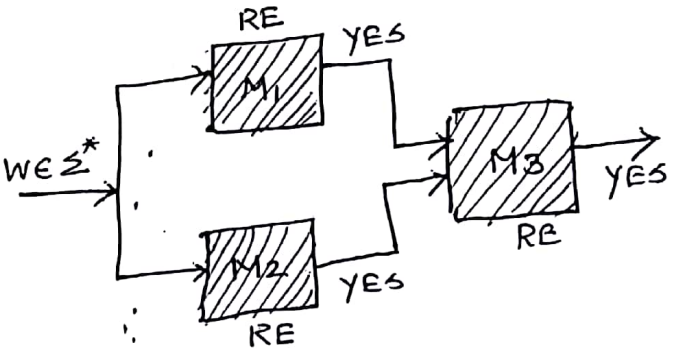
Proof: Let L_1 and L_2 be two recursively enumerable language accepted by the Turing machine M_1 and M_2.

⇒ If a string $w \in L_1$ then M_1 return "YES", accepting the input string, else loops forever. Similarly if a string $w \in L_2$ then M_2 return "YES", else loop forever.

⇒ Here, the output of M_1 and M_2 are written on the input tape of M_3.

⇒ The TM, M_3 returns "YES", if at least one of the outputs of M_1 and M_2 is "YES".

⇒ M_3 decides on $L_1 \cup L_2$ that halts with the answer "YES" if $w \in L_1$ or $w \in L_2$.



⇒ Else M_3 loops forever for both M_1 and M_2 loop forever.

Hence, the Union of two recursively enumerable languages is also recursively enumerable.

Property 2: Intersection of two recursively enumerable language is recursively enumerable.

Proof: Let L_1 and L_2 be two recursively enumerable language accepted by the TM M_1 and M_2.

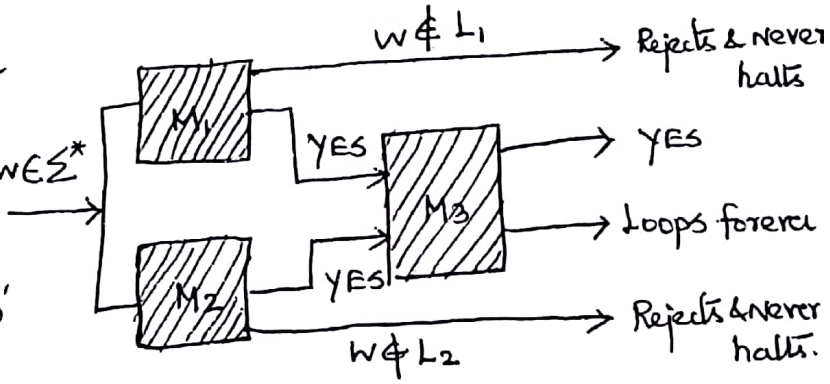
⇒ If a string $w \in L_1$ and then M_1 returns "YES" accepting the input. Else it will not halt after rejecting $w \notin L_1$.

⇒ Similarly if a string, $w \in L_2$ the M_2 returns "YES" else reject it and loop forever.

⇒ Here o/p of M_1 and M_2 are written the i/p tape M_3.

⇒ M_3 returns "YES" if both the outputs of M_1 and M_2 is "YES".

⇒ If at least one of M_1 or 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.

Property3 · 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 language accepted by TM's M_1 and M_2.

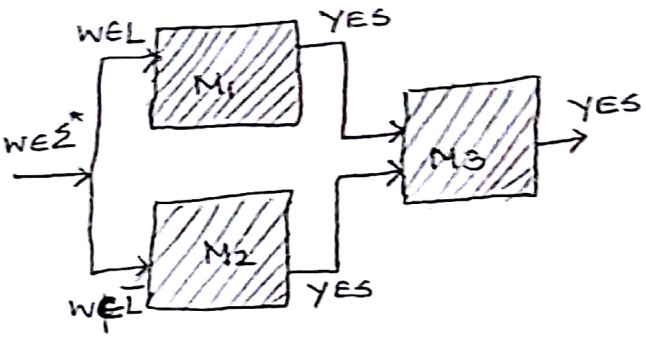
⇒ If a string $w \in L$, it is accepted by M_1 and M_2 halts with answer "YES". Else M_1 enters into infinite loop.

⇒ 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.

⇒ From the diagram, if $w \in L$, then M_1 accepts w and halts with "YES".

