Theory of Computation
(USCS301)

S. Y. B. Sc. (Computer Science) Semester III
(Mumbai University)

Credit Based Semester and Grading System with effect from the academic year 2017-2018

Prof. Manisha Abhyankar
B.Sc Computer Science Co-ordinator,
and Asst. Professor,
Karmaveer Bhaurao Patil College, Vashi,
Mumbai
Maharashtra, India.

Prof. Preeti Sarode
B.Sc (Chemistry),
MCM, M.Phil, MCA
Lecturer in Computer Department
K.M. Agrawal College of Arts, Commerce
& Science, Kalyan (West)
Maharashtra, India.

Tech-Max Publications, Pune
Innovation Throughout
Computer Science Division

BSC9A

Syllabus

Course : USCS301	TOPICS(Credits : 2 Lectures/Week : 3) Theory of Computation	

Objectives :

To provide the comprehensive insight into theory of computation by understanding grammar, languages and other elements of modern language design. Also to develop capabilities to design and develop formulations for computing models and identify its applications in diverse areas.

Expected Learning Outcomes :

1. Understand Grammar and Languages.
2. Learn about Automata theory and its application in Language Design.
3. Learn about Turing Machines and Pushdown Automata.
4. Understand Linear Bound Automata and its applications.

Unit	Details	Lectures
I	**Automata Theory :** Defining Automaton, Finite Automaton, Transitions and Its properties, Acceptability by Finite Automaton, Nondeterministic Finite State Machines, DFA and NDFA equivalence, Mealy and Moore Machines, Minimizing Automata. **Formal Languages :** Defining Grammar, Derivations, Languages generated by Grammar, Chomsky Classification of Grammar and Languages, Recursive Enumerable Sets, Operations on Languages, Languages and Automata. **(Refer Chapters 1 and 2)**	15
II	**Regular Sets and Regular Grammar :** Regular Grammar, Regular Expressions, Finite automata and Regular Expressions, Pumping Lemma and its Applications, Closure Properties, Regular Sets and Regular Grammar **Context Free Languages :** Context-free Languages, Derivation Tree, Ambiguity of Grammar, CFG simplification, Normal Forms, Pumping Lemma for CFG **Pushdown Automata:** Definitions, Acceptance by PDA, PDA and CFG.s **(Refer Chapters 3, 4 and 5)**	15
III	**Linear Bound Automata :** The Linear Bound Automata Model, Linear Bound Automata and Languages. **Turing Machines :** Turing Machine Definition, Representations, Acceptability by Turing Machines, Designing and Description of Turing Machines, Turing Machine Construction, Variants of Turing Machine. **Undecidability :** The Church-Turing thesis, Universal Turing Machine, Halting Problem, Introduction to Unsolvable Problems. **(Refer Chapters 5 and 6)**	15

□□□

Unit I

CHAPTER 1

Automata Theory

University Prescribed Syllabus

Automata Theory : Defining Automaton, Finite Automaton, Transitions and Its properties, Acceptability by Finite Automaton, Nondeterministic Finite State Machines, DFA and NDFA equivalence, Mealy and Moore Machines, Minimizing Automata

1.1 Some Mathematical Terms

Before we start theory of Computation, review some maths terms.

1.1.1 Sets

Informally, a set is a (finite or infinite) collection of distinct elements; the order of elements does not matter.

Finite examples

{0,1,2} = Set of three numbers.

{1,0,2} = The **same** set of three numbers.

{0,0,2} = Not a set (since it has a duplicate).

ϕ = The empty set.

Infinite examples

{0, 1, 2, 3,...} = Set of natural numbers

{i : i > 0 and i is even} = Set of positive even numbers

$x \in S$ = x is an element of S, e.g. $1 \in \{0,1,2\}$

$S_1 \subseteq S_2$ = S_1 is a subset of S_2 ; that is, every element of S_1 is in S_2.

e.g. $\{0, 1\} \subseteq \{0, 1, 2\}$

$\{i : i > 0$ and i is even$\} \subseteq \{0, 1, 2, 3, ...\}$

WRONG : $1 \subseteq \{0,1,2\}$

RIGHT : $1 \subseteq \{0,1,2\}$ or $\{1\} \subseteq \{0,1,2\}$

(Since 1 is not a set).

A. Set operations

1. Union
2. Intersection
3. Difference
4. Complementation

1. Union

$S_1 \cup S_2$ = All elements from S_1 and S_2.

e.g. $\{0\} \cup \{1, 2\} = \{0, 1, 2\}$

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

$\{i : i > 0$ and i is even$\} \cup \{i : i > 0$ and i is odd$\} = \{1,2,3,.....\}$

2. Intersection

$S_1 \cap S_2$ = Elements that are in both sets.

e.g. $\{0,1,2\} \cap \{1,2,3\} = \{1,2\}$

$\{i : i > 0\} \cap \{i : i < 3\} = \{1,2\}$

$\{i : i$ is even$\} \cap \{i : i$ is odd$\} = \phi$

3. Difference

$S_1 - S_2$ = Elements that are in S_1 and not in S_2.

e.g. $\{0, 1, 2\} - \{0,2\} = \{1\}$

$\{0, 1, 2\} - \{2, 3\} = \{0,1\}$

$\{i : i$ is even$\} - \{i : i$ is odd$\} = \{i : i$ is even$\}$

4. Complementation

- We often consider a fixed **universal set** (also known as universe) of all possible values, denoted U.

  e.g. Set of all integer numbers

  Set of all computer programs

  Set of all possible grades for the course

- For a set $S \subseteq U$, the complement of S includes all elements of U that are not in S.

For examples

1. U is a set of all final grades, from 0 to 100.

   $S = \{x : x \geq 50\}$ – passing grades.

   $\bar{S} = \{x : x < 50\}$ – failing grades.

2. U is a set of integers

   $S = \{i : i$ is even$\}$

   $\bar{S} = \{i : i$ is odd$\}$

(i) Size of a set

$|S|$ = Number of elements

e.g. $|\{0,1,2\}| = 3$

$|\phi| = 0$

$|\{i : i$ is even$\}| = \infty$

(ii) Power set

2^s = Set of all subsets of S

e.g. $2^{\{1,2\}} = \{\phi, \{1\}, \{2\}, \{1, 2\}\}$

$2^{\{1,2,3\}} = \{\phi, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$

$2^{\phi} = \{\phi\}$

If $|S| = n$, then $|2^S| = 2^n$; that is, $|2^S| = 2^{|S|}$

B. Cartesian product

$S_1 \times S_2$ = List of all possible pairs with the first element from S_1 and the second element from S_2.

e.g. $\{1,2\} \times \{a, b\} = \{(1,a), (1,b), (2,a), (2,b)\}$

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

If S_1 is a list of boys, and S_2 is a list of girls, then $S_1 \times S_2$ is the list of possible marriages.

$S_1 \times S_2 \times \ldots \times S_n = \{(x_1, x_2, \ldots x_n) : x_1 \in S_1, x_2 \in S_2, \ldots, x_n \in S_n\}$

e.g. $\{1,2\} \times \{a, b\} \times \{X, Y\}$

$= \{(1,a,X), (1,a,Y), (1,b,X), (1,b,Y))$

$(2,a,X), (2,a,Y), (2,b,X), (2,b,Y)\}$

$|S_1 \times S_2 \times \ldots \times S_n| = |S_1| \cdot |S_2| \ldots |S_n|$

C. Simple laws

$S \cup \phi = S$ $\quad \overline{\phi} = U$

$S - \phi = S$ $\quad$ If $S_1 = \overline{S}_2$, then $S_2 = \overline{S}_1$; $\quad$ that is, $\overline{S} = S$. $S \cap \phi = \phi$

Fig. 1.1.1

D. De Morgan's laws

$\overline{S_1 \cup S_2} = \overline{S}_1 \cap \overline{S}_2$

$\overline{S_1 \cap S_2} = \overline{S}_1 \cup \overline{S}_2$

Fig. 1.1.2

E. Functions and Relations

1. Function : $f : S_1 \rightarrow S_2$

- For each element of S_1, a function determines exactly one element of S_2 (in the textbook, it is called a total function).
- S_1 is the **domain** of f. Set of all elements in S_2 "pointed" by f is the **range** of f as shown in Fig. 1.1.3.

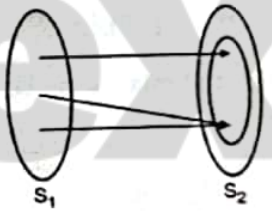


Fig. 1.1.3

Domain : {1,2,3}

Range : {a,b}

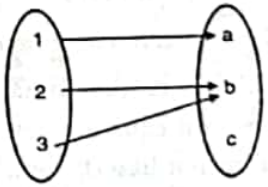


Fig. 1.1.4

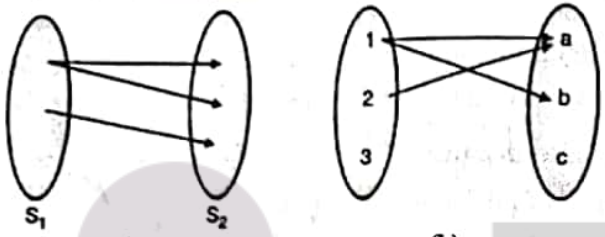
$f(X) = X^2$

Domain : All real numbers

Range : All non-negative real numbers

2. Relation

- For each element of S_1, a relation determines several (possibly none) elements of S_2.



(a) (b)

Fig. 1.1.5

- e.g. Mapping from students to the classes they take–a student can take zero, one, or more classes.

Equivalence

- **A special case of a relation**, usually denoted by "≡", which satisfies three **properties** :

1. $X \equiv X$ (**reflexivity**)
2. if $X \equiv Y$, then $Y \equiv X$ (**symmetry**)
3. If $X \equiv Y$ and $Y \equiv Z$, then $X \equiv Z$ (**transitivity**)

- For example (i) "Living in the same city" is an equivalence. (ii) Liking someone is not an equivalence : If John like Mary and Mary like Donald, John may not like Donald.

More examples

(i) X = Y = Equivalence

X and Y have the same sign = Equivalence,

(ii) X < Y = Not equivalence

1.1.2 Graphs and Trees

1. Graph

- **A graph** is a collection of vertices connected by directed edges (e.g intersections connected by one-way streets).

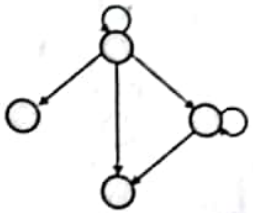


Fig. 1.1.6

- **A path** is a "walk" from one vertex to another along directed edges, where each edge is visited at most once.

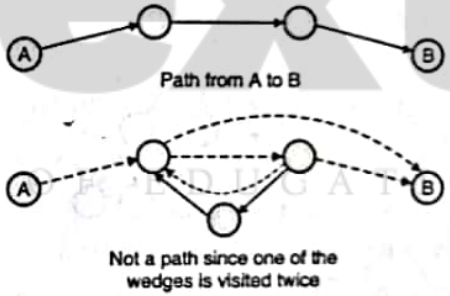


Fig. 1.1.7

- If a path visits each vertex at most once, it is a simple path.

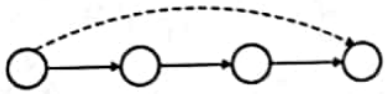


Fig 1.1.8

- A cycle is a path from a vertex to itself.

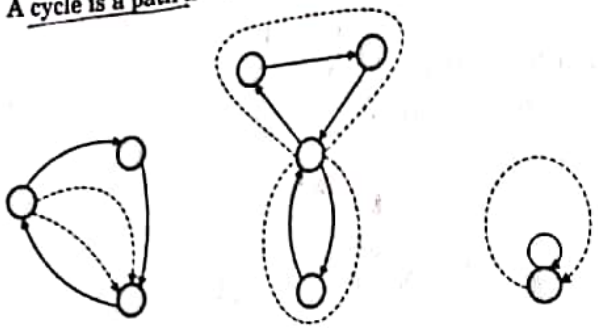


Fig. 1.1.9

2. Trees

- A tree is a special case of a graph. It has a **root** vertex and exactly one path from the root to any other vertex.

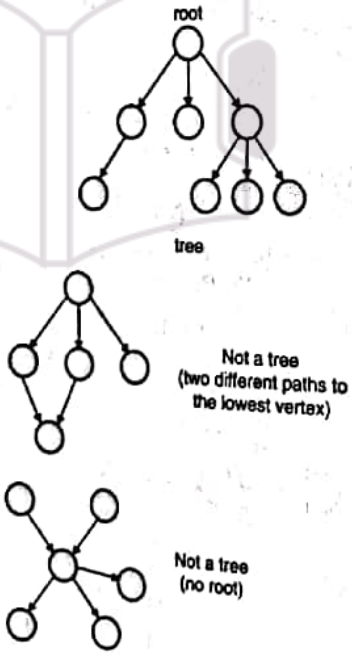


Fig. 1.1.10

- If there is an edge from A to B, then A is a **parent** of B, and B is a **child** of A.

Fig. 1.1.11

- A node without children is a **leaf.**

1.1.3 Proof Techniques

A proof is a sequence of steps that lead from some known facts to the desired conclusion; each step must be **obviously** connect.

A. Proof by contradiction

- To prove some fact P, we show that "not P" is false. That is, we suppose "not P" and demonstrate that it leads to an obviously wrong result.

For example : Prove that $\sqrt{2}$ is not rotational.

- Suppose that it is rotational, that is $\sqrt{2} = \frac{n}{m}$, where n and m do not have a common actor.
- Then, $2 \cdot m^2 = n^2$, which implies that n is even, $n = 2k$. Thus $2 \cdot m^2 = 4 \cdot k^2$, which means that $m^2 = 2 \cdot k^2$; hence, m is even. We conclude that m and n are both even, contradicting the assumption that they do not have a common factor.

B. Proof by induction

- We show that some fact is true for every natural number n using two arguments :

1. **Base :** It is true for n = 1 (or for some other small numbers).
2. **Step :** If it is true for n, then it is also true for (n + 1)

   True for 1 = True for 2 = True for 3 = ...

   Informally, it is like an infinite loop.

For example : Prove that $1 + 2 + ... + n = \frac{n(n+1)}{2}$

Base : $n = 1, i = \frac{1+(1+1)}{2}$ = True

Step : Suppose that it is true for n, that is, $\underbrace{1 + 2 + ...+ n}_{\text{Inductive hypothesis, also known as inductive assumption}} = \frac{n(n+1)}{2}$

$$1 + 2 + 3 + ...+ n + (n+1) = \frac{n(n+1)}{2} + (n+1)$$

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

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

1.2 Basic Concepts

1.2.1 Strings

- A **string** is a finite sequence of symbols; strings are usually denoted by u, v and w.

  e.g. u = abcab is a string on Σ = {a, b, c}
- The **empty string** (no symbols at all) is denoted λ.
- A part of a string is a **substring**.

  e.g. bca is a substring of abcab.
- A beginning of a string (upto any symbol) is a prefix and an ending is a suffix.

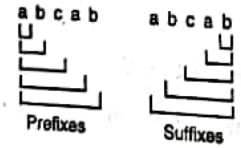


Fig. 1.2.1

Note : A string is a prefix and suffix of itself.

λ is a prefix and suffix of any string.

String operations

1. **Concatenation :** wv = Appearing v to the end of w.

   e.g. w = abc, v = ab

   wv = abc ab

   w v

Note : λw = wλ = w.

2. **Reverse :** w^R = w in reverse order.

   e.g. w = abc, w^R = cba
3. **Length of a string :** |w| = Number of symbols

   e.g. |abc| = 3

Note : |λ| = 0

$|w^R| = |w|$

|wv| = |w| + |v|

w^n = w repeated n times; that is, $w^n = \underbrace{www.....w}_{\text{n times}}$

e.g. $(abc)^3$ = abc abc abc

Note : $w^0 = \lambda$

$|w^n| = n \cdot |w|$

1.2.2 Languages

- Σ = An alphabet is a non-empty finite set of symbols,

  E.g. Σ = {a, b, c} is an alphabet.

- Σ^* is the set of all strings on Σ.

e.g. $\Sigma = \{a, b, c\}$

$\Sigma^* = \{\lambda, a, b, c, aa, ab\ ac, ba, ...\}$

- Σ^+ is the set of all strings except λ.

Note : Σ^* and Σ^+ are infinite.

- A **language** is a subset of Σ^*, usually denoted L.
- It may be finite or infinite.

e.g. : {a, ba, abc}

{λ, a, aa, aaa, ...}

ϕ

- If a string w is in L, we say that w is a **sentence** of L.

Operations on Languages

- Union, intersection and difference are same as on sets.

1. **Intersection :** e.g. {a, ba, abc} $\cap$ {λ, a, aa, aaa, ...} = {a}
2. **Complementation :** $\bar{L} = \Sigma^* - L$

e.g. L = {λ, a, aa, aaa, ...}

L = {w : w includes b or c}

3. **Reverse :** $L^R = \{w^R : w \in L\}$ = The set of reversed strings.

e.g. L = {a, ba, abc}

$\bar{L}$ = {a, ab, cba}

4. **Concatenation :** $L_1 L_2 = \{wv : w \in L_1 \text{ and } v \in L_2\}$

= Similar to the Cartesian product

e.g. L_1 = {a, ba, abc}

L_2 = {bcb, b}

$L_1 L_2$ = {a bcb, a b, ba bcb, ba b, abc bcb, abc a}

$L^n = \underbrace{L\,L\,....\,L}_{n \text{ times}}$ = L concatenated with itself n times

e.g. L = {a, aa}

L^3 = {a, aa} · {a, aa} · {a, aa} = {aaa, aaaa, aaaaa, aaaaaa}.

Note : $L^* = \{\lambda\}$

$L^* = L$

5. **Star-closure** (also known as Kleene star) : $L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup ....$

e.g. L = {a, aa}

L^* = {λ, a, aa, aaa, aaaa, ...}

6. **Positive closure (also known as Kleene plus)**
$L^+ = L^1 \cup L^2 \cup L^3 \cup ....$

e,g. {a, aa}$^+$ = {a, aa, aaa, ...}

{λ, a, aa}$^+$ = {λ, a, aa, aaa, ...}

Note : L^+ includes λ if and only if L includes λ.

1.3 TOC : Theory of Computation

- The theory of computation is a branch of Computer Science that deals with whether and how efficiency problems can be solved on a model of computation, using an algorithm.
- This field is divided into three major branches :
  1. Automata theory
  2. Computability theory
  3. Computational complexity theory
- Automata theory is the study of abstract mathematical machines called automata and the problems that can be solved using these machines.

- An automata is characterized by a number of states it can be in, a number of trensitions between those states and an alphabets of symbols it accepts.

Syllabus Topic : Defining Automaton

1.3.1 Definition of an Automaton

What is Automata ?

- **Automaton** are abstract models of machines that perform computations on an input by moving through a series of states or configurations.
  - At each state of the computation, a transition function determines the next configuration on the basis of a finite portion of the present configuration.
  - As a result, once the computation reaches an accepting configuration, it accepts that input.
- The most general and powerful automata is the **Turing machine.**
- The **major objective** of automata theory is to develop methods by which computer scientists can describe and analyze the dynamic behaviour of discrete systems, in which the signals are sampled periodically.
- The behaviour of these discrete systems is determined by the way that the system is constructed from storage and combinational elements.

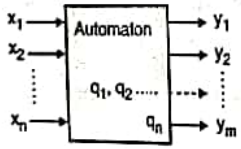


Fig. 1.3.1 : Model of a discrete automaton

1.3.2 Characteristics of an Automata

Write characteristics of an Automata ?

- **Inputs :** It is nothing but sequences of symbols selected from a finite set I of input signals.

For example set I is the set of $(x_1, x_2, x_3, \ldots x_n)$ where n is the number of inputs.

- **Outputs :** It is the sequences of symbols selected from a finite set O.

  For example set O is the set of $(y_1, y_2, y_3, \ldots, y_m)$ where m is the number of outputs.
- **States :** It is finite set Q, whose definition depends on the type of automaton.
- **State relation :** The state relation defines the status of next state which is determined by the current state and the current input at that instance of time.
- **Output relation :** The output depends on either state only or to both input and state.

Control Unit

- The most **important feature** of the automaton is its control unit. It can be in any one of the finite number of interval states at any point.
- It can change state in some defined manner which is determined by a transition function.
- The Fig. 1.3.2 shows a diagrammatic representation of a generic automation.

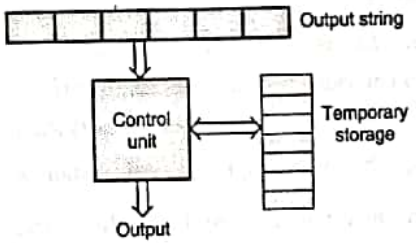


Fig. 1.3.2

- There are **four major families of automaton**
  1. Finite-state machine
  2. Pushdown automata

3. Linear-bounded automata
4. Turing machine

1.3.3 Automaton Types

- **Automaton without a memory :** output depends only on the input
- **automaton with a finite memory :** output depends on the input as well as state.
- **Moore machine:** the output depends only on the states of the machine
- **Mealy machine :** the output depends on the state as well as on the input at any instant of time

1.3.4 Automaton Labels

- The states are represented by circles
- The transitions are represented by arrows.
- Initial state is represented by arrow with Circle

Final state is represented by concentric Circle

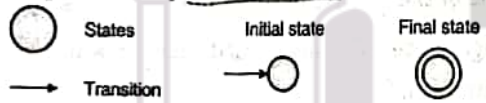


Fig. 1.3.3

- As the automaton sees a symbol of input, it makes a transition (or jump) to another state. This is as per its transition function which takes the current state and the recent symbol as its inputs.

Exam

1.3.5 Uses of Automata : Compiler Design and Parsing

Every automaton fulfils the following basic requirements :

1. Every automaton has a mechanism for reading input.
   - The input is assumed to be a sequence of symbols represented with an alphabet and is placed on an input tape (or written on an input file).

   - The simpler automata can only read the input one symbol at a time from left to right but not change. Complex versions of automata can both read in any direction and can also change the input.
2. The automaton can produce output of some form
   - If the output response to an input string is binary (accept/reject or yes/no, or true/false etc), then it is called an accepter.
   - If the output response is a sequence to an input sequence, then it is called a transducer (or automaton with output).
3. The automaton may have a temporary storage consisting of an unlimited number of cells. Each cell may be capable of holding a symbol from an alphabet (which may be different from the input alphabet).
   - The automaton can read and change the contents of the storage cells in the temporary storage.
   - The accumulating capability of this storage varies depending on the type of the storage.

1.3.6 Operation of the Automaton

Write operation of an Automata.

- At any point of time the automaton is in some integral state and is reading a particular symbol from the input tape by using the mechanism for reading input.
- In the next time step, the automaton then moves to some other integral state (or remains in the same state) as defined by the transition function.
- The transition function is based on the current state, the input symbol that has been read and the contents of the temporary storage.
- It is also possible that at the same time the contents of the storage may be changed and the input that was read may be modified.
- The automation may also produce some output during transition.
- The internal state, input and the content of storage at any point defines the configuration of the automaton at that point.

- The transition from one configuration to the next (as defined by th transition function) is called a move.
- Any system, that is at any point of time in one of the finite number o interval state and moves among these states in a defined manner i response to some input, can be modelled by a **finite automaton**.
- It doesn't have any temporary storage and hence a restricted model o computation.

Syllabus Topic : Transition and It's Properties

1.4 Transition Systems

Transition graph can be interpreted as a flowchart for an algorithm recognizing a language. A transition graph consists of following things :

1. A transition graph or a transition system is a finite directed labele graph in which each vertex (or node) represents a state and the directed edges indicate the transition of a state and the edges are labelled with input/output.
2. In a finite set of states, at least one state should be the start state and some of which are designated as final states.
   - Initial state is represented by arrow with Circle
   - Final state is represented by concentric Circle
5. An alphabet Σ of possible input symbols from which the input strings are formed. Example : The edges are labeled by input / output (e.g. by 1/0 or 1/1 or a/b).
6. A finite set of transitions that show the change of state from the given state on a given input.

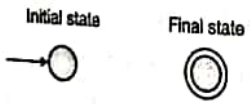


Fig. 1.4.1

- A transition system or transition graph is a finite directed labelled graph in which each vertex represents a state and the directed edges indicate of a state and the edges are labelled with input/output.
- Fig. 1.4.2 shows a transition system and initial and final states.

q_0 is Initial state

q_1 is Final State.

0/1 are inputs and outputs

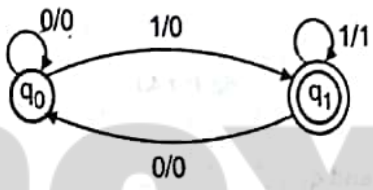


Fig. 1.4.2

1.4.1 Definition of a Transition System

Define transition system in brief.

1. A transition system is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$.
   - Q = the finite nonempty set of states
   - Σ = the input alphabet
   - F = the set of final states,.
   - $q_0 \in Q$ and q_0 is nonempty
   - δ is a finite subset of $Q \times \Sigma^* \times Q$.

   Imp

   In other words, if (q_1, w, q_2) is in δ, it means that the graph starts at the vertex q_1, goes along a set of edges, and reaches the vertex q_2. The concatenation of the label of all the edges thus encountered is w.
2. A transition system accepts a string w in Σ^* if
   - There exists a path which originates from some initial state, goes along the arrows, and terminates at some final state.

- The path value obtained by concatenation of all edge-labels of the path is equal to w.

Example 1.4.1 : Consider the transition system given in Fig. P. 1. 3.1. Determine the initial states, the final states and the acceptability of 101011,111010.

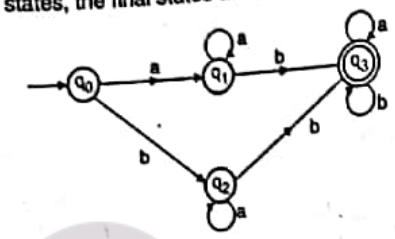


Fig. P. 1.4.1

Solution :

1. Initial states : q_0 and q_1
2. Final state : q_3
3. For string : 101011

   The path-value is $q_0 - q_0 - q_2 - q_3$

   As q_3 is the final state, 101011 is accepted by the transition system.

For string 111010 is not accepted by the transition system as there is no path with path value 111010 because here transition system having two initial states.

1.4.2 Transition Table

- A transition table is a tabular representation of the transition function.
- In table, the column contains the state in which the automaton will be on the input represented by that column. The row corresponds to the state, the finite control unit can be in.
- The entry for one row corresponding to state q and the column corresponds to input a is the state δ (q, a).
- For the transition graph in the Fig. 1.4.3 the transition table would be as shown in Table 1.4.1.

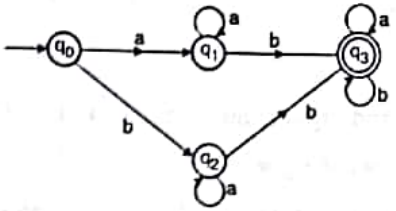


Fig. 1.4.3 : Transition diagram

Table 1.4.1

States	a	b
q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_2	q_3
*q_3	q_3	q_3

Following are the steps to draw a state diagram or transition graph from the transition table.

1. Draw the circles to represent the states given.
2. For each of the states, scan across the corresponding row and draw an arrow to the given state in transition table..

   There can be multiple arrows for an input character if the automaton is an NFA.
3. Designate a state as the start state. The start state is given in the formal definition of the automaton.
4. Designate one or more states as accept state. This is also given in the formal definition.

1.4.3 Properties of Transition Functions

Write transition properties in brief.

Property 1

- δ (q, a) = q is a finite automaton. This means that the state of the system can be changed only by an input symbol.

- It requires valid input.

Property 2

For all strings w and input symbols a, $\delta(q, aw) = \delta(\delta(q, a), w)$.

$$\delta(q, wa) = \delta(\delta(q, w) a)$$

- This property gives the state after the automaton consumes or reads the first symbol of a string aw and the state after the automaton consumes a prefix of the string wa.

Syllabus Topic : Finite Automaton

1.5 Finite Automata

- **Finite state machine** or **finite automation** is the simplest type of **abstract machine.**
- Automata (singular : automation) are simple but useful model of computation.

1.5.1 States, Transitions and Finite – State Transition System

☞ **Write components of finite automata.**

- **A state of a system** is an instantaneous description of that system which gives all relevant information necessary to determine how the system can evolve from that point onwards.
- **Transitions** are changes of states that can occur spontaneously or in response to inputs to the states.
- Some examples of state transition systems are : digital systems, vending machines, etc.
- A system containing only a finite number of states and transitions among them is called a **finite –state transition system.**
- Finite-state transition systems can be modelled abstractly by a mathematical model called **Finite automation.**

1.5.2 Finite Automaton Representation

A finite automaton can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$

(i) **Q** is a finite non-empty **set of states.**

(ii) Σ is a finite non-empty set of inputs called the **input alphabet.**

(iii) δ is a function which maps $Q \times \Sigma$ into Q

- δ is called the direct transition function.
- This is the function which describes the change of states during the transition.
- Transition function is usually represented by a transition table or a transition diagram.

(iv) $q_0 \in Q$ **is the initial state.**

(v) $F \subseteq Q$ is the **set of final states.** (there may be more than one final states)

Syllabus Topic : Acceptability by Finite Automaton

1.5.3 Acceptability of a String by a Finite Automaton

- Consider $a_1, a_2, a_3, \ldots\ldots, a_n$ is a sequence of input symbols.
- $q_0, q_1, q_2, \ldots\ldots, q_n$ are set of states where, q_0 is start state and q_n is final state.
- The transition function is processed as,

$$\delta(q_0, a_1) = q_1$$
$$\delta(q_1, a_2) = q_2$$
$$\delta(q_2, a_3) = q_3$$
$$\ldots\ldots\ldots$$
$$\ldots\ldots\ldots$$
$$\delta(q_{n-1}, a_n) = q_n$$

- Input $a_1, a_2, a_3, \ldots\ldots, a_n$ is said to be 'accepted' since q_n is a member of the final state, and if not then it is 'rejected'.

- Language accepted by DFA, 'M' written as,

$$\{ L(M) = \{ w / \delta(q_0, w) = q_n \text{ for some } q_n \text{ in } F \}$$

Here, F means final state.

Final state is also called as **accepting state.**

For example

Consider the finite state machine whose transition function δ is given in Fig. form of a transition table.

Here, $Q = \{ q_0, q_1, q_2, q_3 \}$ and $\Sigma = \{ 0, 1 \}$, $F = \{ q_0 \}$

States \ Inputs	0	1
→ q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Give the entire sequence of states for the input string 110101.

$$\delta(q_0, 110101) = \delta(q_1, 10101)$$
$$\downarrow$$
$$= \delta(q_0, 0101)$$
$$\downarrow$$
$$= \delta(q_2, 101)$$
$$\downarrow$$
$$= \delta(q_3, 01)$$
$$\downarrow$$
$$= \delta(q_1, 1)$$
$$\downarrow$$
$$= \delta(q_0, \epsilon)$$
$$\downarrow$$
$$= q_0$$

Transition diagram

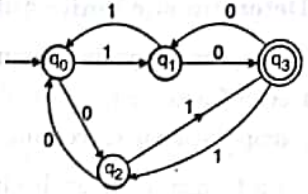


Fig. 1.5.1

The symbol ↓ indicates that the current input symbol is being processed by the machine

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

Syllabus Topic : Non-Deterministic Finite State Machines, DFA and NDFA Equivalence, Minimizing Automata, Mealy and Moore Machines

1.5.4 Finite Automata Classification

Write a classification of finite automata and explain in brief.

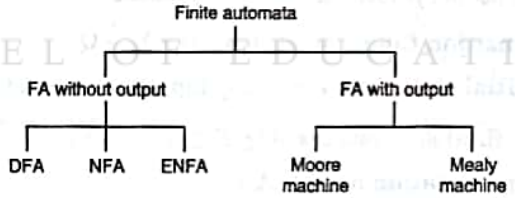


Fig. 1.5.2

Imp Exam

A. Finite Automata without Output

1. Deterministic finite automata (DFA)

DFA is defined as an abstract mathematical concept, but due to the deterministic nature of DFA it is implementable in hardware and software for solving various problems.

- In this concept, each input symbol determines the state to which the machine will move. Hence it is called **Deterministic Automaton.**

- As it has a finite number of states the machine is called **Deterministic Finite Machine** or **Deterministic Finite Automaton.**
- A DFA has a start state (arrow symbol) from where the computations begin. It also has a set of accepted states (denoted by double circles) which define when a computation is successful.
- For each state there is a transition arrow leading out to a next state for each symbol.
- Deterministic means that there is only one outcome. It means move to next state when the symbol matches ($S_0 \rightarrow S_1$) or move back to the same state when the symbol matches ($S_0 \rightarrow S_0$)

Definition of a DFA

Q Define DFA.

A DFA can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where,

- *Q is a finite **set of states.**
- Σ is a finite set of symbols called the **alphabet.**
- δ is the **transition function** where $\delta : Q \times \Sigma \rightarrow Q$
- q_0 is the **initial state** from where any input is processed ($q_0 \in Q$)
- F is a set of **final state/states** of Q ($F \subseteq Q$)

Graphical Representation of a DFA

A DFA is represented by digraphs called **state diagram.**

- The **vertices** represent the states.
- The arcs labelled with an input alphabet show the transitions.
- The initial state is denoted by an empty single incoming arc.
- The final state is indicated by double circles.

Example

Let a deterministic finite automaton be,

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

$$\Sigma = (0, 1)$$

$$q_0 = \{q_0\}$$

$$F = \{q_2\}, \text{ and}$$

Transition function δ as shown by the Table.

Table 1.5.1

States \ Inputs	0	1
q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

Graphical representation would be as shown in Fig. 1.5.3

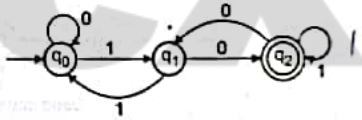


Fig. 1.5.3

Solved examples based on DFA

Example 1.5.1 : Design a DFA for language L_1 ;

L_1 = Set of all strings that state with 0

= {0, 00, 01, 00D , 010, 011, 0000, ...}

Solution :

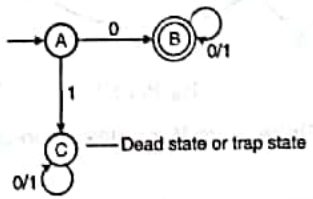


Fig. P. 1.5.1(a)

Dead state or trap state

e.g. 001 is a string

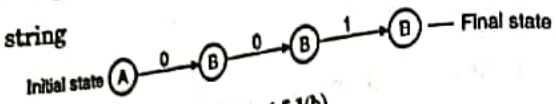


Fig. P. 1.5.1(b)

101 is a string

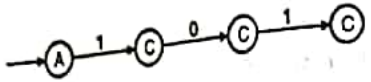


Fig. P. 1.5.1(c)

But C is not a final state, so this string is rejected.

Example 1.5.2 : Construct DFA that accepts sets of all strings over { 0, 1 } of lengths 2.

Solution :

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

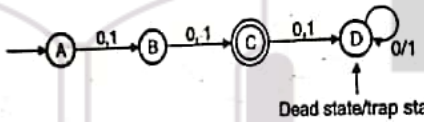


Fig. P. 1.5.2

Example 1.5.3 : Construct a DFA that accepts any string over {a, b} that contain the string aabb in it.

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

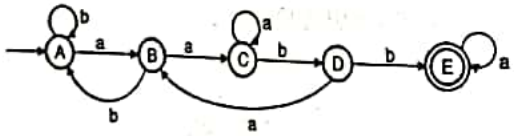


Fig. P. 1.5.3

2. Non-Deterministic Finite State Machines / Non-Deterministic Finite Automaton (NFA)

The Fig. 1.5.4 will explain the concept of non-deterministic finite automaton using a transition diagram.

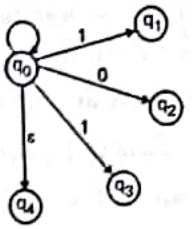


Fig. 1.5.4 : Transition system representing non-deterministic automaton

- In NFA, for the given current state, there could be multiple next states.
- The next state may be chosen at random.
- All the next states may be chosen in parallel.

Definition of Non-deterministic Automata

- For a particular input symbol, the machine can move to any combination of the states in the machine. Final state does not go anywhere.
- In other words, the exact state to which the machine moves cannot be determined, hence it is called **Non-deterministic Automaton.**
- As it has finite number of states the machine is called Non-deterministic Finite Machine or Non-deterministic Finite Automation NDFA.

Example :

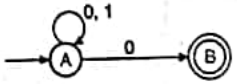


Fig. 1.5.5

Definition of an NDFA

☞ **Define NDFA.**

An NDFA can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where,

- Q is a finite **set of states .**
- Σ is a finite set of symbols called the **alphabets.**
- δ is the **transition function** where $\delta : Q \times \Sigma \rightarrow 2^Q$

- (Here, the power set of Q (2^Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states).
- q_0 is the **initial state** from where any input is processed ($q_0 \in Q$)
- F is a set of **final state/states** of Q ($F \subseteq Q$)

Graphical representation of an NDFA

- The **vertices** represent the **states.**
- The **arcs labelled with an input alphabet** show the **transitions.**
- The **initial state is denoted** by an empty single **incoming arc.**
- The **final state** is indicated by **double circles.**

For example

Let a non-deterministic finite automaton be denoted as follows :

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

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

$q_0 = \{q_0\}$

$F = \{q_2\}$ and

The transition function δ as shown below,

States \ Inputs	0	1
q_0	$\{q_0, q_1\}$	q_1
q_1	q_2	$\{q_0, q_2\}$
q_2	$\{q_1, q_2\}$	q_2

Graphical representation would be as shown in Fig. 1.5.6

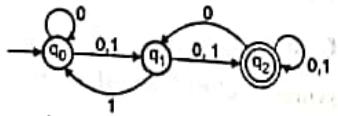


Fig. 1.5.6

NFA Examples :

L_1 = { Set of all strings that end with '1'}

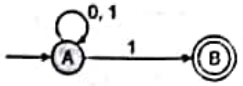


Fig. 1.5.7

L_2 = { Set of all strings that contain '0' }

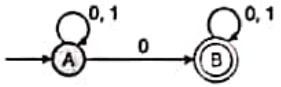


Fig. 1.5.8

L_3 = { Set of all strings that starts with '01' }

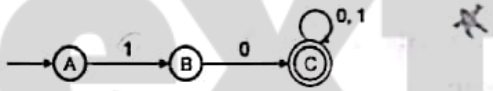


Fig. 1.5.9

L_3 = { Set of all strings that contain '01' }

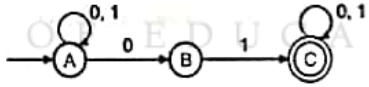


Fig. 1.5.10

L = { Set of all strings that ends with '11' }

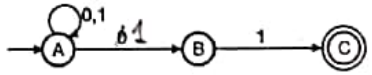


Fig. 1.5.11

Solved examples based on NFA

Example 1.5.4 : Design NFA for language L ;

L = { Set of all strings that start with 0 }

= { 0, 00, 01, 000, 010, 011, ... }

Solution :

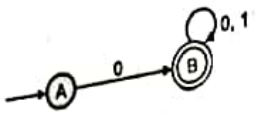


Fig. P. 1.5.4(a)

Initial state = A

Final state = B

Input alphabet = {0, 1}

Check for string 001,

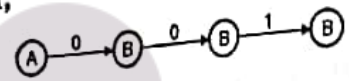


Fig. 1.5.4(b)

Last state = Final state

String is accepted

Check for string 101,

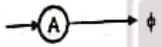


Fig. P. 1.5.4(c) : Dead configuration

String is not accepted

Example 1.5.5 : Construct NFA that accepts set of all strings over {0, 1} of length 2.

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

Solution :

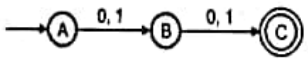


Fig. P. 1.5.5(a)

Check for string 00

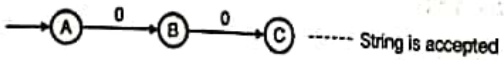


Fig. P. 1.5.5(b)

Check for string 001

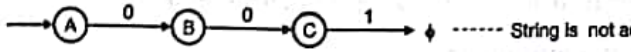


Fig. P. 1.5.5(c)

Imp Exam

Difference between DFA and NDFA

Write the difference between DFA and NDFA.

- The difference between the deterministic and non-deterministic automata is only in δ. $\delta : Q \times \Sigma \to Q$
- For deterministic automaton (DFA), the outcome is a state i.e. an element of Q. $\delta : Q \times \Sigma \to 2^Q$
- For non-deterministic automaton the outcome is a subset of Q.

Table 1.5.1

No.	DFA	NDFA
1.	The transition from a state is to a single particular next state for each input symbol. Hence it is called deterministic.	The transition from a state to multiple next states for each input symbol. Hence it is called non-deterministic.
2.	Empty string transitions are not seen in DFA.	NDFA permits empty string transitions.
3.	Backtracking is allowed in DFA.	Backtracking is not always possible.
4.	Requires more space.	Requires less space.
5.	A string is accepted by a DFA, if it transits to a final state.	A string is accepted by a NDFA, if at least one of all possible transitions ends in a final state.
6.	Empty string transitions are not seen in DFA.	NDFA permits empty string transitions.

Equivalence of two Finite Automata

Write steps to identify equivalence of two finite automata.

Steps to identify equivalence

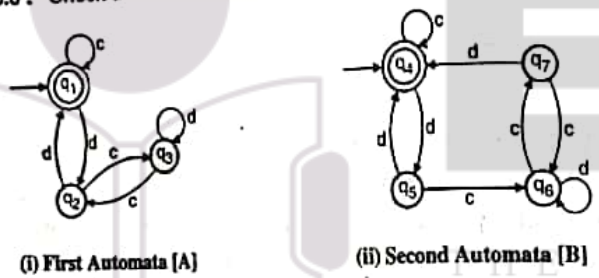
1. For any pair of states $\{q_i, q_j\}$ the transition for input $a \in \Sigma$ is defined $\{q_a, q_b\}$ where ;

   $\delta\{q_i, a\} = q_a$ and $\delta\{q_j, a\} = q_b$

   The two automates are not equivalent if for pair $\{q_a, q_b\}$ one intermediate state and other is final state

2. If initial state is final state of one automation, then in second automa also, initial state must be final state for them to be equivalent.

Example 1.5.6 : Check these two automata are equivalent or not.



(i) First Automata [A]

(ii) Second Automata [B]

Fig. P. 1.5.6

Solution :

Check condition I

- For every pair of states the pair of states generates particular inp showable either both should be on intermediate state or both should on final state.

States	Input	
	C	D
$\{q_1, q_4\}$	$\{q_1, q_4\}$ FS FS	$\{q_2, q_5\}$ IS IS
$\{q_2, q_5\}$	$\{q_3, q_6\}$ IS IS	$\{q_1, q_4\}$ FS FS
$\{q_3, q_6\}$	$\{q_2, q_7\}$ IS IS	$\{q_3, q_5\}$ IS IS
$\{q_2, q_7\}$	$\{q_3, q_6\}$ IS IS	{q1, q4} FS FS

Note FS = final state

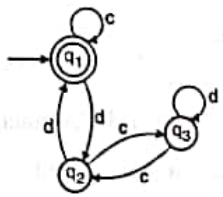
IS = Intermediate state

So first condition is satisfied.

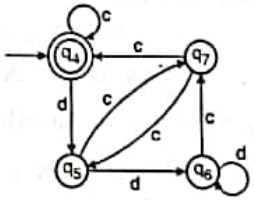
Check condition II

- Initial state and final state are same, in both automata.
- In first automata, q_1 is initial state as well as final state and in second automata q_4 is initial state as well as final state. So these two automata are equivalent.

Example 1.5.7 : Find out whether the following automata are equivalent or not :



[A]



[B]

Fig. P. 1.5.7

Solution

In this example, second condition is satisfied because q_1 is initial and final state in A Automata and q_4 is initial state and final state in B Automata

Check condition I

	Inputs	
State	c	d
(q_1, q_4)	(q_1, q_4) FS FS	(q_2, q_5) IS, IS
(q_2, q_5)	(q_3, q_7) IS, IS	(q_1, q_6) FS, IS

A and B are not equivalent.

DFA and NDFA equivalence

Conversions of NDFA to DFA

Write steps to convert NDFA to DFA.

- Every DFA is an NFA, but not vice versa. But there is an equivalent DF
- for every NFA.
- **Transition function of DFA is,**

$$\delta = Q \times \Sigma \to Q$$

- **Transition function of NFA is,**

$$\delta = Q \times \Sigma \to 2^Q$$

- We can convert every NFA into DFA.

Steps to convert NFA into DFA

1. Construct the transaction table of given NFA machine
2. Scan the next states column in transaction table from initial sta to final state

3. If any of the next state consists more than one state on single input alphabet, then merge them and make a new state .place this new constructed state in DFA transaction table as a present state.
4. The next state of this new constructed state on input alphabet will be the summation of each next state which parts in the NFA transition table.
5. Repeat steps 2 to4 until all states in NFA transition table will be scanned completely.
6. The final transition table must have single next state at single input alphabet.

Example 1.5.8 : Find equivalent DFA for the NFA given by L = [{A, B, C}, (a, b), δ, A, {c}] where δ is given by :

Table P. 1.5.8 : Transition table for NFA

	a	b
A	A, B	C
B	A	B
©C	–	A, B

Solution :

States : A, B, C

Inputs : a, b

Transition function δ : Given in transition table

Initial state : A

Final state : C

State diagram for NFA

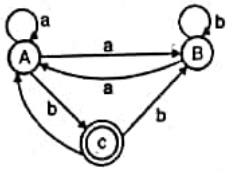


Fig. P. 1.5.8(a)

Convert NFA into DFA

Transition table for DFA

	a	b
→ A	{A, B}	C
AB	AB	BC
(BC)	A	AB
(C)	D	AB
D	0	D

Note

- To find AB states, we must look at transition state diagram of NFA and apply union operation.
- In DFA we cannot leave a state so we have to write new state. In our example we have written 0 is a new state which is called dead state.
- The input that comes in dead state remains there.
- To find **final state of DFA**
  - Check final state of NFA and after that select state from DFA that contains the final state of NFA.
  - In our example, 'C' state D a final state 1 NFA. So select state from DFA which contains 'C' state in DFA. So we get two final state in our example.

- State diagram for DFA

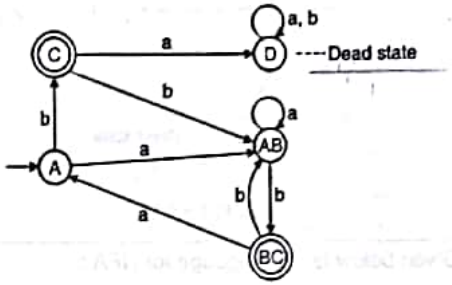


Fig. P. 1.5.8(b)

Example 1.5.9 : L = {Set of all strings over (0, 1) that starts with '0'}

Σ = {0, 1}

Fig. P. 1.5.9(a)

State transition diagram,

States	0	1
A	B	ϕ
B	B	B

Convert the NFA into DFA

Solution :

Transition state

	0	1	
A	B	C	.... C is a dead state or trap state
B	B	B	
C	C	C	

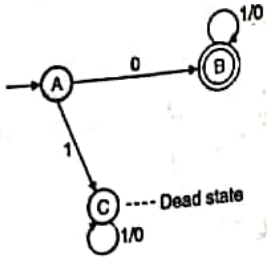


Fig. P. 1.5.9(b)

Example 1.5.10 : Given below is the language for NFA :

L = {Set of all strings over (0, 1) that ends with '01'} construct equivalent DFA.

Solution :

First draw NFA

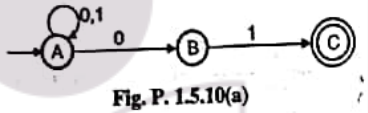


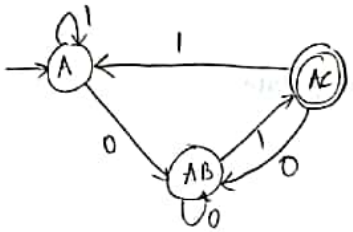
Fig. P. 1.5.10(a)

Transition table

	0	1
→ A	A, B	A
B	ϕ	C
(C)	ϕ	ϕ

Convert NFA into DFA

Transition table for DFA



	0	1
→ A	AB	A
AB	AB	AC
(AC)	AB	A

Fig. P. 1.5.10(b)

Example 1.5.11 : Design an NFA for language that accepts all strings over {0, 1} in which second last symbol is always '1'. Then convert it into its equivalent DFA.
e.g. string 1010 or 110 or 1101010.

Solution :

Design NFA

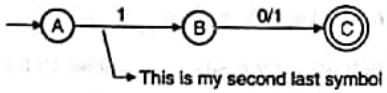


Fig. P. 1.5.11(a)

State transition table for NFA

	0	1
→ A	A	A, B
B	C	C
(C)	ϕ	ϕ

Convert NFA into DFA

Transition table

	0	1
→ A	A	AB
AB	AC	ABC
(AC)	A	AB
(ABC)	AC	ABC

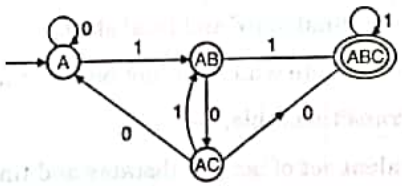


Fig. P. 1.5.11(b)

Minimization of DFA

- How to do minimization of DFA ?
- How to reduce the number of states from DFA ?

– Minimization of DFA means reduce the number of states.

States and equivalence between states

– Consider two states p and q we say that these two states are equivalent.

Condition

$$\delta(p, w) \in f$$

$$\Rightarrow \quad \delta(q, w) \in f$$

Non equivalent :

$\therefore$ If $\delta(p, w) \notin f$

$\Rightarrow \delta(q, w) \notin f$.

If any two states follow this property, then it is called as two states are equivalent.

If length of w = 0, then it is 0 equivalent

$\therefore |w| = 0, \ldots\ldots\ldots 0$ equivalent

$|w| = 1, \ldots\ldots\ldots 1$ equivalent

.

.

.

$|w| = n, \ldots\ldots\ldots n$ equivalent

Steps to find minimization of DFA

1. Identify the non-final state and final state.
2. Try to delete the state which can not be reachable from Initial state.
3. Draw state transition table.
4. Find 0 equivalent set of non-final states and final states.

5. Find 1 equivalence sets :

   Take previous states and check whether they are in same state or not. If they are in same state they are equivalent to each other.
6. Repeat the procedure till we get two same equivalence.
7. Draw a transition diagram or state diagram.