⇒ If $w \notin L$, Then M_2 accepts w 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, $w \in \Sigma^*$



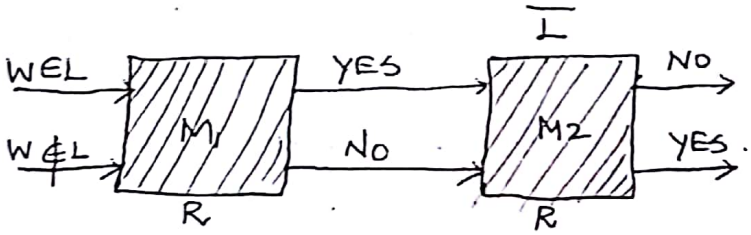
Hence M_3 is a Turing machine that halts for all strings.

Property4: The complement of a recursive language is recursive.

Proof:

Let L be a recursive language accepted by TM, M_1.

Let $\bar{L}$ be a recursive language accepted by TM, M_2.



Let $w \in L$, Then M_1 accepts w and halts with "YES".

M_1 rejects w if $w \notin L$ and 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 $w \in L$ or $w \notin L$ halts with either "YES" or "NO".

Property 5: The union of two recursive language is recursive.

Proof:

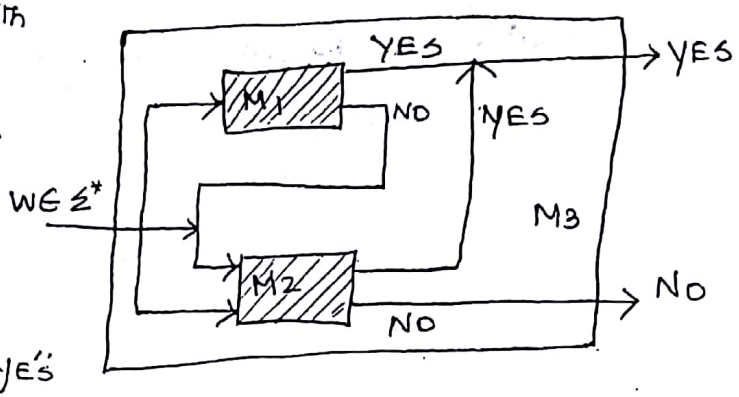
Let L_1 and L_2 be two recursive languages that are accepted by the TM's M_1 and M_2 given by $L(M_1) = L_1$, $L(M_2) = L_2$

⇒ The TM, M_3 first simulates M_1 with the input string w.

⇒ If $w \in L$, then M_1 accepts and thus M_3 also accepts.

Since $L(M_3) = L(M_1) \cup L(M_2)$.

⇒ 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".



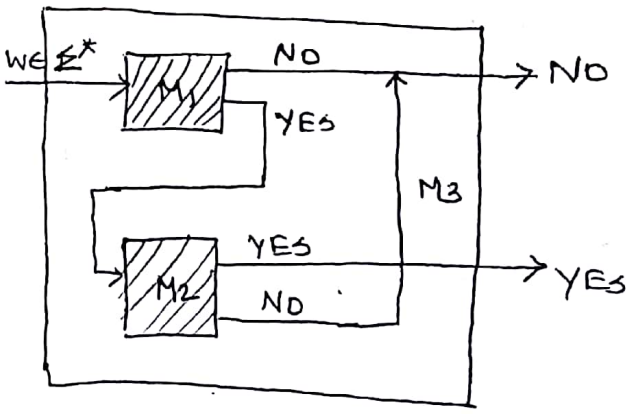
Here M_3, M_2, M_1 that halt with either YES or NO on all possible inputs.

Property 6: 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 $L(M_1) = L_1$, $L(M_2) = L_2$.

⇒ The turning machine M_3 simulates M_1 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 halt with answer "No".

Thus, The intersection of two recursive language is recursive.

Introduction to Computational Complexity

Definition:

Complexity => It defines whether the given problem is easy to solve or hard to solve.

Complexity of problems is classified into two types.

1. **Space Complexity** => Amount of memory space required by the algorithm/problem to complete its computation.
2. **Time Complexity** => Amount of time required to run the program of the problem.

Time and Space complexity of Turing Machine

Time Complexity (T_T)

=> The number of transitions/mores taken by the TM to compute the input is called the time complexity of the TM.

=> Let T be a Turing machine, then the time complexity is a function $T_T(n)$ is the maximum no. of transitions for computing the input, n using T.

T_T depends on the input size of the problem.

When 'n' is lager, T_T will be large; when 'n' is small T_T will be less.

Space Complexity (S_T)

=> Space complexity is the number of tape-cells required to process the turing machine, T for computing the input n.

=> $S_T(n)$ is the maximum number of tape-cells used by T to process an input of length 'n'.

$S_T(n)$ depends on the number of inputs stored on T.

If n is large, then more cells are required to store and process.

If n is small, then limited number of cells on the tape are required by the TM to store and process.