Example 1.5.12 : Construct a minimum DFA equivalent to the DFA

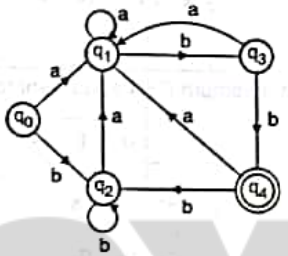


Fig. P. 1.5.12(a)

Solution :

	a	b
q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	q_4
(q_4)	q_1	q_2

		Non final states	Final states
0	Equivalence	$[q_0, q_1, q_2, q_3]$	$[q_4]$
1	Equivalence	$[q_0, q_1, q_2]$ $[q_3]$ $[q_4]$	(q_0, q_1)
2	Equivalence	$[q_0, q_2]$ $[q_1]$ $[q_3]$	$[q_4]$
3	Equivalence	$[q_0, q_2]$ $[q_1]$ $[q_3]$	$[q_4]$

Draw a DFA

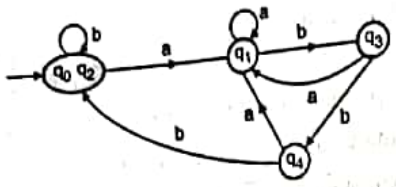


Fig. P. 1.5.12(b)

Minimization of DFA

Example 1.5.13 : Construct a minimum DFA equivalent to the DFA :

	0	1
→ q_0	q_1	q_5
q_1	q_6	q_2
q_2	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

Solution :

Initial state = q_0

Final state = q_2

Input = {0, 1}

0th Equivalence

$\{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}\ \{q_2\}$

1st Equivalence

$\{q_0, q_4, q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\ \{q_2\}$

2nd Equivalence

$\{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}, \{q_2\}$

3rd Equivalence

$\{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}, \{q_2\}$

New transition table

	0	1
→ $\{q_0, q_4\}$	$\{q_1, q_7\}$	$\{q_3, q_5\}$
$\{q_6\}$	$\{q_6\}$	$\{q_0, q_4\}$
$\{q_1, q_7\}$	$\{q_6\}$	$\{q_2\}$
$\{q_3, q_5\}$	$\{q_2\}$	$\{q_6\}$
$(\{q_2\})$	$\{q_0, q_4\}$	$\{q_2\}$

Example 1.5.14 : Construct a minimum DFA equivalent to the DFA,

	0	1
→ q_0	q_1	q_5
q_1	q_6	* q_2
* q_2	q_0	* q_2
q_4	q_7	q_5
q_5	* q_2	q_6
q_6	q_6	q_4
q_7	q_6	* q_2

(* Mark indicates final state)

Solution :

0 equivalence : $[q_0\ q_1\ q_4\ q_5\ q_6\ q_7]\ [q_2]$

1 equivalence : $[q_0\ q_4\ q_6]\ [q_1\ q_7]\ [q_5]\ [q_2]$

2 equivalence : $[q_0\ q_4][q_6][q_1q_7][q_5][q_2]$

3 equivalence : $[q_0\ q_4]\ [q_6][q_1\ q_7]\ [q_5][q_2]$

4 equivalence : $[q_0\ q_4][q_6][q_1\ q_7][q_5][q_2]$

Minimization of DFA : When there are more than one final states involved.

Example 1.5.15 : Construct a minimum DFA equivalent to the DFA,

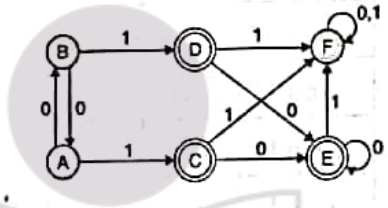


Fig. P. 1.5.15(a)

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

Solution :

0 - Equivalence - {A, B, F} {C, D, E}

1 - Equivalence - {A, B} {F} {C, D, E}

2 - Equivalence - {A, B} {F} {C, D, E}

New transition table

	0	1
→ {A, B}	{A, B}	{C, D, E}
{F}	{F}	{F}
{C, D, E}	{C, D, E}	{F}

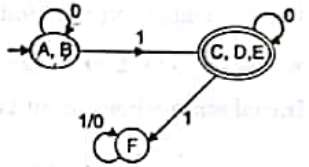


Fig. 1.5.15(b)

B. Finite state Automata with output

There are two types of finite state machine :

(i) Mealy machine
(ii) Moore machine

(i) Mealy machine

☞ **Explain Mealy machine with example.**

The value of the output function in most general case is function of present state and present input.

Lets consider,

y(t) = Output function

q(t) = Present state

x(t) = Input

Then we can say that,

$$y(t) = \delta\ (q(t), x(t))$$

Mealy machine can be described by six tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

where,

Q = Finite set of states

Σ = Input alphabet

Δ = Output alphabet

δ = Input transition function

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

λ = Output transition function

where $\lambda : Q \times \Sigma \to \Delta / Q \to \Delta$

q_0 = Initial state where input is ($q_0 \in Q$)

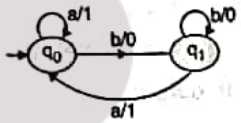


Fig. 1.5.12

$\lambda : Q \times \Sigma \to \Delta$

$(q_0, a) \to 1$

$(q_0, b) \to 0$

$(q_1, b) \to 0$

$(q_1, a) \to 1$

$q_0 \xrightarrow[1]{a} q_0 \xrightarrow[0]{b}$

If we give 'n' bit input then output will be n bit.

Example 1.5.16 : Construct a Mealy machine that takes binary number as input and produces 2's complement of that number as output. Assume the string is read LSB to MSB and end carry is discarded.

Solution :

Σ = {0, 1}

Δ = {0, 1} ... 2's complement of number which is binary

1. First find 1's complement of input

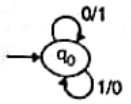


Fig. P. 1.5.16(a)

Check for input 1011 → 1's complement is 0100

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

2. Find 2's complement

∴ MSB LSB

Input → 1 [1] [00]

1's complement ⟶ 0 0 1 1

2's complement (adding 1) ⟶ + 1

0 [1] [0 0]

← LSB

Input → 1 1 1 0 1 [1 0 0]

←

1's complement ⟶ 0 0 0 1 0 0 1 1

\+ 1

2's complement ⟶ 0 0 0 1 0 [1] 0 0

←

Note : Whenever we see first '1' it remains same and after that take 1's complement of each bit.

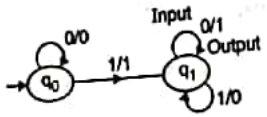


Fig. P. 1.5.16(b)

Check for 1100 → 2's complement for 0100

read from LSB

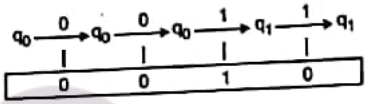


Fig. P. 1.5.16(c)

(ii) Moore machine

Explain Moore machine with example.

Moore machine is an FSM whose outputs depend on only the present state.

A moore machine can be described by a 6 tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

Where -

Q - Finite set of states

Σ - Input alphabet

Δ - Output alphabet

δ - Input transition function where $\delta : Q \times \Sigma \rightarrow \theta$

λ - Output transition function where $\lambda : Q \rightarrow \Delta$

q_0 – Initial state where input is $(q_0 \in \theta)$

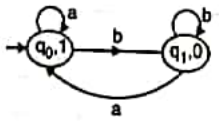


Fig. 1.5.13

In Fig. 1.5.13, a and b are inputs and 1and 0 are outputs, q_0 and q_1 are states.

Output transition function $\lambda : Q \rightarrow \Delta$

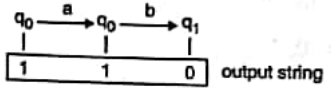


Fig. 1.5.14

If we give 'n' bit input then output will be 'n' bit.

Example 1.5.17 : Construct a Moore machine that takes set of all strings over {a, b} as input and prints '1' as output for every occurrence of 'ab' as a substring.

Solution :

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

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

1. First construct DFA -

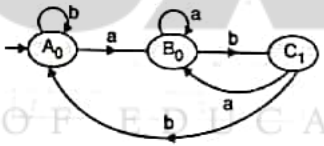


Fig. P. 1.5.17(a)

2. If the input string is 'ab'

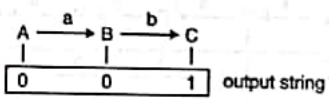


Fig. P. 1.5.17(b)

3. If the input string is 'abab'

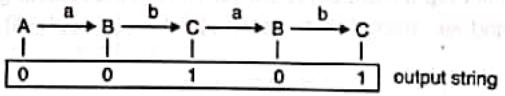


Fig. P. 1.5.17(c)

Example 1.5.18 : Construct a moore machine that takes set of all strings over {a, b} and count number of occurrence of substring 'baa'.

Solution :

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

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

(1) First construct DFA

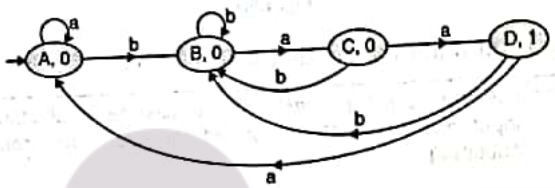


Fig. P. 1.5.18(a)

Note : (1) Search the state where "baa" sub-string is present.

(2) In that state put 1 as a output and remaining outputs are zero.

1. If input is "baa"

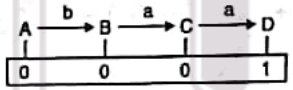


Fig. P. 1.5.18(b)

If input is "baabaa"

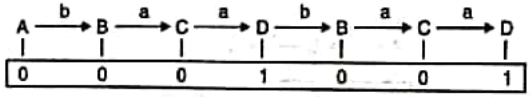


Fig. P. 1.5.18(c)

Example 1.5.19 : For the following moore machine the input alphabet is $\sum$ = {a, b} and the output alphabet is Δ = {0, 1}. Run the following input sequence and find the respective outputs : (i) aabab (ii) abbb (iii) ababb

States	a	b	Outputs
$\rightarrow q_0$	q_1	q_2	0
q_1	q_2	q_3	0
q_2	q_3	q_4	1
q_3	q_4	q_4	0
q_4	q_0	q_0	0

Solution :

(i) If input is "aabab"

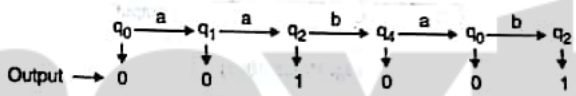


Fig. P. 1.5.19(a)

(ii) If input is "abbb"

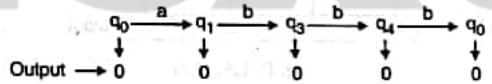


Fig. P. 1.5.19(b)

(iii) If input is "ababb"

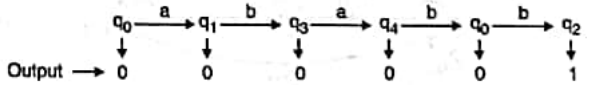


Fig. P. 1.5.19(c)

Example 1.5.20 : Construct a moore machine that takes set of all strings over {0, 1} and produces 'A' as output. If input ends with '01' or produces B as output if input ends with '11' otherwise produce 'C'.

Solution :

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

$$\Delta = \{A, B, C\}$$

If input ends with 01 output $\rightarrow$ A

If input ends with 11 output → B

Otherwise output → C

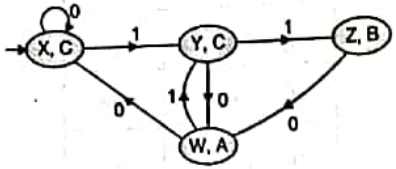


Fig. P. 1.5.20(a)

Check for 111

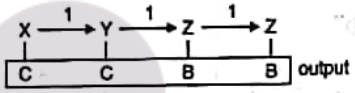


Fig. P. 1.5.20(b)

Check for 110

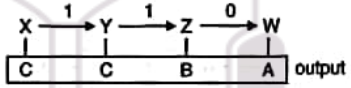


Fig. P. 1.5.20(c)

Example 1.5.21 : What is the output produced by machine shown in Fig. P. 1.5.20(a) ?

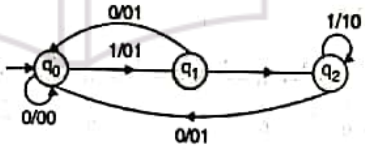


Fig. P. 1.5.21(a)

Solution :

Steps

- Start searching string from LSB.
- Whenever we find '1' in string let it remain as it is.
- From next bit onwards take 1's complement of each bit.

1. State table of Moore machine

Present state	Next State		Output
	a = 0	a = 1	
→ q_0	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

Draw state diagram

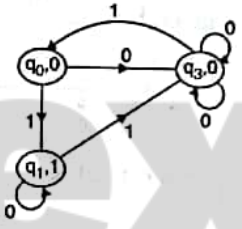


Fig. P. 1.5.21(b)

For input string 0 1 1 1 the transition of states is given by,

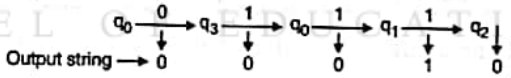


Fig. P. 1.5.21(c)

2. State table of Mealy machine

Present state	Next state			
	a = 0		a = 1	
	State	Output	State	Output
→ q_1	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

Draw a state diagram

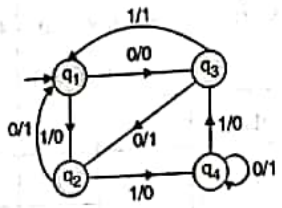


Fig. P. 1.5.21(d)

For input string 0 0 1 1

The transition of states is given by,

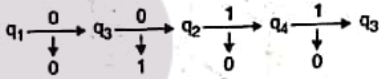


Fig. P. 1.5.21(e)

Output string is 0 1 0 0

Conversion of Mealy machine into Moore machine

☞ **Convert Mealy machine into Moore machine.**

Input - Mealy machine

Output - Moore machine

Step 1 : Calculate number of different outputs for each state that are in state table.

Step 2 : If all the outputs of Q_i are same, copy state Q_i. If it has n distinct outputs, break Q_i into n states where n = 0, 1, 2, ...

Step 3 : If output of initial state is 1, insert new initial state at the beginning which gives 0 output.

For example State table of mealy machine

Present	Next state			
	a = 0		a = 1	
	State	Output	State	Output
→ q_1	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

In this example,

q_1 is associated with output 1, q_0 is associated with two different outputs 0 and 1

Similarly q_3 is associated with 0 and q_4 is associated with two different output 0 and 1.

Split q_2 into q_{20} and q_{21}

Similarly q_4 into q_{40} and q_{41}

Reconstruct state table for new states

Present	Next state			
	State	Output	State	Output
	input a = 0		input a = 1	
→ q_1	q_3	0	q_{20}	0
q_{20}	q_1	1	q_{40}	0
q_{21}	q_1	1	q_{40}	0
q_3	q_{21}	1	q_1	1
q_{40}	q_{41}	1	q_3	0
q_{41}	q_{41}	1	q_3	0

Step 4. Pair of states and outputs in next column can be rearranged

Present	Next state		Output
	a = 0	a = 1	
→ q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

- In this state table, we observe that initial state q_1 is associated with output 1, it means that if machine starts at state q_1 we get output 1.
- Thus this Moore machine accepts a zero length sequence which is not accepted by mealy machine.
- To overcome this problem, we have to add new starting state q_0, whose state transition are similar to q_1 but output is 0.

Step 5 : State table for Moore machine

Present State	Next state		Output
	a = 0	a = 1	
→ q_0	q_3	q_{20}	0
q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

Conversion of Moore machine into Mealy machine.

How to Convert Moore machine into Mealy machine ? Explain it with example.

Input = Moore machine

Output = Mealy machine

Step 1 : Take black mealy machine transition table format.

Step 2 : Copy all moore machine transition state into this table format.

Step 3 : Check present states and their corresponding outputs in moore machine state table, if for a state Q_i output is m, copy it into output columns of mealy machines state table wherever Q_i appears in next state.

State table for moore machine

Present State	Next state		Output
	a = 0	a = 1	
→ q_0	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

Step 4 : Draw a blank mealy machine transition table format and copy all moore machine transition state into table.

Present state	Next state			
	a = 0		a = 1	
	State	Output	State	Output
→ q_0	q_3		q_1	
q_1	q_1		q_2	
q_2	q_2		q_3	
q_3	q_3		q_0	

Step 5 : Transition

Draw table for mealy machine

Present state	Next state			
	a = 0		a = 1	
	State	Output	State	Output
→ q_0	q_3	0	q_1	1
q_1	q_1	1	q_2	0
q_2	q_2	0	q_3	0
q_3	q_3	0	q_0	0

Difference between Mealy machine and Moore machine

☞ **Write the difference between Mealy Machine and Moore Machine.**

Sr. No.	Mealy Machine	Moore Machine
1.	Output depends upon present state as well as present input.	Output depends only on present state.
2.	It has less states than moore machine.	However, it has more states than mealy machine.
3.	These machines react faster to inputs.	In these machines more logic is needed to decode the output. Since it has more circuit delays.

□□□

CHAPTER 2

Formal Languages

University Prescribed Syllabus

Formal Languages : Defining Grammar, Derivations, Languges generated by Grammar, Chomsky Classification of Grammar and Languages, Recursive Enumerable Sets, Operations on Languages, Languages and Automata.

2.1 Introduction

- The formal languages in automata theory started in 1959, with formal definition given by Roam Chomsky. He tried to find out what is a grammar ? What is a mathematical model of grammar ?
- Let us for take some sentences in English and see how the sentences can be parsed.

For Example,

"The man ate the fruit".

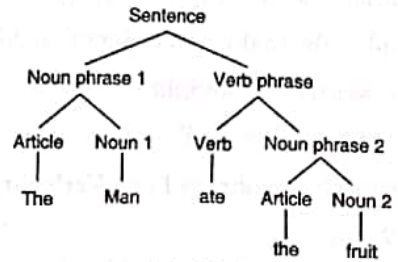


Fig. 2.1.1

- So this is parse tree for this sentence. This parse tree is syntax tree for grammar so it is called as derivation tree.
- In our example, 'Sentence' symbol goes to 'Noun phrase 1' and 'Verb phrase', 'Noun phrase 1' goes to 'Article' and 'Noun 1'. 'Article' goes to 'the' and 'Noun 1' goes to 'man'.
- 'Verb phrase' goes to 'Verb' and 'Noun phrase 2'. 'Noun phrase 2' goes to 'Article' and 'Noun 2'. 'Noun phrase 2' goes to article and 'noun 2'. 'Noun 2' goes to 'fruit'.
- Similarly, 'Verb' goes to 'ate', again 'Article' goes to 'the' which is already there.
- So these are called **rules** or **production** or **production rule**. Using these rules you can desire the sentence.

These rules are,

$$\begin{aligned} <S> &\Rightarrow <NP_1><VP> \\ <NP_1> &\Rightarrow <\text{Article}><N_1> \\ <\text{Article}> &\Rightarrow \text{the} \\ <N_1> &\Rightarrow \text{man} \\ <VP> &\Rightarrow <\text{Verb}><NP_2> \\ <\text{Verb}> &\Rightarrow \text{atc} \\ <NP_2> &\Rightarrow <\text{article}><N_2> \\ <N_2> &\Rightarrow \text{fruit} \end{aligned}$$

(i) Every derivation start with symbol S.

(ii) When we apply rule, that is written by a double arrow.

(iii) Left side is rewritten as the right hand side.

$$<S> \Rightarrow <NP_1><VP>$$

- Here S is rewritten as Noun phrase 1 and Verb phrase, this double arrow represents directly.
- So S directly derives 'Noun phrase 1' and 'Verb phrase' remains there.

$$<S> \Rightarrow <\text{Article}><N_1><VP>$$

- 'Noun phrase 1' directly derives 'Article' and 'Noun'. When we get something from something else, this is called as **sentential form**.
- In this, sentential form a 'Noun phrase 1' is rewritten as 'Article' and 'Noun 1'.
- Apply only one rule for one step :

$$\begin{aligned} <S> &\Rightarrow <\text{Article}><N_1><VP> \\ &\Rightarrow \text{The} <N_1><VP> \\ &\Rightarrow \text{The man} <VP> \\ &\Rightarrow \text{The man} <V><NP_2> \\ &\Rightarrow \text{The man ate} <NP_2> \\ &\Rightarrow \text{The man ate} <\text{Article}><N_2> \\ &\Rightarrow \text{The man ate the} <N_2> \\ &\Rightarrow \text{The man ate the fruit} \end{aligned}$$

2.1.1 Terminals and Non Terminal

☞ Define the term Terminals and Non Terminals.

(a) Terminals

Exam

- **Terminal Symbols** are literal symbols which may appear in the outputs of production rules of formal grammar and which cannot be changed using the rules of the grammar.
- In our previous example, the, man, ate, fruit; they cannot be rewritten as something else, the derivation terminates there, so these are called **terminals**.
- Set of terminals are denoted by 'T'
- Terminals are placed on right hand side in production rule.

(b) Non terminals

- **Non terminal symbols** are those symbols which can be replaced by groups of terminal symbols according to production rules.
- In our example, 'NP$_1$' is replaced by 'Article' and 'Article' is replaced by 'the'; so the 'NP$_1$' and 'Article' are non terminals and so on.
- Set of non terminal symbols are denoted by 'N'.
- Non terminal symbols are placed on left hand side in production rules.

Syllabus Topic : Defining Grammar

2.1.2 Grammar

☞ **Define Grammar.**

☞ **Write components of grammar explain it with example.**

A grammar consists of 4 tuples,

G = (N, T, P, S)

N = Finite set of non terminals

T = Finite set of terminals

P = Finite set of production or production rule

S = Start symbol ∴ S ∈ N

For example,

1. Grammar G_1 = ({S, A, B}, {a, b}, S, {S → AB, A → a, B → b})

Here,

Non terminal = S, A, B

Terminal symbol = a, b

S = start symbol, S ∈ N

Production P :

S → AB, A → a, B → a.

2. Grammar G_1 = ({S, A}, {a, b}, S, {S → aAb, aA → aaA, A → ε})

Here,

Non terminal = S, A

Terminal symbol = a, b

S = start symbol, S ∈ N

Production P :

S → aAb

aA → aaAb

A → ε

Syllabus Topic : Derivations, Languages Generated by Grammar

2.2 Derivations from Grammar and Languages Generated by Grammar

2.2.1 Languages Generated by Grammar

☞ **How to generate language by grammar explain it with example ?**

Set of all strings that can be derived from grammar is said to be **Language** generated from that grammar.

For example,

1. Consider the grammar

G_1 = ({S, A}, {a, b}, S, {S → aAb, aA → aaAb, A → ε})

S → aAb ...[By production rule S → aAb]

→ aaAbb ...[aA → aaAb]

→ aaaAbbb ...[aA → aaAb]

→ aaabbb ... [by A → ε]

Final string aaabbb is derived from this grammar.

2.

G_2 = ({S, A, B}, {a, b}, S, {S → AB, A → a, B → b})

$S \rightarrow AB$

$\rightarrow ab$

$L(G_2) = \{a\ b\}$ ...Language generated by this grammar

3. $G_3 = (\{S, A, B\}, \{a, b\}, S, \{S \rightarrow AB, A \rightarrow aA \mid a, B \rightarrow b\ B \mid b\})$

$S \rightarrow \underline{A}B$

$\rightarrow ab$

$S \rightarrow \underline{A}B$ ...[by $A \rightarrow aA$, $B \rightarrow bB$]

$\rightarrow aAbB$ ...[by $A \rightarrow a$, $B \rightarrow b$]

$\rightarrow aabb$

$S \rightarrow AB$ ...[by $A \rightarrow aA$; $B \rightarrow b$]

$\rightarrow aAb$ ...[$B \rightarrow b$]

$\rightarrow aab$

Depending upon choice we can generate many strings.

$L(G_3) = \{aa, a^2b^2, a^2b \ldots.\}$

Generalise language

$L(G_3) = \{a^m b^n \mid m \geq 0 \text{ and } n \geq 0\}$

exam

2.2.2 Converting Language into Grammar

How to generate grammar by language explain it with example ?

Consider some language and convert it into grammar G which produces those language.

For example,

1. Lets consider $L = \{a^n b^m \mid n \geq 1 \text{ and } m \geq 0\}$

Language $L(a^n b^m)$ generate strings which having any power of n and m

For $a^n = \{a, aa, aaa \ldots\}$

$A \rightarrow aA \mid a$

Consider if $m = 0$

$\therefore\ b^m \Rightarrow b^0 \Rightarrow 1 \Rightarrow \varepsilon$

$b^m = \{\varepsilon, b, bb, bbb \ldots\}$

$B \rightarrow bB \mid \varepsilon$ ...This is grammar for b^m

Grammar is;

$S \rightarrow AB$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid \varepsilon$

2. $L = \{a^n b^m \mid n \geq m\}$

$L(a) = \{ab, aab, aabb, aaabb \ldots\}$

Generate more number of a than b.

$A \rightarrow aA \mid \varepsilon$

Generate equal number of a and b.

$S \rightarrow a\,Ab \mid asb$

$A \rightarrow aA \mid \varepsilon$

Syllabus Topic : Chomsky Classification of Grammar and Languages

2.3 Chomsky Classification of Languages

2.3.1 Classification of Grammar

Explain Chomsky Classification of grammar.

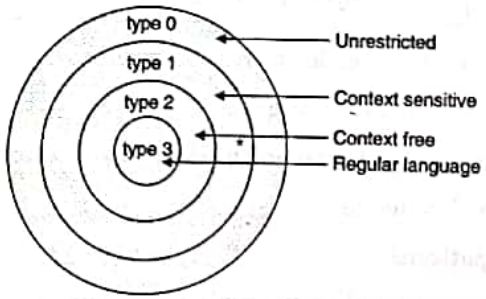


Fig. 2.3.1

- Grammar is classified by considering the form of production.
- They are classified into four types in term of production. **(type 0, type 1, type 2 and type 3).**
- Classification provides a basic for understanding the relationship between grammars.

Class	Grammars	Languages	Automaton
type 0	Unrestricted	Recursive enumerable	Turing machine
type 1	Context sensitive	Context sensitive	Linear bound
type 2	Context free	Context free	Pushdown
type 3	Regular	Regular	Finite

- Type 0 is unrestricted grammar and Type 3 is more restricted grammar

Note : Kleen star : Σ^* → set of all strings

Kleen plus : $\Sigma^+ \rightarrow \Sigma^* - \varepsilon$

(Σ^+ is positive clousure)

Type 3 : Regular grammar

- **Explain type 3 Grammar.**
- **Which machine is used for type 3 ?**
- **Which language is used for type 3 ?**

- Type 3 grammar is also called as regular grammar.
- It generates **regular language.**
- Language is generated by using **finite state automaton machine.**
- Use of Regular languages
  1) Search patterns
  2) Lexical structure of programming language.

Regular grammar is divided into two types :

1. Right linear grammar
2. Left linear grammar

1. Right linear grammar

- **Explain Right linear grammar.**

- A grammar is said to be **right linear** if all production are of the form,

$$A \rightarrow xB$$
$$A \rightarrow x$$

Where A, B ∈ N and x ∈ T

N : Non terminal

T : Terminal

$A \rightarrow xB$

- If non terminal symbols lies to the right of the terminal symbol, then it is said to be a **right linear grammar.**

For example : $S \rightarrow abs \mid b$

a, b = terminal, S = non terminal

2. Left linear grammar

- **Explain Left linear grammar.**

- A grammar is said to be **left linear** if all productions are of the form,

$$A \rightarrow Bx$$
$$A \rightarrow x$$

Where A, B ∈ N and x ∈ T

- If non terminal symbol lies to the left of terminal symbol, then it is said to be a **left linear grammar.**

Here B is a non terminal and x is terminal.

e.g.

$S \rightarrow Sbb|b$

b = terminal, S = Non terminal

Example :

$A \rightarrow \varepsilon$

$A \rightarrow a$

$A \rightarrow abc$

$A \rightarrow B$

$A \rightarrow abcB$

Type 2 : Context free grammar

- Explain type 2 Grammar.
- Which machine is used for type 2 ?
- Which language is used for type 2 ?

- Type 2 grammar generates **context free language.**
- These are defined by rules of the form $A \rightarrow y$,

  where $A \in N$ [Non terminal]

  $y \in (T \cup N)^*$ (string of terminals and non terminals)
- These languages are exactly all languages that can be recognized by non **deterministic pushdown automaton.**
- Context free languages are theoretical basis for syntax of most programming languages.

  e.g.

$S \rightarrow aS$

$S \rightarrow aSa$

$S \rightarrow aA$

$S \rightarrow aAB$

$A \rightarrow a$

$B \rightarrow b$

Type 1 : Context sensitive grammar

- Explain type 1 Grammar.
- Which machine is used for type 1 ?
- Which language is used for type 1 ?

- Type 1 grammar generates the **context sensitive languages.**
- These grammars have rule of the form,

  $\alpha A \beta \rightarrow \alpha y \beta$

  where,

  A = Non-terminal

  $\alpha, \beta, y \in (T \cup N)^*$ (string of terminals and non terminals)

  the string α and β may be empty but y must be non empty.
- Languages described by there grammars are exactly all languages that can be recognised by **Linear Bounded Automaton.**

  e.g.

$AB \rightarrow AbBC$

$A \rightarrow bcA$

$B \rightarrow b$

Type 0 : Unrestricted grammar

- Explain type 0 Grammar.
- Which machine is used for type 0 ?
- Which language is used for type 0 ?

- It is also known as **recursively enumerable languages.**
- Type 0 grammar includes all formal grammars.
- The production have no restrictions.
- Unrestricted grammar can be recognized by a **Turing machine.**

- Production rule can be in the form of :

$\alpha \rightarrow \beta$

α = string of terminals and non terminals with at least one non terminal, α should not be null or empty.

β = string of terminals and non terminal

$\alpha, \beta \in (T \cup N)^*$ (string of terminals and non terminals)

e.g.

$S \rightarrow ACaB$

$BC \rightarrow aCB$

$CB \rightarrow DB$

$aD \rightarrow Db$

Syllabus Topic : Operations on Languages

2.3.2 Operations on Languages

What are different operations performed on languages ?

1. Union

$$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$$

It is exactly same line a sets

2. Concatenation

$$A \cdot B = \{xy \mid x \in A \text{ and } y \in B\}$$

It means that joining two symbols together.

3. Star

$$A^* = \{x_1, x_2, x_3 \ldots x_k \mid k \geq 0 \text{ and } x \in A\}$$

It means take as many symbols that you want from set A and join them together.

For example,

$A = \{pq, r\}, B = \{t, uv\}$

Union

$A \cup B = \{pq, r, t, uv\}$

Concatenation

$A \cdot B = \{pqt, pquv, rt, ruv\}$

Star

$A^* = \{\varepsilon, pq, r, pqr, rpq, pqpq, rrpqpqpq \ldots\}$

($\because \varepsilon$ = empty string)

Theorem 1

- The class of regular languages is closed under **union.**
- This means that if we have two regular languages A and B. If we perform union operation on A and B, then the resultant of **(Please check sentence)**union of A and B are also regular language.

Theorem 2

- The class of regular language is closed under **concatenation.**
- This means that it we have two regular languages A and B. If we perform concatenation operation on A and B then resultant of **(Please check sentence)** concatenation of A and B are regular language.

Syllabus Topic : Languages and Automata

2.4 Languages and Automata

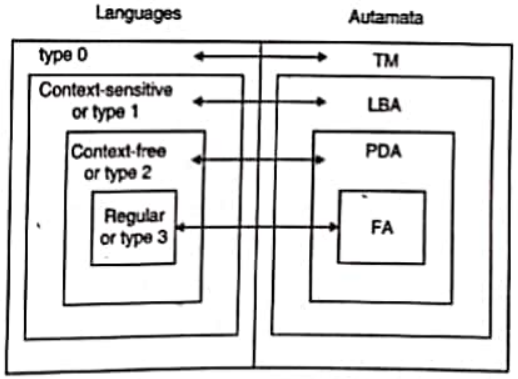


Fig. 2.4.1 : Languages and corresponding automata

TM = Turing machine

LBA = Linear Bounded Automaton

PDA = Pushdown Automation

FA = Finite Automaton

□□□

CHAPTER 3

Regular Set and Regular Grammar

University Prescribed Syllabus

Regular Sets and Regular Grammar : Regular Grammar, Regular Expressions, Finite automata and Regular Expressions, Pumping Lemma and its Applications, Closure Properties, Regular Sets and Regular Grammar.

Syllabus Topic : Regular Grammar

3.1 Regular Grammar

- A regular grammar is a mathematical object,
- Grammar having four tuples, G = (N, Σ, P, S), where
  N is a nonempty, finite set of non-terminal symbols,
- Σ is a finite set of terminal symbols , or alphabet, symbols,
- P is a set of grammar rules,
- S is the start symbol. Where S ∈ N (S belong to N)
- Every regular grammar describes a regular language. A **regular grammar** is formal grammar that is right-regular or left-regular.
- A **right regular grammar** is also called **right linear grammar** such that all the production rules in P are in following forms:

  $A \rightarrow a$

  $A \rightarrow aB$

  $A \rightarrow \varepsilon$.

where A and B are non-terminals in N and a is in Σ and ε denotes the empty string

- **Example :**

  $S \rightarrow abS$, $S \rightarrow b$

  In this example, S is nonterminal and a,b are terminals

- A left regular grammar is also called left linear grammar , such that all the production rules in P are in following forms:

  $A \rightarrow a$

  $A \rightarrow Ba$

  $A \rightarrow \varepsilon$.

  where A and B are non-terminals in N and a is in Σ and ε denotes the empty string [note :empty string means string of length is 0]

- For Example :

  $S \rightarrow Sbb$, $S \rightarrow b$

- In this example, S is non terminal and b is terminal.

Syllabus Topic : Regular Expressions

3.2 Regular Expression

- Regular expression is useful for representing the set of strings. Regular expression describes the languages accepted by finite automata.

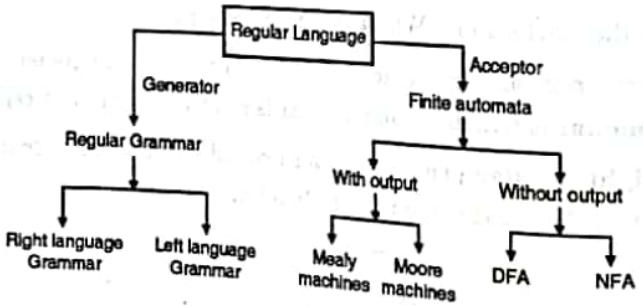


Fig. 3.2.1

- Regular expression contains **three** operations :
  1. **union (+)**
  2. **concatenation (.)**
  3. **Iteration on kleene closure (*)**

3.2.1 Primitive Expression

☞ **Write a primitive expression ?**

- ϕ and ε are an empty set. These are both regular expression .

  ϕ = regular expression represents { }

  ε = regular expression represents {ε}

  $a \in \Sigma$ (a belongs to Σ) is also regular expression.

  $a \in \Sigma$ = represents {a}.

- These regular expressions are called as **primitive expression.**
- If we take any primitive expressions r_1 and r_2 and apply all operations.

1. **union operation**

   $r_1 + r_2$

2. **concatenation operation**

   $r_1 \cdot r_2$

3. **kleene closure operation**

   r_1^*

3.2.2 Rules for Regular Expression

☞ **Write a rules for regular expression ?**

- Regular expressions are used for representing certain sets of strings in an algebraic fashion.
- **Rules** for Regular expression.

1. Any **terminal symbol** i.e. symbols belongs to Σ including empty (ε) and Null (ϕ) are regular expression.

e.g. a, b, ... A, Q – are regular expression

2. The **union of two regular expressions** is also regular expression.

   Let consider R_1 and R_2 are regular expression, then $R_1 \cup R_2$ or $(R_1 + R_2)$ are regular expression.

3. The **concatenation of two regular** expression is also regular expression.

   Let consider R1 and R2 are regular expression

   $R_1 \cdot R_2$ ---- are R.E.

   Concatenation denoted by (·)

4. The **iteration or closure of regular expression** is also a regular expression.

   $R \rightarrow R^*$

   e.g. $a^* = \varepsilon, a, aa, aaa.....$

5. The RE over Σ are precisely those obtained recursively by application of above rules once or several times.

For example :

Describe the following sets as regular expression.

1. {0,1,2}

   R = 0 + 1 + 2 [+ symbol denotes or]

2. {ε, ab}

   R = ε ab [here we have not used + symbol]

3. {abb, a, b, bba}

   R = abb + a + b + bba

4. { ε, 0, 00, 000}

   R = 0* closure of 0

5. {1,11,111,1111...}

   R = 1+

3.2.3 Regular Set

What is a Regular set ?

Any set represented by regular expression is called **regular set.**

For example,

- a, b $\in \Sigma$ then,
  - (i) a denotes the set {a}
  - (ii) a + b denotes set {a, b}
  - (iii) ab denotes {a, b}
  - (iv) a* denotes set {ε, a, aa, aaa ---}
  - (v) (a + b)* denotes {a b}*
- The set is represented by R.
- Language is denoted by L(R)
- These are three basic operations
  - (i) union (+)
  - (ii) concatenation (·)
  - (iii) iteration or closure (*)

For Example, describe the following sets by regular expression.

1. L_1 = set of all stirngs of 0's and 1's ending in 00.

   $\Rightarrow L_1$ is obtained by concatenary any string over {0,1} and 00.

   {0,1} is represented by 0 + 1

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

2. L_2 = set of all strings of 0's and is beginning with 0 and ending with 1.

   $\Rightarrow L_2$ is obtained by concatenating 0, anything over {0,1} and 1.

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

3. L_3 = {ε, 11, 111, 1111, ...}

   $\Rightarrow L_3$ is either ε or string of even number of 1's. i.e. string is in of $(11)^n$ where $n \geq 0$.

$L_2 = (11)^*$

4. Lets consider $\Sigma = (a, b)$

L_1 = Set of all strings which are having length **exactly two**. Find regular expression.

$\Rightarrow L_1 = \{aa, ab, ba, bb\}$

$\therefore$ R.E. $= aa + ab + ba + bb = a(a + b) + b(a + b)$

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

5. Find regular expression, which represents set of all string which are having length **at least two.**

$\Rightarrow L_1 = \{aa, ab, ba, bb, aaa.....\}$

Language is infinite language.

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

6. Find regular expression which represents set of all strings which are having length **at most 2** (at most means length is 0, 1, 2).

$\Rightarrow L_1 = \{\varepsilon, a, b, aa, ab, ba, bb\}$

$\therefore (a + b + \varepsilon) \quad (a + b + \varepsilon)$

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

Even length strings

$L = \{\varepsilon, aa, ab, ba, bb, ...\}$

(∵length is 0, 2, 4, 6....)

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

8. **Odd length** strings

(length is 1,3,....)

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

9. Length is divisible by '3'

Language contains length 0,3,9,12...

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

10. Number of a's have to be **exactly 2**

$\Rightarrow b^*ab^*ab^*$

11. **a's at least 2**

$\Rightarrow b^*ab^*a\,(a + b)^*$

12. **a's at most 2**

$\Rightarrow b^*\,(\varepsilon + a)\,b^*\,(\varepsilon + a)b^*$

13. Number of a's are **even**

$\Rightarrow (b^*ab^*ab^*)^* + b^*$ or $(b^*a\,b^*ab^*)\cdot b^*$

14. **Starts** with a's

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

15. **Ends** with a's

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

16. **Containing** a's

$\Rightarrow (a + b)^*\,a\,(a + b)^*$

17. **Starting** and **ending** with **different symbols**

$\Rightarrow a\,(a + b)^*\,b$ or $b\,(a + b)^*\,a$

18. Starting and ending with **same symbols**

L is $\{\varepsilon, a, b, aa, bb.....\}$

$\Rightarrow a\,(a + b)^*\,a + b\,(a + b)^*\,b + a + b + \varepsilon$

19. Not a's come together (two aa's **should not come together**)

$\Rightarrow$ L is $\{\varepsilon, b, bb, bbb, a, ab, aba\ abab, ababa, ba, bab, baba, babab.....\}$

$(b + ab)^*\,(\varepsilon + a)$

3.2.4 Identities of Regular Expression

Write a identities of regular expression

1. $\phi + R = R$ .... ϕ = Empty set union of ϕ and R where R is Regular expression.
2. $\phi R + R\phi = \phi$
   ϕ concatenation R Union R concatenation $\phi = \phi$
3. $\varepsilon R = R\varepsilon = R$.
4. $\varepsilon^* = \varepsilon$ and $\phi^* = \varepsilon$
5. $R + R = R$
   RE union RE = RE
6. $R^* R^* = R^*$
   Concatenation of closure of two regular expressions = closure of Regular expression.
7. $RR^* = R^*R$
8. $(R^*)^* = R^*$
9. $\varepsilon + RR^* = \varepsilon + R^*R = R^*$
10. $(PQ)^*P = P(QP)^*$
11. $(P + Q)^* = (P^* Q^*)^* = (P^* + Q^*)^*$
12. $(P + Q) R = PR + QR$ and
    $R(P + Q) = RP + RQ$

3.2.5 Arden's Theorem

If P and Q are two regular expressions over Σ and if P does not contain ε, then the following equation in R given by $R = Q + RP$ has a unique solution i.e $R = QP^*$

Proof

$$R = Q + RP \quad \ldots(3.2.1)$$

Replace $R = QP^*$,

$$= Q + QP^* P$$

$$= Q(\varepsilon + P^* P) \quad \ldots\text{Identity } [\varepsilon + R^*R = R^*]$$
$$= QP^*$$

Prove that $R = QP^*$ is a solution.

∴To prove that this is a unique solution.

$$R = Q + RP \quad [\text{Replace R with } Q + RP]$$
$$= Q + [Q + RP] P$$
$$= Q + QP + RP^2 \quad [\text{R is a replaced by } Q + RP]$$
$$= Q + QP + [Q + RP] P^2$$
$$= Q + QP + QP^2 + RP^3$$
$$\vdots$$
$$= Q + QP + QP^2 + \ldots + QP^n + RP^{n+1}$$

Replace $R = QP^*$

$$= Q + QP + QP^2 + \ldots + QP^n + QP^* P^{n+1}$$
$$= Q [\varepsilon + P + P^2 + \ldots + P^n + P^* P^{n+1}]$$
$$R = QP^*$$

This is a unique solution.

Example proofs using Identities of Regular expression

Example 3.2.1 : Prove that $(1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^* (0 + 10^*1)$ is equal to $0^*1(0 + 1^* 1)^*$

Solution :

$$\text{LHS} = (1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^* (0 + 10^*1)$$
$$= (1 + 00^*1)[\varepsilon + (0 + 10^*1)^* (0 + 10^*1)] \quad \ldots \varepsilon + R^* R = R^*$$
$$= (1 + 00^*1)(0 + 10^*1)^*$$
$$= (\varepsilon \cdot 1 + 00^*1)(0 + 10^*1)^* \quad \ldots \varepsilon \cdot R = R$$
$$= (\varepsilon + 00^*) 1 (0 + 10^*1)^*$$
$$= 0^*1 (0 + 10^* 1)^*$$
$$= \text{R.H.S}$$

Example 3.2.2 : Give an Regular expression for representing the set L of strings in which every 0 is immediately followed by at least two 1's.

Solution. :

String does not contain only zero, string contains 0 preceded by 1 and followed by 11 ;

so $L = (1 + 011)^*$

Example 3.2.3 : Prove that regular expression,
$R = \varepsilon + 1^* (011)^* (1^*(011)^*)^*$
also describes the same set strings in Example 3.1.2.

Solution. :

$$R = \varepsilon + P^1P_1^*$$

where $P_1 = 1^* (011)^*$

$R = P_1^*$ ...using identity $\varepsilon + RR^* = R^*$

$= (1^*(011)^*)^*$

Lets consider $P_2 = 1$ and $P_3 = 011$

$= \left(P_2^*P_3^*\right)^*$

$= (P_2 + P_3)^*$ by using identity $\left(P^* + Q^*\right)^* = \left(P^* + Q^*\right)^*$

$R = (1 + 011)^*$

Syllabus Topic : Finite Automata and Regular Expression

3.3 Finite Automata and Regular Expression

In this topic, we will study the representation of Regular expression.

3.3.1 Transition System Containing ε Moves

Write a steps for converting transition system with ε moves into transition system without ε moves ? Explain it with example

- ε-transition or ε moves which are associated with null symbol.
- Transition system can be generalized by permitting ε moves.
- These transition can occur when no input is applied.

- We can convert transition system with ε moves into transition system without ε moves.

Steps for converting transition system with ε moves into transition system without ε moves

1. First find out all ε transitions from each state from Q, that is called as ε-closure (q_i) when $q_i \in Q$.

Note : Every state on ε goes to itself.

2. The δ' transition can be obtained. The δ' transition means an ε-closure on δ moves.
3. Step 2 is repeated for each input symbol and for each state of given NFA.
4. Using the resultant states the transition table for equivalent NFA without ε moves can be built.

Note : ε* closure- All states that can be reached from a particular state only by seeing ε b symbol.

Example 3.3.1 : Converting ε NFA to its equivalent NFA.

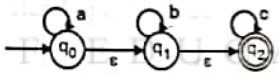


Fig. P. 3.3.1(a)

Solution :

This is for q_0 state

For a input

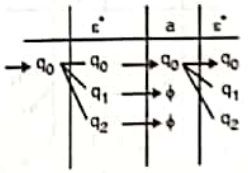


Fig. P. 3.3.1(b)

For b input

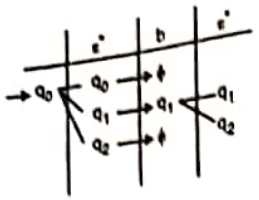


Fig. P. 3.3.1(c)

For c input

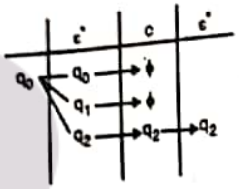


Fig. P. 3.3.1(d)

Now find out for q_1 state,

For a input

Fig. P. 3.3.1(e)

For b input

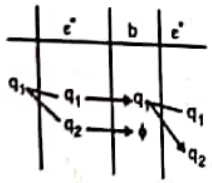


Fig. P. 3.3.1(f)

For c input

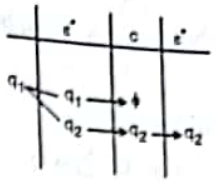


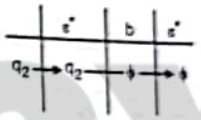
Fig. P. 3.3.1(g)

Now find out for q_2 state,

For a input

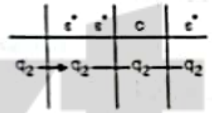
(h)

For b input



(i)

For c input



(j)

Fig. P. 3.3.1

State transition table

	a	b	c
→ 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

Fig. P. 3.3.1(k)

Transition diagram

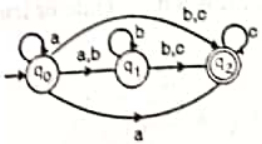


Fig. P. 3.3.1(l)

3.3.2 Conversion of NFA to DFA

- Every DFA is an NFA, but not viceversa, but there is an equivalent DFA for every NFA.
- Transition function of DFA is,

$$\delta = Q \times \Sigma \rightarrow Q$$

- Transition function of NFA is,

$$\delta = Q \times \Sigma \rightarrow 2^Q$$

- We can convert every NFA into DFA.

For Example

L = {Set of all strings over (0, 1) that states with '0'}.

Σ = {0, 1}

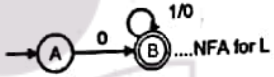


Fig. 3.3.1

State transition diagram

State s	0	1
A	B	ϕ
B	B	B

Convert the NFA into DFA

Transition State

	0	1	
A	B	C	... C is a dead state or trap state
B	B	B	
C	C	C	

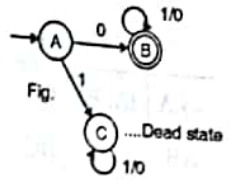


Fig.

Fig. 3.3.2

3.3.3 Conversion of DFA to NFA

Find the equivalent DFA for the NFA given by L = [{A, B, C}, (a, b), δ, A, {C}]

Which δ is given by :

	a	b
→ A	A, B	C
B	A	B
©	–	A, B

- **States :** A, B, C
- Inputs : a, b
- Transits function :δ
- Initial state : A
- Final state : C

State diagram for NFA

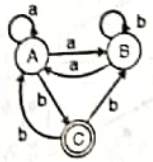


Fig. 3.3.3

Convert NFA into DFA

	a	b
→ A	{A, B}	C
AB	AB	BC
(BC)	A	AB
(C)	D	AB
D	D	D

Note : To find AB states, we must look at transition state diagram of NFA and apply union operation.

In DFA we can not leave a state, so we have to write new state. In our example we have written D is a new state which is called dead state.

The input that comes in dead state remains there.

To find state of DFA

- Check the final state of NFA and after that select the state from DFA that contains the final state of NFA.
- In our example, 'C' state is a final state in NFA. So, select state from DFA which contains 'C' state in DFA. So we get 2 final states in our example.

State diagram for DFA

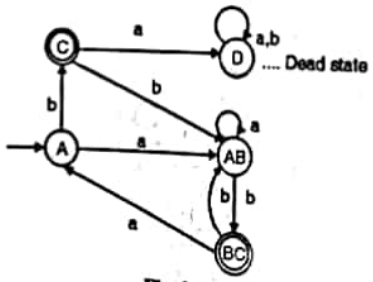


Fig. 3.3.4

Example 3.3.2 : Given below is the NFA for a language,

L = { set of all strings over (0, 1) that ends with '01' }

Construct its equivalent DFA

Solution :

⇒ First draw NFA

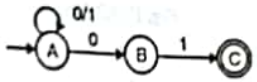


Fig. P. 3.3.2(a)

Transition table

	0	1
→ A	A, B	A
B	ϕ	C
(C)	ϕ	ϕ

Convert NFA into DFA

Transition table for DFA

	0	1
→ A	AB	A
AB	AB	AC
(AC)	AB	A

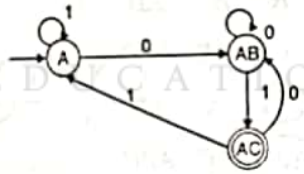


Fig. P. 3.3.2(b)

Example 3.3.3 : Design an NFA for language that accepts all strings over {0, 1} in which the second last symbol is always '1'. Then convert it to its equivalent DFA.

e.g. string 1 0 1 0 or 1 1 0 or 1 1 0 1 0 1 0

Solution. :

Design NFA

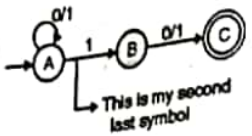


Fig. P. 3.3.3(a)

State transition table for NFA

	0	1
→ A	A	A, B
B	C	C
(C)	ϕ	ϕ

Convert into DFA

Transition table

	0	1
→ A	A	AB
AB	AC	ABC
(AC)	A	AB
(ABC)	AC	ABC

Fig. P. 3.3.3(b)

3.3.4 Algebraic Methods Using Arden's Theorem

We consider following assumption regarding the transition system,

1. The transition graph or diagram does not have ε moves,
2. It has only one initial state say V_1,
3. Its vertices are $v_1, v_2 \ldots v_n$.
4. α_{ij} denotes regular expression representing the set of labels of edges from V_i to V_j.

$$V_1 = v_1\alpha_{11} + v_2\alpha_{21} \ldots + v_n\alpha_n + A$$
$$V_2 = v_1\alpha_{12} + v_2\alpha_{22} \ldots + v_n\alpha_{n2}$$
$$\vdots$$
$$V_n = v_1\alpha_{1n} + v_2\alpha_{2n} \ldots + v_n\alpha_{nn}$$

by repeatedly applying substitutions and applying Arden's theorem we can express V_i in terms at α_{ij}'s

Example 3.3.4 : Consider the transition system given in Fig.**, Prove that the strings recognized are :

$(a + a\,(b + aa)^*\,b)^*a(b + aa)^*a$

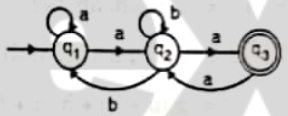


Fig. P. 3.3.4

Solution :

There is no any ε move and there is only one initial state.

3 equations for q_1, q_2, q_3 can be written as,

$$q_1 = q_1 a + q_2 b + \varepsilon \quad \ldots(1)$$
$$q_2 = q_1 a + q_2 b + q_3 a \quad \ldots(2)$$
$$q_3 = q_2 a \quad \ldots(3)$$

by substituting Equation (3) in Equation(2) we get,

$$q_2 = q_1 a + q_2 b + q_2 aa$$
$$= q_1 a + q_2 (b + aa)$$
$$= q_1 a\,(b + aa)^* \quad \ldots(4)$$

by substituting Equation (4) in Equation (1) we get,

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

Hence,

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

$$q_2 = (a + a (b + aa)^* b)^* a (b + aa)^*$$

$$q_3 = (a + a (b + aa)^* b)^* a (b + aa)^* a$$

Since, q_3 is a final state, the set of strings recognized by graph is given by,

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

3.3.5 Designing Regular Expressions

For example, design regular expression for the following language over {a, b}

1. L = accepting strings of length exactly 2

$$\Rightarrow L_1 = \{aa, ab, ab, bb\}$$

$$R = aa + ab + ba + bb$$

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

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

2. L = accepting string of length at least 2

$$\Rightarrow L_1 = \{aa, ab, ba, bb, aaa \ldots.\}$$

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

3. L = accepting strings of length atmost 2

$$\Rightarrow L_1 = \{\varepsilon, a, b, aa, ab, ba, bb\}$$

$$R = \varepsilon + a + b + aa + ab + ba + bb$$

$$= (\varepsilon + a + b) (\varepsilon + a + b)$$

For example,

1. **Find the regular expression for the following NFA.**

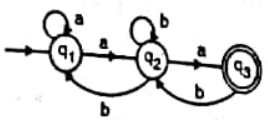


Fig. 3.3.5

We have written equation for each states of NFA and put all Equations of all states to the equation of final state and that gives regular expression for given NFA.