Complexity types / Efficiency types

(i) Best case ⇒ The minimum number of time/space required by the TM to compute the input.

(ii) Average case ⇒ the average number of time/space required by the TM to compute the input.

(iii) Worst case ⇒ The maximum number of time/space required by the TM to process the input.

Complexity of Non-deterministic TM.

Time Complexity ⇒ $T_T(n)$ is a function that refers to the maximum time taken by the NTM to compute the input X, where $|x|=n$.

Space Complexity ⇒ $S_T(n)$ is the maximum amount of tape-cells required by the TM to process X with $|x|=n$.

The time and space complexity is undefined if the NTM loop forever.

Complexity classes: class P and class NP.

Class P:

⇒ P refers to the class of problems that can be solved in polynomial time.

⇒ polynomial time algorithms are efficient ones that can be solved more rapidly.

⇒ these are also referred as tractable problems.

⇒ Example: Searching an element in an ordered list, sorting elements in a list, finding minimum spanning tree of a graph.

Class NP:

⇒ NP refers to the class of problems that are solved by non-deterministic Polynomial time.

⇒ These types of NP problems are known as intractable problems.

⇒ Example: Towers of Hanoi, Travelling salesman problem, Graph coloring problem, Hamiltonian circuit problem, satisfiability problem, etc..

NP - Complete problem:

⇒ A problem is said to be NP-complete if it belongs to NP class problem and can be solved in polynomial time.

⇒ They are also called polynomial time reducible problems.

⇒ A NP-complete problem can be transformed into any other in polynomial time.

NP - Hard problem:

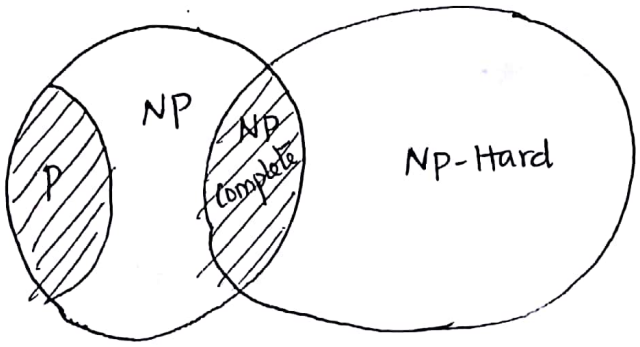
⇒ A problem is said to be NP hard if there exists an algorithm for solving it and it can be translated into one for solving another NP-problem.

⇒ A problem P_1 is NP-hard if

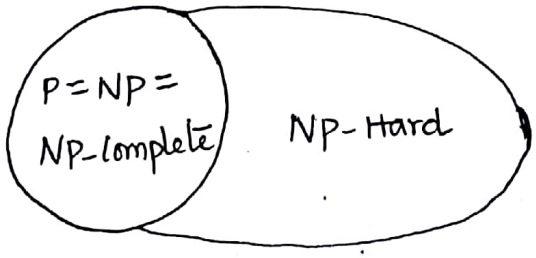
- The problem is an NP class problems.
- For any other problem P_2 in NP, There is a polynomial reduction of L_2 to L_1

⇒ Every NP complete problem must be NP-hard problem.

Conclusion:



$P \neq NP$



$P = NP$

banner motd $ NO ACCESS $
exit
no access message of the day.

Router enable
Router # con
Router # conf
Router # configure ter
Router # configure terminal

London > enable
London # conf t
London (config) # enable sec
London (config) # enable secret class

Router (config) # hostname London.

London (config) # line con
London (config) # line console 0
(config-line) # password cisco
login
exit

line vty 0 4
London (config) # line
Virtual terminal lines - password

London (config-line) # password cisco
login
exit

exist
London # show ru
show running config

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

a, b, c
3
$2^3 = 8$
a
b
c
a, b
a, c
b, c
a, b, c
ϕ

2^n $n \geq 1$
$0^1 = 0$
0^2

a
b
a, b
c
a, c
a, b, c
ab, c
ϕ
b, d
a, b, d
c, d
a, c, d
b, c, d
a, b, c, d

0 / 0000 /
000000000

Scanned by CamScanner

RE

1) The set of strings over alphabet {a,b,c} containing at least one a and atleast one b.
2) The set of strings of 0's and 1's whose tenth symbol from the right end is 1
3) The set of strings of 0's and 1's with at most one pair of consecutive 1's.
4) The set of all strings of 0's and 1's such that every pair of adjacent 0's appears before any pair of adjacent 1's.
5) The set of strings of 0's and 1's whose number of 0's is divisible by 5.

4th floor – Wall – Crack, classroom, floor, ceil