For writing equation, check incoming input transition.

$$q_3 = q_2 a \quad \ldots(3.3.1)$$

$$q_2 = q_1 a + q_2 b + q_3 b \quad \ldots(3.3.2)$$

$$q_1 = \varepsilon + q_1 a + q_2 b \quad \ldots(3.3.3)$$

Simplify these Equation,

$$q_3 = q_2 a$$

$$= (q_1 a + q_2 b + q_3 b) a$$

$$= q_1 aa + q_2 ba + q_3 ba \quad \ldots(3.3.4)$$

$q_2 = q_1 a + q_2 b + q_3 b$...Putting value of q_3 from Equation (3.3.1)

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

$$= q_1 a + q_2 b + q_2 ab$$

$$\underbrace{q_2}_{R} = \underbrace{q_1 a}_{Q} + \underbrace{q_2}_{R} \underbrace{(b + ab)}_{P}$$

...(Informed R = Q + RP R = QP* by Arden's theorem)

$$\therefore q_2 = (q_1 a) (b + ab)^* \quad \ldots(3.3.5)$$

$$q_1 = \varepsilon + q_1 a + q_2 b$$

Putting value of q_2 from Equation (3.3.5)

$$q_1 = \varepsilon + q_1 a + ((q_1 a) (b + ab)^*) b$$

$$\underbrace{q_1}_{R} = \underbrace{\varepsilon}_{Q} + \underbrace{q_1}_{R} \underbrace{((a + a) (b + ab)^*) b}_{P}$$

...Which is in form R = Q + RP, R = QP*

$$q_1 = \varepsilon ((a + a (b + ab)^*) b)^* \quad \ldots\text{By identities } \varepsilon \cdot R = R$$

$$q_1 = (a + a (b + ab)^* b)^* \quad \ldots(3.3.6)$$

Final state q_3,

$$q_3 = q_2 a$$

$$= q_1 a (b + ab)^* a \quad \text{...Putting value of } q_2 \text{ from (3.3.5)}$$

$$= ((a + a (b + ab)^* b)^* a (b + ab)^* a \quad \text{...Putting value of } q_1 \text{ from (3.3.6)}$$

2. Find the regular expression for the following DFA

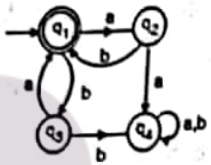


Fig. 3.3.6

Here q_1 is a final state

For writing equation check input transitions in each state

$$q_1 = \varepsilon + q_2 b + q_3 a \quad \text{...(3.3.7)}$$

$$q_2 = q_2 a \quad \text{...(3.3.8)}$$

$$q_3 = q_1 b \quad \text{...(3.3.9)}$$

$$q_4 = q_2 a + q_3 + b + q_4 a + q_4 b \quad \text{...(3.3.10)}$$

$$q_1 = \varepsilon + q_2 b + q_3 a$$

Putting values of q_2 and q_3 from Equations (3.3.7) and (3.3.9)

$$q_1 = \varepsilon + q_1 ab + q_1 ba$$

$$\underbrace{q_1}_{R} = \underbrace{\varepsilon}_{Q} + \underbrace{q_1}_{R} \underbrace{(ab + ba)}_{P} \quad \text{...In form of } R = Q + RP$$

$R = QP^*$ by Arden's theorem

$$q_1 = \varepsilon (ab + ba)^*$$

$$q_1 = (ab + ba)^* \quad \text{...}\varepsilon R = R$$

3.3.6 Construction of Finite Automata

Construction of finite automata equivalent to regular expression

This method we use, to construct a finite automation equivalent to a given regular expression is called subset method.

Some basic element :

(i) (a + b) means a union b.

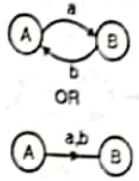


Fig. 3.3.7

(ii) (a · b) means a concatenation with b.

Fig. 3.3.8

(iii) a^* means closure of a

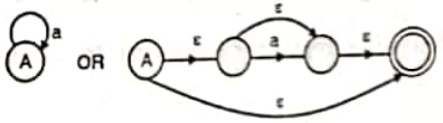


Fig. 3.3.9

(iv) ϕ

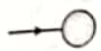


Fig. 3.3.10

(v) a

Fig. 3.3.11

(vi) ab^*

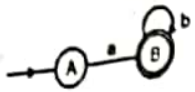


Fig. 3.3.12

(vii) $(ab)^*$

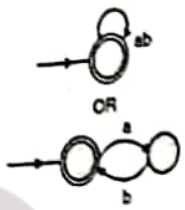


Fig. 3.3.13

(viii) $(ab + ba)^*$

Fig. 3.3.14

3.3.7 Conversion of Regular Expression to Finite Automata

Convert the following regular expressions to their equivalent finite automata.

For example

1. **ba^*b :**

String accepted like {bb, bab, baab...}

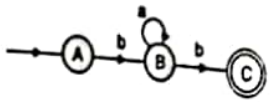


Fig. 3.3.15

2. **$(a + b)\,c$:**

In this case first is union operation followed by concatenation.

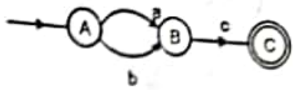


Fig. 3.3.16

String accepted by the R.E are {ac, bc}

3. **$a\,(bc)^*$**

String accepted by the R.R. are {a, abc, abcbc...}

Fig. 3.3.17

4. **$(a \mid b)^*\,(abb \mid a^*b)$**

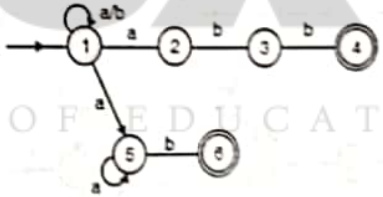


Fig. 3.3.18

$a^+ = \{a, aa, aaa....\}$

5. **$10 + (0 + 11)0^*1$**

Step I :

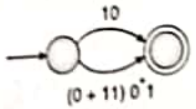


Fig. 3.3.19

Step II :

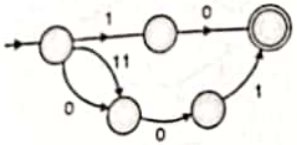


Fig. 3.3.20

Step III :

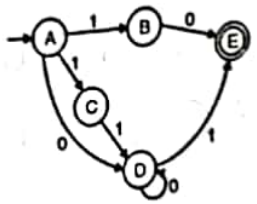
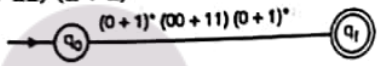


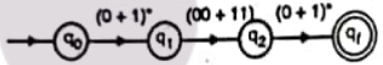
Fig. 3.3.21

6. $(0 + 1)^* (00 + 11) (a + 1)^*$

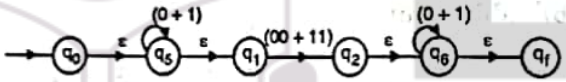
Step I :



Step II :



Step III :



Step IV :

Fig. 3.3.22

Step V :

Remove the ε moves.

Step VI :

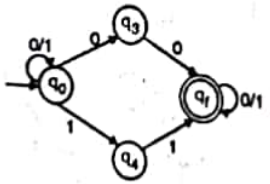


Fig. 3.3.23

Constructing a finite automaton equivalent to

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

Step VII : Construct transition table for NFA.

	0	1
$\longrightarrow q_0$	q_0, q_3	q_0, q_4
q_3	q_f	ϕ
q_4	ϕ	q_f
q_f	q_f	q_f

Step VIII : Construct transition table for DFA using transition table of NFA.

	0	1
$\longrightarrow q_0$	$\{q_0, q_3\}$	$\{q_0, q_4\}$
$\{q_0, q_3\}$	$\{q_0, q_3, q_f\}$	$\{q_0, q_4\}$
$\{q_0, q_3, q_f\}$	$\{q_0, q_3, q_f\}$	$\{q_0, q_4, q_f\}$
$\{q_0, q_4\}$	$\{q_0, q_3\}$	$\{q_0, q_3, q_f\}$
$\{q_0, q_4, q_f\}$	$\{q_0, q_3, q_f\}$	$\{q_0, q_4, q_f\}$

Step IX : Construct transition diagram for DFA

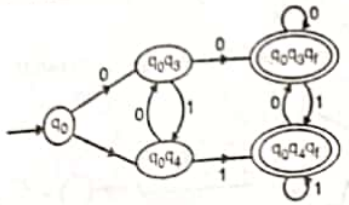


Fig. 3.3.24 : Finite automata

3.3.8 Conversion of F.A. to R.E.

State elimination method

Rules :

(i) Initial state could not have any incoming edge.

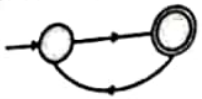


Fig. 3.3.25

If any incoming edge there, create new initial state with ∈ move.

Fig. 3.3.26

(ii) Final state should not have any outgoing edge.

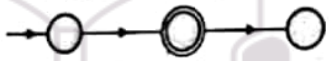


Fig. 3.3.27

If any outgoing edge of there, create new final state with ε move.

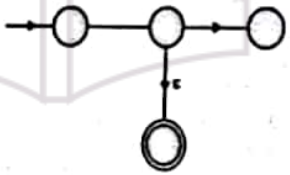


Fig. 3.3.28

If we have more than one final state then make it into one final state.

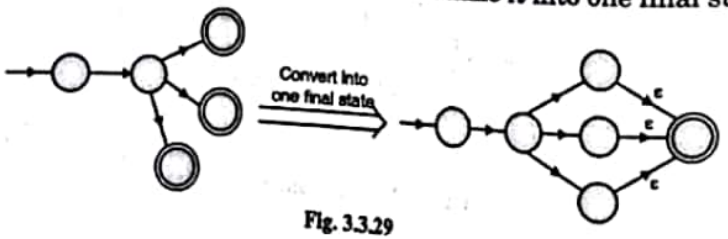


Fig. 3.3.29

(iii) Eliminate other state apart from initial and final state.

Example :

1.

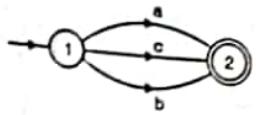


Fig. 3.3.30

StepI :

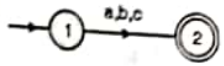


Fig. 3.3.31

R.E. = a + b + c

2.

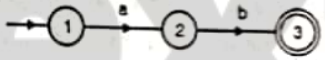


Fig. 3.3.32

Step I : Remove second (②) state.

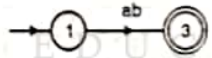


Fig. 3.3.33

RE = ab

3.3.9 Equivalence of Two Finite Automata

☞ **How to identify equivalence of two finite automata ?**

Steps to identify equivalence

1. For any pair of states $\{q_i, q_j\}$ the transition for input a $\in \Sigma$ is defined by $\{q_a, q_b\}$ where ;

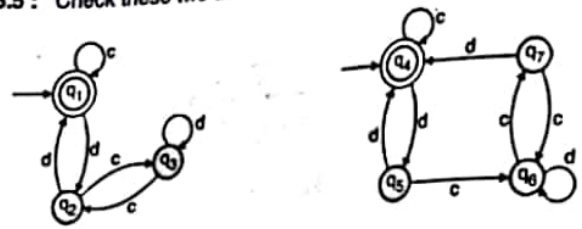
   $\delta\{q_i, a\} = q_a$ and $\delta\{q_j, a\} = q_b$

   The two automates are not equivalent if for pair $\{q_a, q_b\}$ one is intermediate state and other is final state

2. If initial state is final state of one automation, then in second automation

also, initial state must be final state for them to be equivalent.

Example 3.3.5 : Check these two automata are equivalent or not.



(i) First Automata [A] (ii) Second Automata [B]

Fig. P. 3.3.5

Solution :

Check condition I :

- For every pair of states the pair of states generates particular input showable either both should be on intermediate state or both should be on final state.

States	Input	
	C	D
(q_1, q_4)	(q_1, q_4) FS FS	(q_2, q_5) IS IS
(q_2, q_5)	(q_3, q_5) IS IS	(q_1, q_4) FS FS
(q_3, q_6)	(q_2, q_7) IS IS	(q_3, q_6) IS IS
(q_2, q_7)	(q_3, q_6) IS IS	(q1, q4) FS FS

Note FS = final state

IS = Intermediate state

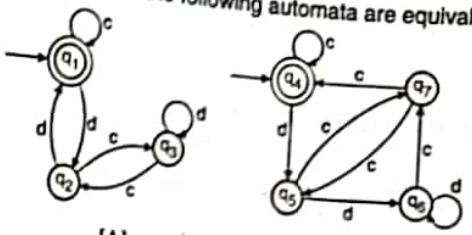
So first condition is satisfied.

Check condition II :

- Initial state and final state are same, in both automata.

- In our Example, q_1 is initial state as well as final state and in second automata q_4 is initial state as well as final state. So these two automata are equivalent.

Example 3.3.6 : Find out whether the following automata are equivalent or not :



[A] [B]

Fig. P. 3.3.6

Soln. :

In this example, second condition is satisfied because q_1 is initial and final state in A Automata and q_4 is initial state and final state in B Automata.

Check condition I :

	Inputs	
State	c	d
(q_1, q_4)	(q_1, q_4) FS FS	(q_2, q_5) IS, IS
(q_2, q_5)	(q_3, q_7) IS, IS	(q_1, q_5) FS, IS

A and B are not equivalent.

Equivalence of two regular expressions

Write methods for checking equivalence of regular expression ?

There are two methods for checking equivalence of given regular expression,

1. Simplification of regular expression by using indentities.
2. Conversion of regular expression into finite automata.

Solved examples on Equivalence of two regular expressions

Example 3.3.7 : $(a + b)^* = a^* (ba^*)^*$

Check the equivalence of given regular expression by Converting of regular expression into finite automata.

Solution :

Method II

Step I : Construct finite automata for $(a + b)^*$

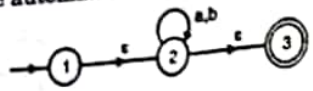
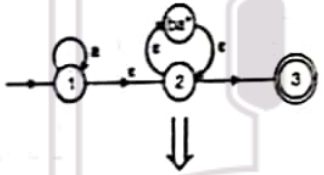


Fig. P. 3.3.7(a)

After removing ε moves we get the diagram,

Fig. P. 3.3.7(b)

Step II : Transition diagram for $a^* (ba^*)^*$



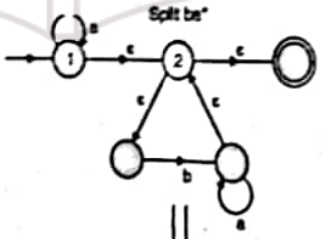


Fig. P. 3.3.7(c)

Hence prove that given regular expression are equal.

Syllabus Topic : Pumping Lemma and its Application

3.4 Pumping Lemma and its Application

Write a note on pumping lemma for regular expression ?

3.4.1 Pumping Lemma for Regular Expression

- Pumping lemma is used to prove that a language is NOT regular.
- It cannot be used to prove that a language is regular.
- If A is regular language, then A has a pumping length 'p' such that any string 's',

where $|s| \geq p$ may be divided into 3 parts.

$s = xyz$ such that the following conditions most be true-

(i) $xy^i z \in A$ for every $i \geq 0$

(ii) $|y| > 0$

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

To prove that a language is not regular using pumping lemma, follow the below steps,

(i) Assume that A is regular.

(ii) It has to have a pumping length (say p).

(iii) All strings longer than p can be pumped $|s| \geq p$.

(iv) Now find a string 's' in A such that $|s| \geq p$.

(v) Divide s into x, y, z.

(vi) Show that $xy^i z \in A$ for some i.

(vii) Then consider all ways that s can be divided into xyz.

(viii) Show that none of these can satisfy all the 3 pumping conditions at the same time.

(ix) S cannot be pumped = contradiction means language is not regular.

Example 3.4.1 : Using pumping lemma, prove that the language,

$A = \{a^n b^n \mid n \geq 0\}$ is not regular.

Solution :

Proof :

- Assume that A is regular.
- Pumping length = p.

$$s = a^p b^p$$

- Divide it into 3 parts x, y, z.
- Assume that p = 7.
- So $s \Rightarrow$ aaaaaaabbbbbbb

Condition I :

Case 1 :

The y is in 'a' part.

a aa a a aa b b b b b b b

x y z

Case 2 :

The y is in 'b' part.

a a a a a a a b bb b b bb

x y z

case y^2 $|xy| \leq p$

Case 3 :

The Y is in the 'a' and 'b' Part.

aaaaa aabb bbbbb

X Y Z

For case 1 :

$x y^i z \Rightarrow x y^2 z$

in case 1 → x = aa

y = aaaa

z = abbbbbbb

$\therefore$ $x y^2 z \Rightarrow$ aaaaaaaaaa bbbbbbb

In this string number of a = 11 and b = 7.

$11 \neq 7$

So this string does not lie in our language.

For case 2 :

$x y^i z \Rightarrow x y^2 z$

in case 2 → x = aaaaaaabb

y = bbbb

z = b

$\therefore$ $x y^2 z \Rightarrow$ aaaaaaabbbbbbbbbbb

$7 \neq 11$

So this string does not lie in our language .

For case 3 :

$x y^i z \Rightarrow x y^2 z$

in case 3 → x = aaaaa

y = aabb

z = bbbbb

$\therefore$ $x y^2 z \Rightarrow$ aaaaaaaabbaabbbbbbb

This string does not follow the $a^n b^n$ pattern, so these strings do not lie in our language.

Condition 3

$|xy| \leq p$

where, $p = 7$

Case 1 :

$|x| = 2, \quad |y| = 4$

$\therefore$ $|xy| = 2 + 4 = 6$ ...satisfied

$6 \leq 7$

Case 2 :

$|x| = 9, \quad |y| = 4$

$\therefore$ $|xy| = 9 + 4 = 13$ ...not satisfied

$13 \nleq 7$

Case 3 :

$|x| = 5, \quad |y| = 4$

$\therefore$ $|xy| = 5 + 4 = 9$ ...not satisfied

$9 \nleq 7$

So we can say our language A is not regular.

Example 3.4.2 : Using pumping Lemma prove that the language, $A = \{ y\,y \mid y \in (0, 1)^* \}$ is not regular.

Solution :

Consider,

$\underbrace{01}_{y}\ \underbrace{01}_{y}$

Proof

- Assume that A is a regular.
- Then it must have a pumping length = P
- Choose string S;

$S = \underline{0^P 1 0^P 1}$

- Divide S into three parts X, Y, Z
- Consider P = 7
- String, S = 0000000100000001
- Divide this string into X, Y, Z

$\underbrace{00}_{X}\ \underbrace{0000}_{Y}\ \underbrace{0100000001}_{Z}$

Condition I

- Showing string in form of $X Y^i Z$

Let, take $i = 2$

$\therefore$ $X Y^i Z \Rightarrow X Y^2 Z$

- In our case,

$X = 00, \ Y = 0000, \ Z = 0100000001$

$X Y^2 Z = 000000000001000000001$

This string does not lie in language A.

Condition II

$|Y| > 0$

$4 > 0$ ...satisfied

Condition III

$|XY| \leq P$

Where $P = 7, \ X = 02, Y = 04$

$|XY| = 02 + 04 = 06$

$|XY| \leq 7$

$06 \leq 7$ ...satisfied

But condition is not satisfied so our language is not a regular.

Syllabus Topic : Closure Properties

3.5 Closure Properties of Regular Sets

3.5.1 Regular set

Any set that represents the value of regular expression is called **regular set.**

Properties of regular sets :

1.	Union	2.	Intersection
3.	Complement	4.	Difference
5.	Reversal	6.	Closure of regular
7.	Concatenation		

Property 1 : Union of two regular set is regular.

Proof :

Lets consider two regular expressions.

$R_1 = a(aa)^*$

$R_2 = (aa)^*$

For R_1 :

$L(R_1) = \{a, aaa, aaaaa, ...\}$

$= \{$ Strings of odd length excluding Null $\}$

For R_2 :

$L(R_2) = \{\varepsilon, aa, aaaa, aaaaaa, ...\}$

$= \{$ Strings of even length including null$\}$

$\therefore L(R_1) \cup L(R_2) = \{\varepsilon, a, aa, aaa, aaaa, ...\}$

$\therefore RE = (L(R_1) \cup L(R_2)) = a^*$ which is regular expression itself.

Property 2 : Intersection of two regular sets is regular.

Proof : $R_1 = a(a^*)$, $R_2 = (aa)^*$

So,

$L(R_1) = \{a, aa, aaa, ...\}$

$L(R_2) = \{\varepsilon, aa, aaaa, aaaaaa, ...\}$

$\therefore L(R_1) \cap L(R_2) = \{aa, aaaa, aaaaaa, ...\}$

$\therefore RE(L(R_1) \cap L(R_2)) = aa(aa)^*$ which is a regular expression itself.

Property 3 : Complement of regular set is regular

Proof :

$R_1 = (aa)^*$

so $L(R_1) = \{\varepsilon, aa, aaaa, aaaaaa, ...\}$

Complement of $L(R_1)$ is all strings that is not in L

$L'(R_1) = \{a, aaa, aaaaa, ...\}$

$\therefore RE\ L' = a(aa)^*$ which is regular expression itself

Property 4 : The difference of two regular set is regular.

Proof :

$R_1 = a(a)^*$, $R_2 = (aa)^*$

$\therefore L(R_1) = \{a, aa, aaa, aaaa, ...\}$

$L(R_2) = \{\varepsilon, aa, aaaa, ...\}$

$L(R_1) - L(R_2) = \{a, aaa, aaaaa, ...\}$

$\therefore RE(L(R_1) - L(R_2)) = a(aa)^*$ which is regular expression itself

Property 5 : The reversal of regular set is regular

Proof :

$\therefore L(R_1) = \{01, 10, 11, 10\}$

$RE(L(R_1)) = 01 + 10 + 11 + 10$

$\therefore L^R = \{10, 01, 11, 01\}$

$RE(L^R) = 01 + 10 + 11 + 10$ which is regular expression

Property 6 : Closure of regular set is regular

Proof :

$R_1 = a(aa)^*$

$\therefore L(R_1) = \{a, aaa, aaaaa, ...\}$

$\therefore L^* = \{a, aa, aaa, aaaa, ...\}$

$\therefore RE(L^*) = a(a^*)$ which is regular expression

Property 7 : Concatenation of two regular sets is regular

Proof :

$R_1 = (0+1)^* 0$, $R_2 = 01(0+1)^*$

$\therefore$ $L(R_1) = \{0, 00, 10, 000, 010, \ldots\}$ all string ending in 0

$\therefore$ $L(R_2) = \{01, 010, 011, \ldots\}$ all string beginning with 01

then after concatenation of these two strings we get,

{ 001, 0010, 0011, 0001, 00010, 00011, ... }

Set of string containing 001 as a substring

Which can be represented by,

$$RE = (0+1)^* 001 (0+1)^*$$

Syllabus Topic : Regular Sets and Regular Grammars

3.6 Regular Sets and Regular Grammars

- In this section, we show that class of regular sets over Σ is precisely the regular languages over the terminal set Σ.
- Construction of regular grammar generating T(M) for a given DFA M.

Let, $M = (\{q_0, \ldots\ldots q_n\}, \Sigma, \delta, q_0, F)$

If w is in T(M), then it is obtained by concatenating labels corresponding to several transitions, the first from q_0 and the last terminating at some final state. So for grammar G to be constructed, productions should correspond to transition.

Now construct the Grammar ;

$$G = (\{A_0, A_1, \ldots\ldots, A_n\}, \Sigma, P, A_0)$$

Where production P :

(1) $A_i \rightarrow aA_j$ is included in P,

if $\delta(q_i, a) = A_j \in F$.

(2) $A_i \rightarrow a A_j$ and $Ai \rightarrow a$ are included in P

if $\delta(q_i, a) = q_j \in F$.

We can show that L(G) = T(M) by string construction of P, such a construction gives,

$A_i \Rightarrow aA_j$ if $\delta(q_i, a) = q_j$

$A_i \Rightarrow a$ if $\delta(q_i, a) \in F$

So,

$A_0 \Rightarrow a_1 A_1 \Rightarrow a_1 a_2 A_2 \Rightarrow \ldots\ldots \Rightarrow a_1, \ldots\ldots, a_{k-1} A_k \Rightarrow a_1 a_2 \ldots\ldots a_k$

If $\delta(q_0, a_1) = q_1$

$\delta(q_1, a_2) = q_2$

:

:

$\delta(q_k, a_k) \in f$

This proves that $w = a_1 \ldots\ldots a_k \in L(G)$

If $\delta(q_0, a_1 \ldots\ldots a_k) \in F$ i.e. if $w \in T(M)$

For example,

1. **Construct regular grammar G generating the regular set represent by**

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

Step I : Construct finite automata of given regular expression

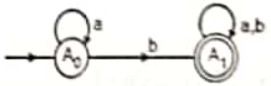


Fig. 3.6.1

Step II : Convert this finite automata into regular grammar

Grammar consists of non-terminal, terminal, productions and initial symbol.

$G = (N, T, P, S) = (\{A_0, A_1\}, \{a, b\}, P, A_0)$

Non-terminal = $\{A_0, A_1\}$

Terminals = $\{a\ b\}$

Initial symbol = A_0

Production P : $A_0 \rightarrow a A_0$, $A_0 \rightarrow bA_1$, $A_0 \rightarrow b$

$A_1 \rightarrow b A_1$, $A_1 \rightarrow a A_1$, $A_1 \rightarrow a$, $A_1 \rightarrow b$

2. Construct regular grammar G at given finite

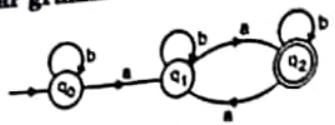


Fig. 3.6.2

$G = (\{q_0, q_1, q_2\}, \{a, b\}, P, q_0)$

Production P :

$q_0 \rightarrow b\, q_0$, $q_1 \rightarrow a\, q_2$, $q_2 \rightarrow \in / b$

$q_0 \rightarrow a\, q_1$, $q_2 \rightarrow b\, q_2$, $q_1 \rightarrow a$

$q_1 \rightarrow b\, q_1$, $q_2 \rightarrow a\, q_1$

3.6.1 Construction of Transition System

Construction of transition system M accepting L(G) for given regular grammar G.

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

   We construct transition system M whose,

1. States correspond to variable
2. Initial state corresponds to A_0.
3. Transition corresponds to production P.
4. Last production applied in any derivation is of the form $A_j \rightarrow a$.
5. Transition terminals at final state.

   M is defined as, $(\{q_0, q_1, \ldots\ldots, q_f\}, \Sigma, \delta, q_0, \{q_f\})$

   Where δ is transition function is defined as,

1. Each production $A_i \rightarrow aA_j$ induces a transition from q_i to q_f with label a
2. Each production $A_k \rightarrow a$ induces a transition from q_k to q_f with label a

Suppose,

$A_0 \Rightarrow a_1 A_1 \Rightarrow a_1 a_2 A_2 \Rightarrow \ldots\ldots$

$\Rightarrow a_1 \ldots\ldots, a_{n-1} A_{n-1} \Rightarrow a_1 \ldots\ldots a_n$ is a derivation of $a_1 a_2, \ldots\ldots$ an. If there is a path in M starting from q_0 and terminating q_f with path value $a_1 a_2 \ldots\ldots a_n$

$\therefore$ L (G) = T(m)

Example 3.6.1 : $G = (\{A_0, A_1\}, \{a, b\}, P, A_0)$ when

P : $A_0 \rightarrow a A_1$

$A_1 \rightarrow b A_1$

$A_1 \rightarrow a$

$A_1 \rightarrow b A_0$

Construct a transition system M accepting L(G)

Solution :

Lets consider,

$M = (\{q_0 q_1 q_f\}, \{a, b\}, \delta, q_0, \{q_f\})$

when q_0 and q_1 correspond to A_0 and A_1 respectively and q_f is a final state introduced,

$A_0 \rightarrow a A_1$ .... induces a transition from q_0 to q_1 with label a

$A_1 \rightarrow b A_1$ .... induces a transition from q_1 to q_1 with label b

$A_1 \rightarrow b A_0$ .... induces a transition from q_1 to q_0 with label b

$A_1 \rightarrow a$ .... induces a transition from q_1 to q_f with label a

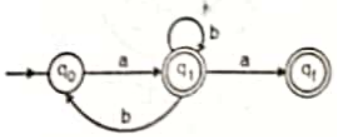


Fig. P. 3.6.1

Example 3.6.2 : If regular grammar G is given by,

$S \rightarrow as \mid a$ find M accepting L(G)

Solution :

Let q_0 correspond to S and q_f is final state.

$$M = (\{q_0, q_f\}, \{a\}, \delta, q_0, \{q_f\})$$

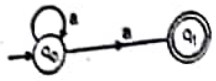


Fig. P. 3.6.2

Exercise

Q. 1. Construct a regular expression corresponding to the state diagram described by Fig. 1

Fig. 1

Q. 2. Find regular expression :

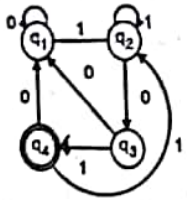


Fig. 2

Q. 3 Construct DFA with reduced state equivalent to regular expression,

$10 + (0 + 11)\,0^*1$.

□□□

CHAPTER 4

Context Free Languages

University Prescribed Syllabus

Context Free Languages : Context-free Languages, Derivation Tree, Ambiguity of Grammar, CFG simplification, Normal Forms, Pumping Lemma for CFG.

Syllabus Topic : Context Free Languages

4.1 Context Free Languages

Write a note on context free languages.

- A grammar is a set of rules for putting strings together and so corresponds to a language.
- A type 2 grammar is called as **context-free grammar** (as A can be replaced by α in any context).
- A language generated by context free grammar is called **a type 2 language** or **a context free language.**

Definition of Context Free Grammar (CFG)

Context Free Grammar consisting a finite set of grammar rules is a quadruple (N, Σ, P, S) where,

N = A set of non-terminal symbols.

Σ = A set of terminals; where, $N \cap T$ = NULL (Set N intersection set T is always equal to NULL)

P = Set of production rules.

P : N (NUT)*, i.e. left hand side of production rule P, does have any right context or left context.

S = A starting symbol.

Note : For non-terminals use uppercase letters.

For example

(1)

Start Terminal

↓ ↓

S → as → Non-terminal

S → ε

The grammar : ({A}, {a, b, c}, P,)

Compare this to quadruple (N, T, P, S)

Here,

N → {A}

T → {a, b, c}

∴ P : A → aA

A → a b c

(2) The grammar : {{S, F}, {0, 1}, P, S}

P :

S → 00S | 11F

F → 00F | ε

(3) Construct a context free Grammar G generating all integers (with sign).

∴ G = (N, T, P, S)

N = {S, (sign), (digit), (integer)}

T = {0, 1, 2, ..., 9, +, – }

P consists of S → < sign> <integer>

<sign> → + 1 –

<integer> → <digit> <integer> | <digit>

<digit> → 0|1|2 . . . . |9

L(G) = Set of all integer

e.g. Derivation of – 17 can be obtained as follows :

S ⇒ <sign> <integer>

⇒ – <integer>

⇒ – <digit> <integer>

⇒ – 1 <digit>

⇒ – 17

(4) Production

S → as

S → ε

Derivation for aaaa is,

S ⇒ as

⇒ aas

⇒ aaas

⇒ aaaas

⇒ aaaas

= aaaa

(5) Production :

S → SS

S → a

S → ε

Derivation of aa :

$S \Rightarrow SS$

$\Rightarrow SSS$

$\Rightarrow SSa$

$\Rightarrow SSSa$

$\Rightarrow SASA$

$\Rightarrow \varepsilon ASA$

$\Rightarrow \varepsilon a \varepsilon a$

$\Rightarrow aa$

Syllabus Topic : Derivation Tree

4.1.1 Derivation Tree

What is a derivation tree ? Explain it with example.

- Derivation can be represented using tree. Such trees are called **derivation trees.**
- Derivation tree is also called as parse tree.
- Derivation tree is an ordered rooted tree that graphically represents the semantic information of a string derived from a context free grammar.

Representation

- **Root vertex :** Start symbol indicates root vertex.
- **Vertex :** All non-terminal symbols are vertices.
- **Leaves :** All terminal symbols or ε are leaves.

For example

If $S \rightarrow x_1 x_2 \ldots . x_n$ is a production rule in CFG, then parse tree will be as shown in Fig. 4.1.2 .

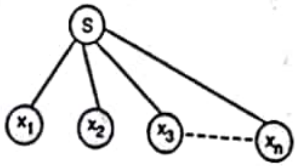


Fig. 4.1.2

Definition

For CFG, G = (N, Σ, P, S) is a **derivation tree** satisfying following condition :

1. Each interior node is labeled by variable in N.
2. The root has label S.
3. Each vertex is labeled by either variable, terminal or ε.
4. A leaf labeled by ε must be the only child of parent.
5. If interior node labeled by A with children labeled by $x_1, x_2, \ldots, x_n$ (from left), then $A \rightarrow x_1, x_2 \ldots . x_n$ must be rule.

Example 4.1.1 : Let G = ({S, A}, {a, b}, P, S)

where, P consists of :

$S \rightarrow aAS \mid a \mid SS$

$A \rightarrow SbA \mid ba$

Draw a derivation tree.

Solution :

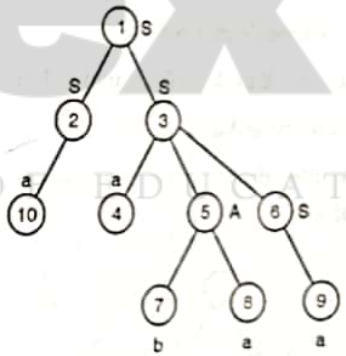


Fig. P. 4.1.1

Explanation

- Vertex S is a root vertex.
- Vertices 4 to 6 are child of 3, written from left using S → aAS production in P.
- Vertices 7 and 8 are child of 5 written from left using A → ba production in P.
- Vertex 5 is internal vertex labeled as A, which is a variable.

Ordering of leaves from left

1. Successors of root (means child of root) are ordered from left by definition.
2. Vertices at level 1 are ordered from left, if v_1 and v_2 are any two vertices at level 1 and v_1 is to the left of v_2, then we can say v_1 is to the left of any child of v_2.
3. Any child of v_1 is to left of v_2 and to the left of any child of v_2. Thus, we get left to right ordering of vertices at level 2.

- Repeating the process upto level n, where n is height of tree.
- Note that ordering of all vertices is from left.

E.g. See the ordering of vertices shown in Fig. 4.1.3

1. Child of root 1 are 2 and 3 order from left
2. Child of 2 namely 10 is a left of any child of 3.
3. The child of 3 ordered from 4-5-6.
4. The vertices at level 2 in the left to right ordering are 10-4-5-6.
5. The vertex 4 is to the left of 6.
6. The child of 5 ordered from left are 7-8, so 4 is to the left of 7. Similarly 8 is to the left of 9.

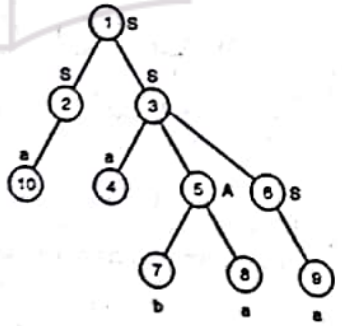


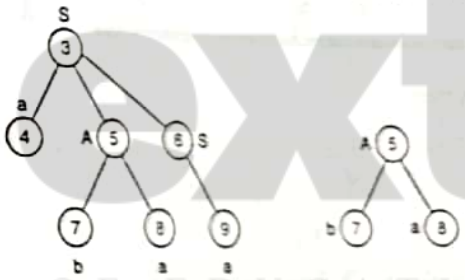
Fig. 4.1.3

- Thus, order from left is 10-4-7-8-9. If we are ordering left to right, direction is always anticlockwise direction.
- Result of derivation tree is concatenation of the label of leaves without repetition in left to right ordering.
- For Fig. 4.1.3, the result of derivation tree is aabaa.

Definition

A subtree of derivation tree 'T' is a tree,

- Whose root is some vertex v of T.
- Whose vertices are descendants of vertex v together with their labels.
- Whose edges are those connecting the descendants of vertex V.



(a) Subtree (b) 'A' subtree

Fig. 4.1.4

Example 4.1.2 : Consider the grammar G whose productions are

$S \rightarrow aASla$

$A \rightarrow SbA|SS|ba$

Show that $S \Rightarrow aabbaa$ and construct a derivation tree.

Solution :

$S \Rightarrow a\underline{A}S$ applying $A \rightarrow SbA$

$\Rightarrow a\underline{S}bAS$ applying $S \rightarrow a$

$\Rightarrow aab\underline{A}S$ applying $A \rightarrow ba$

$\Rightarrow aabba\underline{S}$ applying $S \rightarrow a$

$\Rightarrow aabbaa$

Derivation tree :

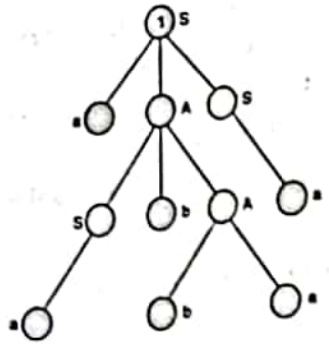


Fig. P. 4.1.2

Derivation trees can be represented in two ways :

A.	Left derivation tree	B.	Right derivation tree

A. Left derivation tree

Left derivation tree is obtained by applying production to the leftmost variable in each step.

For example,

For generating string aabaa from grammar

$S \rightarrow aAS \mid aSS \mid \varepsilon$

$A \rightarrow SbA \mid ba$

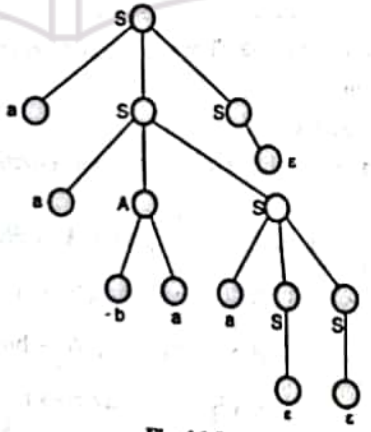


Fig. 4.1.5

B. Right derivation tree

A right derivation tree is obtained by applying production to the right most variable in each step :

For example,

For generating string aabaa from grammar

$S \rightarrow aAS \mid aSS \mid \varepsilon$

$A \rightarrow SbA \mid ba$

Fig. 4.1.6

Example 4.1.3 : For generating string 00110101 form grammar,

$S \rightarrow 0B \mid 1A$

$A \rightarrow 0 \mid 0S \mid 1AA$

$B \rightarrow 1 \mid 1s \mid 0BB$

Find (i) Left most derivation (ii) Right most derivation

Solution :

Leftmost derivation

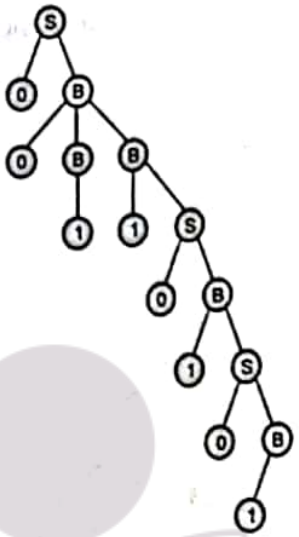


Fig. P. 4.1.3(a)

Rightmost derivation

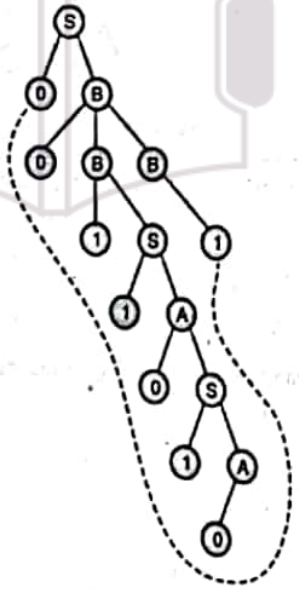


Fig. P. 4.1.3(b)

Syllabus Topic : Ambiguity of Grammar

4.2 Ambiguity in Context Free Grammars

- A CFG is **ambiguous** if one or more terminal strings have multiple left most derivations from the start symbol.
- Deterministic context free grammars are always unambiguous.

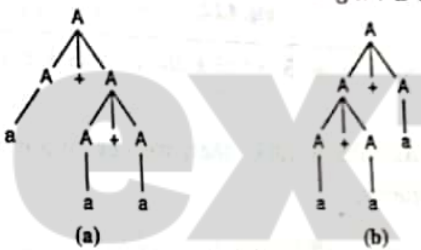
Examples

(i) $A \rightarrow \varepsilon$

$B \rightarrow \varepsilon$

(ii) $A \rightarrow A \mid \varepsilon$

(iii) $A \rightarrow A + A \mid A - A \mid a$ is ambiguous for string a + a + a.



(a) (b)

Fig. 4.2.1

For example

G = {{S}, {a + b, + , *}, P, S} where,

P consists of ;

$S \rightarrow S + S$

$S \rightarrow S * S$

$S \rightarrow a \mid b$

The string a + a * b can be generated as,

Ist way	IInd way
$S \rightarrow S + S$	$S \rightarrow \underline{S} * S$
$\rightarrow a + S$	$\rightarrow S + S * S$
$\rightarrow a + S * S$	$\rightarrow a + S * S$

Ist way	IInd way
→ a + a * S	→ a + a * S
→ a + a * b	→ a + a * b

Thus the grammar is ambiguous grammar.

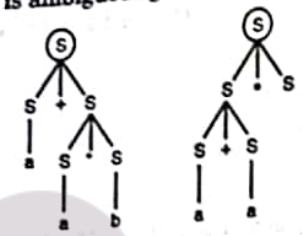


Fig. 4.2.2

Example 4.2.1 : If G is the grammar, S → SbSIa, show that G is ambiguous.

Solution :

Consider string abababa ∈ L(G), then we get two derivation trees for string. Thus G is ambiguous,

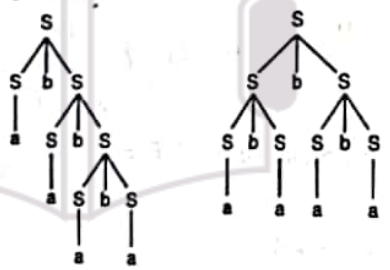


Fig. P. 4.2.1 : Two derivation trees of abababa

exam

Syllabus Topic : CFG Simplification

4.3 Simplification of Context Free Grammars

Write a note on CFG simplification.

- Simplification of CFG means reduction of CFG.
- In CFG, sometimes all production rules and symbols are not needed for the derivation of strings.

- Besides this, there may also be some NULL productions and UNIT production.
- Elimination of these productions and symbols is called simplification of CFG.
- Simplification consists of the following steps :

1. Reduction of CFG
2. Removal of unit productions
3. Removal of Null production

4.3.1 Reduction of CFG

Phase 1 : Derivation of an equivalent grammar G′ from CFG, G such that each variable derives some terminal strings.

Derivation procedure

Step I : Include all symbols w_1 that derives some terminal and initialize i = 1.

Step II : Include symbols w_{i+1} that derives w_i.

Step III : Increment i and repeat step 2 until $w_{i+1} = w_i$.

Step IV : Include all production rules that have w_i in it.

Phase 2 : Derivation of equivalent grammar G″ from the CFG, G′ such that each symbol appears in a sentential form.

Derivation procedure

Step I : Include the start symbol in y1 and initialize i = 1.

Step II : Include all symbols yi + 1, that can be derived from yi and include all production rules that have been applied.

Step III : Increment i and repeat step 2, unit $y_{i+1} = y_i$.

By using phase 1 and phase 2, we obtain reduced CFG

Solved Examples

Example 4.3.1 : Find a reduced grammar equivalent to the grammar G, having production rules.

$P : S \rightarrow AclB.\ A \rightarrow a,\ C \rightarrow clBc,\ E \rightarrow aAle$

Solution :

Lets proceed with phase I

Terminal symbols : $-T = \{a, c, e\}$

Set w_1 includes all symbols which are derived from terminal symbols :

$$w_1 = \{A, C, E\}$$

Set w_2 includes all symbols that can derived the symbols present in w_1

$$w_2 = \{A, C, E, S\}$$

Set w_3 includes all symbols that can derived the symbols present in w_2

$$w_3 = \{A, C, E, S\}$$

New Grammar $G' = \{(A, C, E, S), \{a, c, e\}, P, (S)\}$

G = {NT, T, P, S}

P = Production rule

NT = Non-terminal

T = Terminal

S = Start symbol

$P : S \rightarrow AC,\ A \rightarrow a,\ C \rightarrow c\ E \rightarrow aA/e$

Obtain phase II

1. y_1 includes start symbol :

$$y_1 = \{S\}$$

2. y_2 includes all symbols that can be derived from start symbols.

$$y_2 = \{S, A, C\}$$

3. y_3 includes all symbols that can be derived from y_2.

$$y_3 = \{S, A, C, a, c\}$$

$$y_4 = \{S, A, C, a, c\}$$

$$a'' = \{(A, C, S), \{a, c\}, P, \{S\}\}$$

$$P : S \rightarrow AC,\ A \rightarrow a,\ C \rightarrow c$$

Example 4.3.2 : Let $G = (V_n, \Sigma, P, S)$ be given by the production $S \rightarrow AB,\ A \rightarrow a,\ B \rightarrow b,\ B \rightarrow C,\ E \rightarrow c.$ Find G′ such that every variable in G′ derives some terminal string.

Solution :

First obtain phase I

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

$$w_1 = \{A, B, E\}$$

$$w_2 = \{A, B, E, S\}$$

$$w_3 = \{A, B, E, S\}$$

$$G' = \{(A, B, E, S), (a, b, c), P, S\}$$

$$P : S \rightarrow AB, A \rightarrow a, B \rightarrow b, E \rightarrow c$$

Phase II

$$y_1 = \{S\}$$

$$y_2 = \{S, A, B\}$$

$$y_3 = \{S, A, B, a, b\}$$

$$y_4 = \{S, A, B, a, b\}$$

$$G'' = \{(S, A, B), \{a, b\}, P(S)\}$$

$$P : S \rightarrow AB, A \rightarrow a, B \rightarrow b$$

Thus, we obtained new production rule P.

Example 4.3.3 : Find reduced grammar equivalent to the grammar G whose productions are, $S \rightarrow AB \mid CA$, $B \rightarrow BC \mid AB$, $A \rightarrow a$, $C \rightarrow aB \mid b$

Solution :

Phase I

Let proceed with phase I.

Terminals symbols : T = {a, b}

Set w_1 includes all symbol which are derived from terminal symbols.

$$w_1 = \{A, C\}$$

Set w_2 includes all symbols derived from the symbols present in w_1

$$w_2 = \{A, C, S\}$$

Set w_3 includes all symbols that are derived from the symbols present in w_2,

$$w_3 = \{A, C, S\}$$

Now grammar G′ = {(A, C, S), (a, b), P, (S)}

Grammar, G = {NT, T, P, S}

New production rule

P : S → AC, A → a, C → b

Phase 2

$$y_1 = \{S\}$$

$$y_2 = \{S, A, C\}$$

$$y_3 = \{S, A, C, a, b\}$$

$$y_4 = \{S, A, C, a, b\}$$

$$G'' = \{(A, C, S), \{a, b\} P, S\}$$

P : S → AC, A → a, C → b

4.3.2 Removal of Unit Productions

Any production rule of the form $A \rightarrow B$ where, A, B, $\in$ non-terminals is called **unit production.**

Procedure for Removal

Step I : To remove $A \rightarrow B$; add production $A \rightarrow X$ to the grammar rule whenever

$B \rightarrow X$ occurs in the grammar [$X \in$ Terminal × can be uni]

Step II : Delete $A \rightarrow B$ from the grammar.

Step III : Repeat from step I until all unit productions are removed.

Example 4.3.3 : Remove unit productions from the grammar whose production rule is given by,

$P : S \rightarrow XY, X \rightarrow a, Y \rightarrow Zb, Z \rightarrow m, M \rightarrow N, N \rightarrow a$

Solution :

Unit productions in our grammar are :

$Y \rightarrow Z, Z \rightarrow M, M \rightarrow N$ (all are unit terminals)

- Since, $N \rightarrow a$, we add $M \rightarrow a$

  $P : S \rightarrow XY, X \rightarrow a, Y \rightarrow z \mid b \rightarrow Z \rightarrow M,$

  $M \rightarrow a, N \rightarrow a$

- Since, $M \rightarrow a$, we add $Z \rightarrow a$

  $P : S \rightarrow XY, X \rightarrow a, Y \rightarrow Z \mid b, Z \rightarrow a$

  $M \rightarrow a, N \rightarrow a$

- Since, $Y \rightarrow Z \mid b$, we add $Y \rightarrow a$

  $P : S \rightarrow XY, X \rightarrow a, Y \rightarrow a \mid b, Z \rightarrow a$

  $M \rightarrow a, N \rightarrow a$

Here, Z, M and N are unreachable symbols because start symbol $S \rightarrow XY$, from S we get X and Y.

So remove the unreachable symbols,

$P : S \rightarrow XY, X \rightarrow a, Y \rightarrow a \mid b$

This is our final production rule.

4.3.3 Removal of NULL Production

In CFG, non-terminal symbol 'A' is a nullable variable, if there is a production, $A \to \varepsilon$ or there is derivation that starts at 'A' and lead to ε (like A, ... → E).

Procedure for Removal

Step I : To remove $A \to \varepsilon$, look for all productions whose right side contains A.

Step II : Replace each occurrences of 'A' in each of their productions with ε.

Step III : Add the resultant productions to the grammar.

Example 4.3.4 : Remove NULL production from the following grammar :
$S \to ABAC, A \to aA|\varepsilon, B \to bB|\varepsilon, C \to c$

Solution :

First find out NULL production, $A \to \varepsilon$, $B \to \varepsilon$,

these two null productions we have to remove.

(a) To eliminate $A \to \varepsilon$

(i) $S \to ABAC$

(ii) Replace each occurrences of A in each of these production with ε.

$S \to ABAC$

$S \to ABC|BAC|BC$

$A \to aA$

$A \to a$

New production :

P : $S \to ABAC \mid ABC \mid BAC \mid BC$

$A \to aA \mid a$

$B \to bB|\varepsilon$

$C \to c$

(b) To eliminate $B \to \varepsilon$,

We have to check new production where we find B on right side of any production that replace with ε.

$S \to ABAC \mid ABC \mid BAC \mid BC$

Replace B with ε

We get,

$S \to AAC \mid AC \mid C$

$B \to b$

New production :

$S \to ABAC \mid ABC \mid BAC \mid BC \mid AAC \mid AC \mid C$

$A \to aA|a$

$B \to bB|b$

$C \to c$

This is our result.

Example 4.3.5 : Consider the grammar G where production are $S \to AS \mid AB$, $A \to \varepsilon$, $B \to \varepsilon$, $D \to b$. Construct a grammar G_1 without null productions generating.

Solution :

First find out NULL production,

$A \to \varepsilon$, $B \to \varepsilon$

(a) To eliminate $A \to \varepsilon$

$S \to aS \mid AB$

Replace A in each of these production with ε.

$\therefore S \to aS \mid AB$

$\to aS \mid B$

New production

P : $S \to aS \mid AB \mid B$

$B \rightarrow \varepsilon$

$D \rightarrow b$

(b) To eliminate $B \rightarrow \varepsilon$

Replace B in each of these productions with ε

$S \rightarrow aS \mid AB$

$S \rightarrow aS \mid A$

New production :

$P : S \rightarrow aS \mid AB \mid A$

$D \rightarrow b$

$P : S \rightarrow aS, S \rightarrow AB; S \rightarrow A, D \rightarrow b$

Syllabus Topic : Normal Forms

4.4 Normal Forms

☞ **Write a note on Normal forms.**

- The process of removal of duplication of production is called **normalized CFG.**
- In CFG, the RHS of a production can be any string of variable and terminals.
- When production in grammar G satisfies certain restriction, then G is said to be a normal forms.
- There are two normal forms :
  - (i) Chomsky Normal Form (CNF)
  - (ii) Greibach Normal Form (GNF)

4.4.1 Chomsky Normal Form(CNF)

- In Chomsky Normal Form (CNF), we have restriction on the length of RHS, which is elements in RHS should either be two variables or terminal.

A CFG is in Chomstry Normal Form if the productions are in the following forms :

$A \rightarrow a$

$A \rightarrow BC$

Where A, B and C are Non-terminals and A is a terminal.

Convert a given CFG to Chomsky Normal Form

Step I : If the start symbol S occurs on some right side, create a new start symbol S′ and new production.

$S' \rightarrow S$

Step 2 : Remove NULL productions

(we already discussed in previous section)

Step 3 : Remove unit production

(we already discussed in previous section)

Step 4 : Replace each production $A \rightarrow B_1 \ldots . B_n$ where $n > 2$ with $A \rightarrow B_1 C$ where $C \rightarrow B_2 \ldots . B_n$

Repeat this step for all productions having two or more symbols on right side.

Step 5 : If right side of any production is in the form $A \rightarrow aB$, where a is 'a' terminal and A and B are non terminals, then the production is replaced by $A \rightarrow XB$ and $X \rightarrow a$

Repeat this step for every production which is of the form $A \rightarrow aB$.

For example

Convert the following CFG to CNF :

$P : S \rightarrow AS\,A \mid aB, \quad A \rightarrow B \mid S, \quad B \rightarrow b \mid \varepsilon$

Step 1 : Start symbol S occurs on right side in production

$A \rightarrow B \mid S$ and $S \rightarrow ASA$

We add a new state S′ and $S' \rightarrow S$ is added to production.

$P : S' \to S,\ S \to ASA|aB,\ A \to B|S,\ B \to b|\varepsilon$

Step 2 : Remove NULL production :

$B \to \varepsilon, A \to \varepsilon$

After removing $B \to \varepsilon$,

$P : S' \to S, S \to AS\,A \mid aB \mid a,\ A \to B|S|\varepsilon, B \to b$

After removing $A \to \varepsilon$,

$P : S' \to S, S \to AS\,A \mid aB \mid a \mid AS \mid SA \mid S, A \to B|S, B \to b$

Step 3 : Remove unit production :

Unit production in our problem :

$S \to S, S' \to S, A \to B$ and $A \to S$

After removing $S \to S$,

$P : S' \to S, S \to ASA \mid aB \mid a \mid AS \mid SA$

$A \to B|S, B \to b$

After removing $S' \to S$,

$P : S' \to ASA \mid aB \mid a \mid AS \mid SA$

$S' \to ASA \mid aB \mid A \mid AS \mid SA$

$A \to B|S,\ B \to b$

After removing $A \to B$,

$P : S' \to ASA \mid aB \mid a \mid AS \mid SA$

$S \to ASA \mid aB \mid a \mid AS \mid SA$

$A \to b/S, B \to b$

After removing $A \to S$,

$P : S' \to ASA \mid aB \mid a \mid AS \mid SA$

$S \to ASA \mid aB \mid a \mid AS \mid SA$

$A \to b \mid ASA \mid aB \mid a \mid AS \mid SA$

$B \to b$

Step 4 : Now find out the productions that has more than two variables in RHS.

$S' \to ASA$, $S \to ASA$ and $A \to ASA$

After removing these we get :

SA will be replaced by X.

$S' \to AX \mid aB \mid a \mid AS \mid SA$

$S \to AX \mid aB \mid a \mid AS \mid SA$

$A \to b \mid AX \mid aB \mid a \mid AS \mid SA$

$B \to b$

$X \to SA$

Step 5 : Now change the productions

$S' \to aB$, $S \to aB$, and $A \to aB$

a is replaced with variable Y and add production $Y \to a$

Finally we get :

$P : S' \to AX \mid YB \mid a \mid AS \mid SA$

$S \to AX \mid YB \mid a \mid AS \mid SA$

$A \to b \mid AX \mid YB \mid a \mid AS \mid SA$

$B \to b$

$X \to SA$

$Y \to a$

This is our result, which is the required Chomsky Normal Form for the given CFG.

4.4.2 Greibach Normal Form(GNF)

- CFG is in GNF, if its productions are in form of :

$A \to a\alpha$

$\downarrow \quad \downarrow \to$ String variable or ε

terminals

For example

$S \rightarrow aAB \mid bBAC$

$A \rightarrow a$

$B \rightarrow b$

$C \rightarrow cC$

These all productions are in GNF

Conversion of CFG to GNF

Step 1 : Check whether the given CFG is a simplified grammar i.e. it should not have null production, unit production or useless symbol.

Step 2 : Check whether the CFG is already converted into CNF.

Step 3 : Change the names of non-terminal symbols to A_i.

For example

$S \rightarrow XA \mid BB$

$B \rightarrow b \mid SB$

$X \rightarrow b$

$A \rightarrow a$

Replace,

S with A_1

X with A_2

A with A_3

B with A_4

After replacing this, we get,

Production P :

$A_1 \rightarrow A_2A_3 \mid A_4A_4$

$A_4 \rightarrow b \mid A_1A_4$

$A_2 \rightarrow a$

$A_3 \rightarrow a$

Step 4 : Modify the rules so that non terminals are in ascending order.

If $A_i \rightarrow A_{jx}$ is a rule, then $i < j$

and ($i > j$ or $i = j$) are not acceptable.

Check production, line by line

(1) $A_1 \rightarrow A_2 A_3$

$i = 1, j = 2$

$i < j$, this production is acceptable.

(2) $A_1 \rightarrow A_4 A_4$

$i = 1, j = 4$

$i < j$ this production is acceptable.

(3) $A_4 \rightarrow A_1 A_4$

$i = 4, j = 1$

$i > j$ which is not acceptable.

So it is not in GNF, we have to convert it into GNF.

Replace value of A_1 in A_4 ($A_1 \rightarrow A_2 A_3$)

$A_4 \rightarrow b \mid A_2 A_3 A_4$

Now which $A_4 \rightarrow A_2 A_3 A_4 \mid A_4 A_4 A_4$

Here, $i = 4, j = 2$

$i > j$ which is not acceptable

so replace A_2 with b.

$A_4 \rightarrow b \mid b A_3 A_4 \mid A_4 A_4 A_4$

$\therefore A_4 \rightarrow A_4 A_4 A_4$

Here, $i = 4, j = 4$

$i = j$ in which we find left recursion.

Now remove left recursion :

when ($A_i = A_j$)

$A_4 \rightarrow A_4 A_4 A_4$

Introduce a new variable and write a whole production with z variable.

$A_4 \rightarrow b \mid b\,A_3\,A_4 \mid bZ \mid b\,A_3A_4\,Z$

$Z \rightarrow A_4\,A_4\,Z \mid A_4\,A_4$

Now grammar is :

$A_1 \rightarrow A_2\,A_3 \mid A_4\,A_4$

$A_4 \rightarrow b \mid b\,A_3\,A_4 \mid bZ \mid bZ$

$Z \rightarrow A_4A_4 \mid A_4\,A_4\,Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Repeat procedure again :

Substitute the value of A_4 and A_2

$A_1 \rightarrow bA_3 \mid b\,A_3\,A_4\,A_4 \mid b\,A_4 \mid b\,A_3\,A_4\,Z\,A_4 \mid bZ\,A_4$

$A_4 \rightarrow b\,A_3\,A_4 \mid b \mid b\,A_3\,A_4\,Z \mid bZ$

$Z \rightarrow bA_3\,A_4\,A_4 \mid bA_4 \mid bA_3\,A_4Z\,A_4 \mid bZ\,A_4 \mid bA_3\,A_4\,A_4\,Z \mid bA_4\,Z \mid bA_3\,A_4ZA_4Z \mid bZA_4Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

This is the result of our example.

Exam

Syllabus Topic : Pumping Lemma for CFG

4.5 Pumping Lemma for CFG

- In formal language theory, the pumping lemma for Context Free Language (CFL) also known as the 'Bar-Hill-Lemma'. This lemma gives a property shared by CFL's.
- The pumping lemma for context-free languages describes property that all context-free languages are guaranteed to have.
- All strings in languages must have a length of at least p, where p is called pumping length which is constant.

- The pumping lemma gives us a technique to show that certain languages are not context free.
- The process of 'pumping up' additional copies of strings is what gives the pumping lemma its name

It was first proved by Dona Scott and Michael Rabin in 1959 and rediscovered shortly by Elishamil and Micha Perles in 1961 as a simpulticaton pumping lemma for CFL.

4.5.1 Definition of Pumping Lemma for CFL

Let L be a context free language, then there is a constant n which depends only on language L. Such that there exists a string $z \in L$ and $|z| \geq n$. (length of z is $\geq n$) where,

$$z = uvwxy$$

Conditions

1. $|vwx| \leq n$
2. $|vx| \geq 1$
3. For All $i \geq 0$, $uv^iwx^iy \in L$.

Example :

Using pumping lemma, prove the following language is not CFL.

$$L = 0^n\,1^n\,2^n \mid n \geq 1$$

$$L = \{012, 001122, 000111222 \ldots\ldots\}$$

Z =	0	0	1	1	2	2
	u	v		w	x	y

Length of string n = 6

Case (i)

$$|vw\,x| \leq n$$

$$|vwx| = 2 + 1 + 1 = 4$$

$$4 \leq 6$$

Case (ii)

$$|vx| \geq 1$$

$$|vx| = 2 + 1 = 3$$

$$3 \geq 1$$

Case (iii)

$$uv^i wx^i y \in L$$

$$i = 2$$

where u = 0, v = 0 1, w = 1, x = 2 and y = 2.

Using these value make a string $uv^i wx^i y$,

which is 0 0 1 0 1 1 2 2 2

0 0 1 0 1 1 2 2 2 does not belong to our languages L.

$\therefore$ 0 0 1 0 1 1 2 2 2 $\notin$ L.

By pumping lemma, we proved that the given language is not a context free language.

Example 4.5.1 : Consider a context free grammar G with the following productions,

S → ASA | B

B → aCb | bCa

C → ACA | A

A → a | b

(i) What are the variable and terminals of G

(ii) Give 3 strings of length 7 in L (G)

Solution :

(i) Variable and terminals of G

V_N = {S, A, B, C}

Terminals = Σ = {a, b}

(ii) Strings of length 7 in given L (G)

S ⇒ AASAA ⇒ AABAA ⇒ AAaCbAA

⇒ AAaAbAA ⇒ ababbab

Exercise

Q. 1. Find reduced grammar equivalent to the grammar G whose productions are

S → AB | CA, B → BC|AB, A → a

C → aB|b

Ans. : New production rule : P : S → CA, A → a, C → b

Q. 2 Construct a reduced grammar equivalent to grammar.

S → aAa, A → sb | bCC | DaA

C → abb|DD, E → aC, D → aDA

Ans. : New production rule : P : S → aAa,

A → sb | bCC,

C → abb

Q. 3 Find the following grammar G to CNF G is,

S → aAD, A → aB | bAB, B → b, D → d

Q. 4 Find a grammar in Chomsky Normal Form equivalent to S → aAbB, A → aAa, B → bBb.

Q. 5 Reduce the following grammar to CNF

S → ASA|bA, A → B|S, B → C

Show that CFG, is ambiguous grammer with production ;

S → SS | (S) | ε

Q. 6 Find derivation tree of a * b + a * b given that a * b + a * b is in L(G), where

G is given by S → S + S | S * S

S → a|b

Q. 7 A context free grammar G has following production.

S → 0S0 | 1S1 | A

A → 2B3

B → 2B3|3 Describe the language generated by parameters.

Q. 8 Consider the following productions :

$S \rightarrow aB \mid bA$

$A \rightarrow as \mid bAAla$

$B \rightarrow bs \mid aBB \mid b$

For the string aaabbabbba find

1. left most derivation
2. right most derivation
3. parse tree

Q. 9 Show that grammar is ambiguous

G : $S \rightarrow alabSblaAb$

$A \rightarrow bSlaAAb$

Q. 10 Show that grammar is ambiguous

G : $S \rightarrow aBlab$

$A \rightarrow aAB \mid a$

$B \rightarrow ABblb$

□□□

CHAPTER 5

Turing Machines

University Prescribed Syllabus

Turing Machines : Turing Machine Definition, Representations, Acceptability by Turing Machines, Designing and Description of Turing Machines, Turing Machine Construction, Variants of Turing Machine

Linear Bound Automata : The Linear Bound Automata Model, Linear Bound Automata and Languages

Pushdown Automata : Definitions, Acceptance by PDA, PDA and CFG

5.1 Introduction to Turing Machine

- In 1936, Mr. Alan Turing introduced advanced mathematical model for modern digital computer which was known as Turing machine. Turing machine is advanced machine of FA and PDA.
- This simple mathematical model has no difference between input and output set. This mathematical model can be constructed to accept a given language or to carry out some algorithm.
- This model sometimes uses it's own output as input for further computation.
- This machine or model can select current location and also decides location of memory by moving left or right. This mathematical model known as Turing machine has become an effective procedure.

- In short Turing machine is equal to,

  ★ **TM = FA + Tape**

  FA = Finite Automata

Tape

1. Infinite memory unit
2. Infinite cell (Each cell contains only one alphabet at one time)
3. Head of records move left or right.

- Turing machine's mathematical model consists of **Head** (moves right or left or stays in position), **infinite tape** and **finite set of states.**

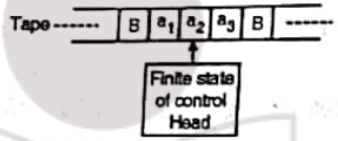


Fig. 5.1.1

- **Working of Turing machine** can be explained as follows,

where, one move of TM shows

1. Changing state location
2. Replacing old symbol by new
3. Left or right move of head

- A simple diagrammatic representation for Turing machine construction. As follows

Let's print the symbol 011 on initially blank tape.

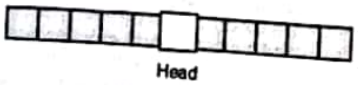


Fig. 5.1.2

First we write 0 in Head

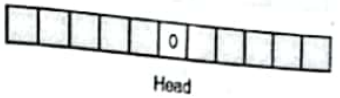


Fig. 5.1.3

Second we move the tape left by Head

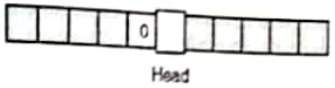


Fig. 5.1.4

Third we write 1 for Head

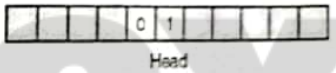


Fig. 5.1.5

Fourth we move left and insert 1

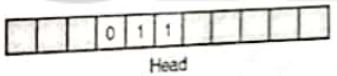


Fig. 5.1.6

- Let's see another one program for bit inversion in above tape (convert 0's and 1's vice-versa)

Simple instructions for program.

Read Symbol	Instructions to write	Instructions to move
Blank	None	None
1	Write 0	R
0	Write 1	R

- The machine checks head and reads the symbol under the read / write new symbol and move left or right. Let's see it step by step.

Fig. 5.1.7

Change the head symbol by 0

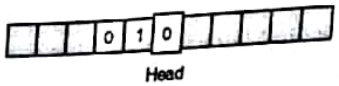


Fig. 5.1.8

Move right and check the head its again 1

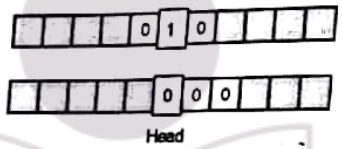


Fig. 5.1.9

Symbol read by head and repeat the same instruction

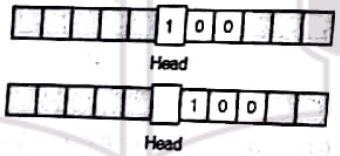


Fig. 5.1.10

hen the Blank symbol reads machine will be in halt state.

Syllabus Topic : Turing Machine Definition

Definition of Turing Machine

fine Turing Machine.

M define TM with seven Tuple,

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

Where,

Q = Finite set of states

Γ = Finite set of tape symbols

$B \in \Gamma$ = Blank space symbol

Σ = Set of input symbols where $\Sigma \subseteq \Gamma - \{B\}$ (excluding blank symbols)

δ = Function for next move $[Q \infty \Gamma$ to $Q \infty \Gamma \infty \{S\ R\}]$

q_0 = Start state

$F \subseteq Q$ = Final state set

Syllabus Topic : Representations

5.1.2 Representations of Turing Machine

How Turing machine can be represented ?

Turing machine can be represented by ID equations table or diagram.

(1) Instantaneous Description (ID)
(2) Transition Table
(3) Transition Graph

1. Instantaneous Description (ID)

- Let $\delta(q_0, a)$ be transition function then ID will be given as,

  $\delta(q_0, a) \rightarrow (q_1, x, R)$

  q_0 is initial state and read 'a' alphabet, transition function δ gives q_1 next state, a is replaced by x and move to the (R) right.
- We denote ID of TM by $\alpha_1\ q_0 \alpha_2$. q_0 is current state of M in Q. α_1 and α_2 are the strings of non blank symbols upto right most or left most head.

- Consider TM shown in the Fig. 5.1.11 obtain its ID

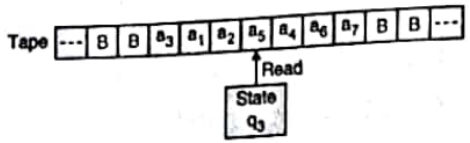


Fig. 5.1.11

- The symbol under head is a_5 and current state is q_3, so a_5 is written to the right of q_3. The non blank symbols to the left of a_5 is a string $a_3\, a_1\, a_2$ which is written to the left of q_3.
- The sequence of non blank symbols to the right of a_5 is $a_4\ a_6\ a_7$, Thus ID is $a_3\, a_1\, a_2\, q_3\, a_5\, a_4\, a_6\, a_7$ then, $\alpha_1 = a_3\, a_1\, a_2$ and $\alpha_2 = a_5\, a_4\, a_6\, a_7$, $\alpha_1\ q_3\ \alpha_2$
- The head can be moved left or right, which can be explained as follows :

(a) TM moves for left

The $\delta(q_0, a_i)$ induces a change in ID of the TM, calling a move.

Let $a_1\, a_2\, a_3\, \ldots\, a_n$ is an input string to be processed and present symbol under tube head is a_i.

Let $\delta(q_0, a_i) = (q_1, y, L)$

ID on a : Before processing will be, $a_1\, a_2\, a_3 \ldots a_{i-1}\, q_0\, a_i, a_{i+1} \ldots a_n$

ID on a : After processing will be, $a_1\, a_2\, a_3 \ldots a_{i-2}\, q_1\, a_{i-1}\, y\, a_{i+1} \ldots a_n$

This change of ID is represented as,

$a_1\, a_2\, a_3 \ldots a_{i-1}\, q_0\, a_i \ldots a_n \vdash a_1 \ldots a_{i-2}\, q_1\, a_{i-1}\, y\, a_{i+1} \ldots a_n$

(b) TM moves on right

$$\delta(q_0\, a_i) = (q_1, y, R)$$

then ID is represented by,

$a_1\, a_2 \ldots a_{i-1}\, q_0\, a_i \ldots a_n \vdash a_1\, a_2 \ldots a_{i-1}\, y\, q_1\, a_{i+1} \ldots a_n$

Note : $\vdash^*$ symbol denotes the reflexive transitive closure of relation $\vdash$.

2. Representation by Transition Table

- We can define δ in the form of table is denoted as transition table.

	a
q_0	x, q_1, R

- Transition function δ gives a alphabet from q_0 stated replaced by x, next state is q_1 and move to the right.

3. Representation of Transition Graph/Diagram

We can use transition diagram to represent Turing machine. Directed edges are used to represent transition of states.

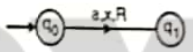


Fig. 5.1.12 $L(M) = \{w \mid w \in \Sigma^* q_0 w \vdash^* \alpha_1 q_f \alpha_2$ for q_f in F, α_1, α_2 in $\Gamma^*\}$

Where, q_0 and q_1 are transition states

a is current alphabet in insert tape

x is replacing alphabet

R means move to the right

E.g. Construct TM for string having even number of 1's over $\sum = \{1\}$.

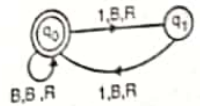


Fig. 5.1.13

B is blank symbol.

Syllabus Topic : Acceptability by Turing Machine

5.1.3 Language Accepted by Turing Machine

- The language accepted by TM is

$L(M) = \{w \mid w \in \Sigma^*$ and $q_0 w \vdash \alpha_1\, q_1\, \alpha_2$, for some q_1 in F and $\alpha_1\, \alpha_2$ in $\Gamma^*\}$

- Accepting states of machine have no **outgoing transition's.**
- Let M be Turing machine represented as,

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

- For this machine string w is set of input Σ^* of M is not accepted if M does not halt or halt in non accepting areas.
- The string w of set of input Σ^* is said to be acceptable by M only if

$$q_0 w \vdash^* \alpha_1 P \alpha_2 \text{ for } P \in F \text{ and } \alpha_1, \alpha_2 \in \Gamma^*$$

Acceptance of string by Turing machine

Accept IP string = If machine halts in an **accept state**

Reject IP string = If machine halts in a **non accept state** or

If machine centers an **infinite loop.**

Syllabus Topic : Designing and Description of Turing Machines

5.2 Designing and Description of Turing Machine

5.2.1 Design of Turing Machine

Explain design of Turing Machine.

1. Despite their simplicity R/W Turing machines are very powerful computing devices.
2. Head of Turing machine scans the symbol and machine remembers it, for future.
3. Minimization of number of states is activity by changing the states only when there is a change in the written symbol or when there is a change in the movement of the R/W head.
4. Turing machine works as mathematical model for solving complexity of language recognition problem.
5. TM is used for solving arithmetic operations, problems.
6. Diagram of language recognizing by Turing machine.

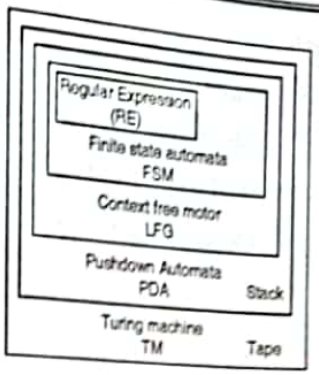


Fig. 5.2.1

All Regular Expressions and Context Free language are recognized by a particular type of machine, regular and not context free are also accepted by Turing machine.

Let's see the designing of Turing machine with examples.

Example 5.2.1 : Design Turing machine which can accept languages $0^*, 1^*$

Solution :

1. $L = \{\varepsilon, 0, 00, 000, ..., 001, 0011, 00111...0001...\}$
2. Scan all zero's using state q_0.
3. If string S ends accepts string, otherwise scan all 1's using state q_1 in this state, it will not accept any 0's

Turing machine is given below,

$Q = \{q_0, q_1\}$ $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, B\}$

δ = Transition function

Transition Table

State	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	q_{acc}
q_1	–	$(q_1, 1, R)$	q_{acc}
q_{acc}			Final

4. **Transition function**

$\delta(q_0, 0) = (q_0, 0, R)$ $\delta(q_0, 1) = (q_1, 1, R)$

$\delta(q_1, 1) = (q_1, 1, R)$ $\delta(q_0, B) = q_{acc}$ $\delta(q_1, B) = q_{acc}$

q_0 = It is initial state, $q_0 \in Q$

B = Blank $\in \Gamma$ it is used to mention start of string as well as end of string

F = It is finite set of the final state

$F \subseteq Q\ \{q_{acc}\}$

5. **ID for 000111**

In tape, string is B000111BB

$\rightarrow B\ q_0\ 000111BB \rightarrow B0\ q_0\ 00111BB \rightarrow B00\ q_0\ 0111BB$

$\rightarrow B000\ q_0\ 111BB \rightarrow B0001\ q_1\ 11BB \rightarrow B00011\ q_1\ 1BB$

$\rightarrow B000111\ q_1\ BB \rightarrow B000111\ q_{acc}\ B \rightarrow B000111\ q_{acc} \rightarrow$ Accept

6. **ID for 000110B**

In tape string B000110BB

$\rightarrow B\ q_0\ 000110BB \rightarrow B0\ q_0\ 00110BB \rightarrow B00\ q_0\ 0110BB$

$\rightarrow B000\ q_0\ 110BB \rightarrow B0001\ q_1\ 10BB \rightarrow B00011\ q_1\ 0BB \rightarrow$ Error state

$\rightarrow$ Notaccept

Example 5.2.2 : Design Turing machine which can accept language $0(0 + 1)^* 1$.

Solution :

1. L = {01, 001, 011, 0101, 0011 ...}
2. State q_0 is used to check whether string started with 0 or not ? (1 is not the 1st symbol)
3. State q_1 and q_2 are used to assure incoming symbol and check last one symbol is 1 or not ? (0 is not the last symbol)
4. State q_{acc} is final state.

Turing machine is given below :

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

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

$\Gamma = \{0, 1, B\}$

δ = Transition function

Transition function table

State	0	1	B
q_0	$(q_1, 0, R)$	–	–
q_1	$(q_1, 0, R)$	$(q_2, 1, R)$	–
q_2	$(q_1, 0, R)$	$(q_2, 1, R)$	q_{acc}
q_{acc}			Final

Transition function

$(q_0, 0) = (q_1, 0, R), (q_1, 0) = (q_1, 0, R), (q_1, 1) = (q_2, 1, R)$

$(q_2, 0) = (q_1, 0, R), (q_2, 1) = (q_2, 1, R), (q_2, B) = q_{ac}$

ID for 0101 string $\rightarrow$ In tape, string is B0101BB

$\rightarrow B\ q_0\ 0101BB \rightarrow B0q_1\ 101BB \rightarrow B01\ q_2\ 01BB$

$\rightarrow B010\ q_1\ 1BB \rightarrow B0101\ q_2\ BB \rightarrow B0101\ q_{acc} \rightarrow$ Final state accept

Now we see ID for 0100

$\rightarrow$ In tape string is B0100BB

$\rightarrow Bq_0\ 0100BB \rightarrow B0\ q_1\ 1\ 00BB \rightarrow B01\ q_2\ 00BB$

$\rightarrow Bq_0\ 010\ q_1\ 0BB \rightarrow B0100\ q_1\ BB \rightarrow$ Reset

---

Example 5.2.3 : Design Turing machine that accepts $\{a^n b^n \mid n \geq 1\}$

Solution :

- L = {ab, aabb, aaabbb....}
- 1st scan leftmost a and make it x using state q_0.

- State q_1 is used for scan b, change b to y and move backward.
- Find rightmost X and repeat above steps in all characters change.
- Check whether there are extra characters or not.

Turing machine is given below :

$u = \{q_0, q_1, q_2, q_{chk}, q_{acc}\}$

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

$\Gamma = \{a, b, x, y, B\}$

δ = Transition function

Transition table

State	a	b	x	Y	B
q_0	(q_1, x, R)	-	-	(q_{chk}, y, R)	-
q_1	(q_1, a, R)	(q_2, y, L)	-	(q_1, y, R)	-
q_2	(q_2, a, L)	-	(q_0, x, R)	(q_2, y, L)	-
q_{chk}	-	-	-	(q_{chk}, y, R)	q_{acc}
q_{acc}	-	-	-	-	Final

Make it clear by transition diagram

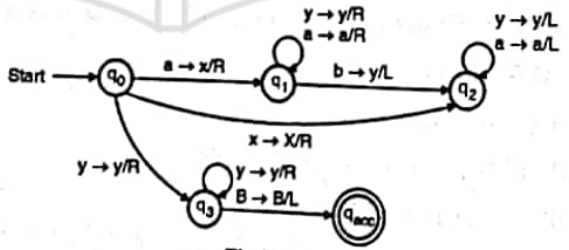


Fig. P. 5.2.3

Diagram explains that :

1. State q_0 takes initiative to replace leftmost a by x and change the state to q_1.

2. Function of q_1 is to search rightmost a's and y's until it finds leftmost b. If machine finds b, it changes it to y and enters state q_2.
3. State q_2 proceeds searching left for x and enters state q_0 upon finding it, move right to change a and the state.
4. All a's and b's are changes to x and y then q_{chk} condition is satisfied. If B or x is encountered before b, then input is rejected either there is existence of a is moré than b or input $a^* b^*$.
5. From q_0 scanning y state q_3 is entered to scan over y's and check for number of b's remaining. If the y's are followed by a B, state q_{acc} is accepted otherwise rejected.

ID for aa bb → BaabbBB

→ Bq_0 aa bb BB

→ B x q_1 abb BB → xa q_1, bb BB → x q_2 ay b BB

→ B q_2 x a yb → Bx q_0 a y b BB → B xx q_1 yb BB

→ Bxx y q_1 b BB → B xx q_2 yy BB → B x q_2 x yy BB

→ Bxx q_0 yyBB → Bxx y q_3 yBB → B xx yy q_3 BB

→ Bxx yy B q_{acc} → q_{accept} → Final state

→ aabb is accepted.

Example 5.2.4 : Design Turing machine for language, $L = \{a^m b^n \mid m > n \times n > 0\}$

Solution :

$L = \{aab, aaab, aaaab, \ldots aaabb, aaaabb\}$

Turing machine,

$Q = \{q_0, q_1, q_2, q_{chk}, q_{acc}\}$

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

$\Gamma = \{a, b, x, y, B\}$

δ = Transition function

Steps in this construction can be written as,

1. M starts in state q_0

2. If symbol a is read, then state q_0 changes to q_1 and replace symbol by x and move towards right then check for 1st occurrence of b, skip a and y.
3. b is replaced by y and enter the state q_2 and move towards left to search for x then move to right.
4. If symbol is b, then repeat above step, if symbol x is found out then move towards left until current symbol is not equal to a.
5. If current symbol is a that means there is at least one a is extra, then check blank on left side of the string.

Transition table

	a	b	x	y	B
q_0	(q_1, x, R)	-	-	-	-
q_1	(q_1, a, R)	(q_2, y, L)	-	(q_1, y, R)	(q_{chk}, y, L)
q_2	(q_2, a, L)	-	(q_0, x, R)	(q_2, y, L)	-
q_{chk}	(q_{chk}, a, L)	-	(q_{chk}, x, L)	(q_{chk}, y, L)	q_{acc}
q_{acc}	-	-	-	-	Final state

ID for aaab $\rightarrow$ BaaabBB

$\rightarrow B q_0 aaa b BB \rightarrow B x q_1 aa bB \rightarrow B x a q_1 a b B$

$\rightarrow B x aa q_1 b B \rightarrow B x aa q_2 y B \rightarrow x a q_2 a y B$

$\rightarrow B x q_2 a a y B \rightarrow B x q_0 aa y B \rightarrow B xx q_1 a y B$

$\rightarrow B xx a q_1 y B \rightarrow B xx a y q_1 B \rightarrow Bxx a y q_{chk} B$

$\rightarrow Bxx a q_{ch} k B$

$\rightarrow B xx q_{chk} ay \rightarrow Bx q_{chk} x a y B$

$\rightarrow B q_{chk} xx ay B \rightarrow q_{accept}$

Example 5.2.5 : Design Turing machine to recognize language $L = \{ a^{2^n} \mid n \geq 0 \}$.

Solution :

$L = \{aa, aaaa, aaaaaa\}$

Turing machine for language,

$Q = \{ q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej} \}$

$\Sigma = \{a\}$

$\Gamma = \{a, x, B\}$

Steps in this construction can be written as,

1. For constructing Turing machine for this language we have to check step 1 contained single a, then accept it.
2. If step 1 contained more than a single a and the number of a's was odd then reject it.
3. Return the head to the left hand of the tape.
4. Visit the step 1 again. At each repetition of step 1 number of a's will be half.
5. If the number of a's is odd and greater than one, the original number could not have been power of 2 then machine rejects it.
6. If the number of a is one than the original number of 'a' must have been power of 2 so it will be acceptable by machine.

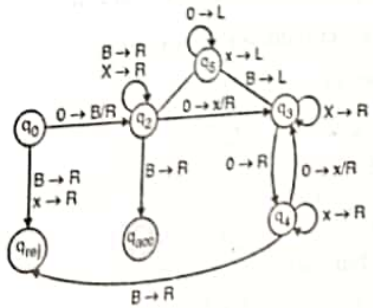


Fig. P. 5.2.5

Find out ID for aaaa

$\rightarrow q_0$ aaaa $\rightarrow$ Bq_1 aaa $\rightarrow$ Bx q_2 aa $\rightarrow$ Bx a q_3 a

$\rightarrow$ Bx a x q_2 B $\rightarrow$ B x a q_4 x B $\rightarrow$ B q_4 x a x B

$\rightarrow q_4$ Bx a x B $\rightarrow$ B q_1 x a x B $\rightarrow$ B x q_1 a x B

$\rightarrow$ B xx q_2 x B $\rightarrow$ B xxx q_2 B $\rightarrow$ B xx q_4 x B

$\rightarrow$ B x q_4 xx B $\rightarrow$ B q_4 xxx B $\rightarrow q_5$ BxxxB

$\rightarrow$ B q_1 xxx B $\rightarrow$ Bx q_1 xx B $\rightarrow$ Bxx q_1 x B

$\rightarrow$ B xxx q_1 B $\rightarrow$ B xxx B q_{acc}

This Turing machine begins for using a blank over the leftmost. This allows to find the left end of the tape. It also allows machine to identify the case when tape contains one a only.

Example 5.2.6 : Construct Turing machine accepting following language,

$L = \{ a^n b^m c^{m+n} \mid m, n >= 0\}$.

Solution :

Language L = {ε, ac, bc, aacc, bbcc, abcc}

1. First scan leftmost a or b and make it α using state q_0.
2. Scan rightmost c and make it y.
3. Find out rightmost x
4. Repeat step 1 to 3 till all a's, b's and c's are not over.
5. Check whether there is extra a or b or c.

Turing machine is given below

$Q = \{ q_0, q_1, q_2, q_3, q_{Acc}\}$

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

$\Gamma = \{a, b, c, x, y, B\}$

δ = Transition function.

Transition table

State	a	b	c	x	y	B
q_0	(q_1, x, R)	(q_1, y, R)	-	-	(q_4, y, R)	-
q_1	(q_1, a, R)	(q_1, b, R)	(q_1, c, R)	-	(q_2, y, L)	(q_2, B, L)
q_2	-	-	(q_3, y, L)	-	-	-
q_3	(q_3, a, L)	(q_1, b, L)	(q_1, c, L)	(q_0, x, R)	-	q_{acc}
q_4	-	-	-	-	(q_4, y, R)	q_{acc}
q_{acc}	-	-	-	-	-	Final

Instantaneous description for abcc $\rightarrow$ Babcc BB

$\rightarrow$ B q_0 a b cc B $\rightarrow$ B X q_1 b cc B $\rightarrow$ B X b q_1 c cB

$\rightarrow$ B x b c q_1 c B $\rightarrow$ B x b cc q_1 B $\rightarrow$ B x b cc q_2 B

$\rightarrow$ B x b c q_3 y B $\rightarrow$ B x b q_3 c y B

$\rightarrow$ B x q_3 bc y B $\rightarrow$ B x q_0 b c y B

$\rightarrow$ B x x q_1 c y B

$\rightarrow$ B xx c q_1 y B

$\rightarrow$ B xx c q_2 y B

$\rightarrow$ B x x q_3 yy B

$\rightarrow$ B xx y q_3 y B

$\rightarrow$ B xx yy q_3 B $\rightarrow q_{accept}$

Example 5.2.7 : Design a Turing machine for language,

$L = \{ww^R \mid w \in (0 + 1)^R\}$ $L = \{ww^R \mid w \in (0 + 1)^R$ where w^R is reverse of w}.

Solution :

Language for above Turing machine is

L = {00, 11, 0110, 1001, 001100, 100001...}

We can apply simple logic on this language.

1. If initial state q_0 read 0, then last state also read 0.
2. If initial state q_0 read 1, then last state also read 1.

Turing machine is :

$Q = \{ q_0, q_1, q_2, q_3, q_4, q_5, q_{Acc} \}$

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

$\Gamma = \{0, 1, x, B\}$

Where δ is

Step	0	1	x	B
q_0	(q_1, x, R)	(q_4, x, R)	(q_{acc}, x, R)	-
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	(q_2, x, L)	(q_2, B, L)
q_2	(q_3, x, L)	-	-	-
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, x, R)	-
q_4	$(q_4, 0, R)$	$(q_4, 1, R)$	(q_5, x, L)	(q_5, B, L)
q_5	-	(q_3, x, L)	-	-
q_{acc}	Final state	-	(q_{acc}, x, R)	(q_{acc}, B, R)

Consider string 0110

ID for B 0 11 0 BB

$\rightarrow B\ q_0\ 0\ 110\ B \rightarrow B\ x\ q_1\ 110\ B\ B$

$\rightarrow B\ x\ 1\ q_1\ 10\ B \rightarrow B\ x\ 110\ q_1\ BB$

$\rightarrow B\ x\ 11\ q_2\ 0\ BB \rightarrow B\ x\ 1\ q_3\ 1x\ B\ B$

$\rightarrow B\ x\ q_3\ 11\ x\ B\ B \rightarrow B\ q_3\ x\ 11\ x\ B\ B$

$\rightarrow B\ q_0\ 11\ x\ BB \rightarrow B\ x\ x\ q_4\ 1x\ Bb$

$\rightarrow Bxx\ 1\ q_4\ x\ B \rightarrow Bxx\ q_5\ 1xB$

$\rightarrow B\ x\ q_3\ xxx\ B$

$\rightarrow B\ xx\ q_0\ xx\ B \rightarrow B\ xxx\ q_{acc}\ x\ B$

$\rightarrow B\ xx\ xx\ q_{acc}\ B \rightarrow B\ xxxx\ q_{acc}$

$\rightarrow q_{acc}$ = accept the string

5.2.2 Description of Turing Machine

- The formal description of Turing machine includes current status of head input symbol, resulting state, the tape symbol replacing the input symbol and movement of Read/Write (R/W) head to the left or right.
- The transition function δ is a partial function and not defined for all (q, x) where partial function δ is defined as,

  $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times (L, R)$
- The higher level of description and transition function for a Turing machine is called as a implementation description.
- Where movement of head, symbols stored, states etc can be described in English.
- E.g. move to the left till the end of input string. This instruction requires multiple move.

  One instruction = n × instruction

  (Implementation description) (standard Turing machine)

  n → n can be any number more than one.
- High level description involves instruction in English language and specification of state or transition function is not required,

Syllabus Topic : Turing Machine Construction

5.3 Turing Machine Construction Techniques

- Construction of Turing machine needs complete set of states used for storing information.

- In this section the basic ideas will be presented through various example.

(a) Use of storage in the finite control

- Finite control of Turing machine means its tape, head which is used for reading and writing information on the tape.
- Finite control can be used to hold finite amount of information.
- To do so, the state is written on a pair of elements are exercise of control and others storing the symbol and the new state becomes $Q \times \Gamma$.
- Turing machine is supplied input from left to the end of the tape squares immediately to the right of the blank symbol.
- We define δ (q0, a) $\rightarrow$ (q1 X, R/L). This means that q1 is changing state of q_0, a is replaced by X and move the state right or left and continues it in remains cells.

(b) Use of multiple tracks

- We can imagine single tape of Turing machine is divided into k tracks where, $K \geq 1$.
- In multiple track, Turing machine consists K tuples, therefore tape symbols are elements of Γ^K. The moves are in similar manner.

For K = 3

B	1	0	1	1	B
B	B	B	1	1	B
B	1	0	1	0	B

(c) Use of checking symbols

Checking of symbols is a useful track for visualizing how a Turing machine recognizes languages defined by repeated strings such as

{ww | w in $\sum^*$}

{wwR | w in $\sum^*$ and w^R is reverse of w}

(d) Use of shifting over

A Turing machine can make space on its tape by shifting all non blank symbols a finite number of cells to the right.

(e) Use of subroutines

- In computer languages, programs are designated by modulating or top-down design. Some task elementary process or has to be done repeatedly, this is called sub-routines.
- A Turing machine is similar are type of sub-routing found in programming languages including recursive procedure and any of the known parameter parsing mechanism.

Syllabus Topic : Variants of Turing Machine

5.4 Variants of Turing Machine

☞ **Explain variants of Turing Machine.**

- We have seen basic structure for Turing machine with infinite tape in both direction and deterministic standard.
- Now we are going to learn enhancement in Turing machine with its modified structure.
- Acceptance power of variations of Turing machine are same as basic structure of Turing machine. Different types of Turing machine as per following :

1. Multitrack Turing machine
2. Multitape Turing machine
3. Non-deterministic Turing machine
4. Multistack Turing machine
5. Counter machine.

1. Multitrack Turing Machine

- Multitrack Turing machine is a one tape, containing multiple track. In multitrack Turing machine, a single finite control head reads n symbols from n tracks at one step. It can be understood by Fig. 5.4.1.

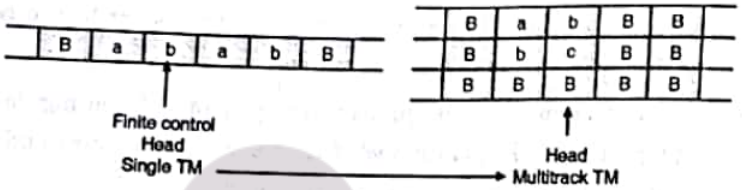


Fig. 5.4.1

- Transition function for mutlitrack Turing machine :

  $\delta(q_0, b, c, B) = (q_{next}, x, y, z, R \text{ or } L)$

- q_0 is initial state, b, c, B are symbols which are held by head. Head will be changing the states to q_{next}.
- Transition function for multirack Turing machine is

  $\delta : Q \times \Gamma^n \Rightarrow Q \times \Gamma^n \times X\,(L, R)^n$.

- Symbol, b is changed to the α, c is changed to y and B is changed to z, move the head left or right. Language acceptance power of single track Turing machine is equivalent to multitrack Turing machine.

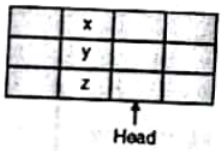


Fig. 5.4.2

2. Multitape Turing Machine

- Multitape Turing machine have n tapes for reading and writing information's with different finite control head.

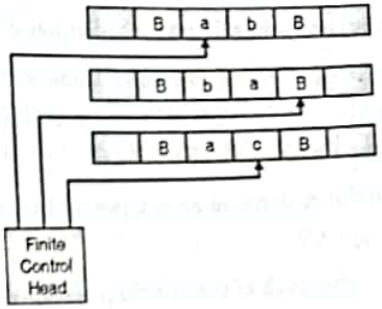


Fig. 5.4.3

- Fig. 5.4.3 depicts that different heads for multitape Turing machine.
- We can see transition function for multitape Turing machine,

  $\delta(q_0, a, B, C) = (q_{next}, x, y, z, L, R, L)$

- q_0 is initial state contains a, B, C symbols, individuals head of each tape change the symbol to x, y, z in q_{next} state as per the movement left or right.

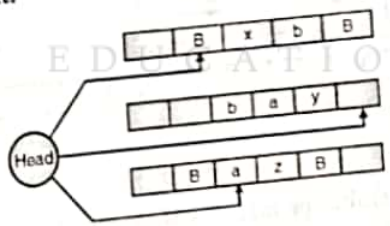


Fig. 5.4.4

- Transition function for Multitape Turing machine,

  $\delta : Q \times \Gamma^n \rightarrow Q\Gamma^n \times \{L, R, S\}^n$

- Where S is no shift or the tape head should stay there. It i transition function based on all n current symbols, write all symbo to each of the n tapes and move each of the n tapes either left right.

- Tree structure for ID is,

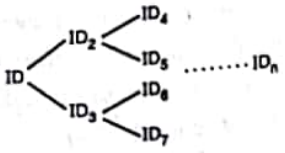


Fig. 5.4.7

- NTM accepts string w if there exists a sequence of moves starting from the initial ID to an accepting condition.
- Example Fig. 5.4.8 explains the states of NDTM.

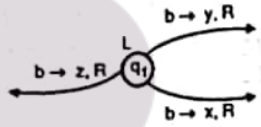


Fig. 5.4.8

- We can construct tree structure for Fig. 5.4.8. Like.

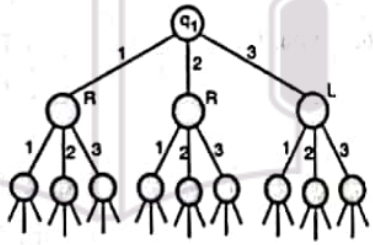


Fig. 5.4.9

- In Fig. 5.4.9 q_1 has 3 choices, in some computation there will be fewer than 3 choices in the computation steps.
- Some points will have zero choice these states or branches of the non determinism halt or reject.
- Despite of NDTM has several action performing over step.
- Acceptance power of NDTM and DTM are same.

- We will show this by simulating the NDTM at each step in the computation with deterministic one.

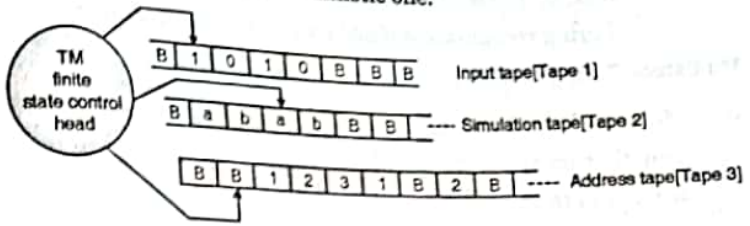


Fig. 5.4.10

1. Input tape store initial input, never modified.
2. Simulation tape used like the tape of a deterministic Turing machine to perform the simulation.
3. Address tape keep track of states like breadth first search, it tells which choices to make during simulation.

- A set of transitions exist at each step in the computation of NDTM. We will number these 1 to n where n is the largest number of options any state has.
  1. Initially tape 1 contains the input and tapes 2 and 3 are empty.
  2. Copy tape 1 to tape 2.
  3. Run the simulation using tape 2. When choices occur (i.e. when non deterministic branch points are encountered) consult tape 3. Tape 3 contains the path, the next symbol on each number tells which choice to make. For example, 132 means that we took the first option initially, then the third option and last will be the second option. Run the simulation all the way as far as the address or path goes, if the choice given is invalid, abort this state or branch, and go to the step 4. If get abort for reject state then go to step 4, otherwise get accept.

4. Try the next state or branch and increment the address on tape 3, repeat the step 2. So we can say that, a language is Turing recognizable if and only if NDTM recognizes it.

4. Multistack Turing Machine

- Multistack Turing machine is like semi-infinite tape i.e. read only tape that means you can only read but not write, we can move left or right in tape.
- In PDA, we cannot modify the input and scan the symbol only left to right in one stack.
- But PDA with more than one stack can accept any language which a Turing machine can accept.

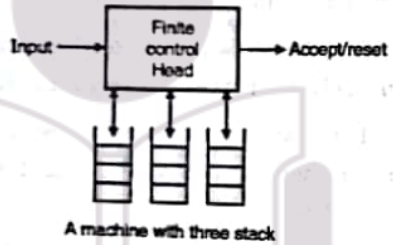


Fig. 5.4.11

In Multistack Turing machine,

1. Finite control head obtain its input from an input source and not from the tape.
2. It has finite stack symbols, which it uses for all its stack.
3. Finite control head reads input symbol which is chosen from the finite input symbols.
4. The top stack symbol on each of its stacks.
5. Finite control head moves to the next state, replace the top symbol with blank or another stack symbols. Usually a different replacement of string for each stack.

5. Counter Machines

1. Counter machine is like multistack machine but in place of each stack is a counter.
2. Counter register holds non-negative integers .
3. Move of the counter machine depends on its state, input symbol and counters which are zero.
   (a) Change the new state.
   (b) Increment or decrement by one from any of its counters.

A counter has always positive value, so counter value cannot be less than 0.

Syllabus Topic : The Linear Bound Automata Model

5.5 Linear Bound Automaton

Write a short note on Linear Bound Automaton and Linear Bound Automaton Model.

- LBA is a restricted form of Turing machine. This machine accepts context sensitive languages.
- A Turing machine that uses only the tape space occupied by the input is called a Linear Bound Automaton (LBA).

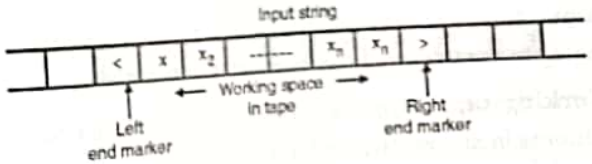


Fig. 5.5.1

- A Linear Bounded Automata is a Non-deterministic Turing machine which has a restricted length of tape but bounded by a linear function of the length of input string.

$M = \{Q, \Sigma, \gamma, \delta, q_0, B, <, >, R\}$

Such that all symbols have the same meaning as in the basic model, < and > are special symbols, <, > ∈ Σ.

Both symbols are end markers < is left end marker and > is right end marker. The Turing machine cannot replace < or > with anything else, nor move the tape head left of < or right of >.

LBA model

- We represent Fig. 5.5.2 in block diagram as a LBA model.

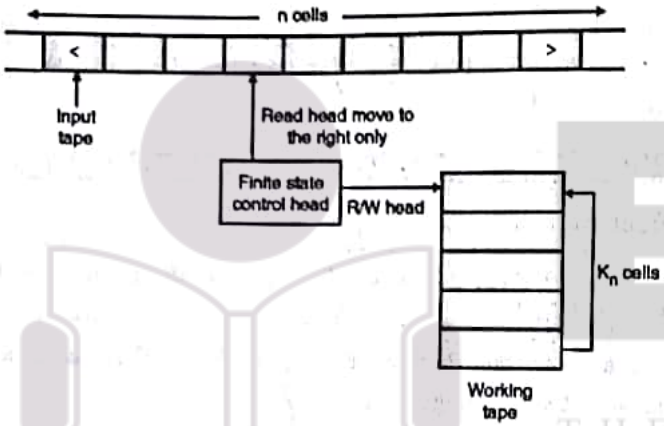


Fig. 5.5.2 : LBA model

- There are two tapes
  1. **Input tape :** On the input tape, the head never prints and never moves to the left.
  2. **Working tape :** On the working tape the head can modify the contents in any way without any restriction.

The mapping of transition function δ is represented by (q, w, k)

where q ∈ Q, w ∈ Γ, k is some integer between 1 and n.

The transition of ID's is similar except,

1. If R/W head move to left, k changes to k – 1.
2. If R/W head move to right, k changes to k + 1.

The language accepted by LBA is defined as the set,

$\{w \in (\Sigma - \{<, >\})^* \mid (q_0, < w >, 1)) \vdash^* (q, \alpha, i)\}$

for some q ∈ F and for some integer i between one to n.

* Null string may accepted by LBA.

Syllabus Topic : Linear Bounded Automata and Languages

5.5.1 Linear Bounded Automata and Languages

What are the languages accepted by Linear Bound Automaton ?

- The set of strings accepted by non deterministic LBA is the set of strings generated by the context sensitive grammar (CSG), excluding the null strings.
- If L is context sensitive language, then L is accepted by a linear bounded automata the converse is also true. We can construct type 0 grammar accepted by Turing machine for language which will take input string M, tries to apply productions backwards to get finally start state q_0.
- The productions are constructed in two steps.
  1. Transform the string $(q_0 < w >)$ into the string $(q_1\ b)$ where q_0 is initial state q_1 is an accepting state, < and > are left and right end marker respectively. This is called the **transformational grammar.**
  2. Inverse of production rules by reversing the production of transformational grammar. The construction is in such a that w is accepted by M iff w ∈ < L (G).
- A LBA machine accepts a string w after starting at the state with R/W head reading < (left and mark), and halts over > (right end mark) in a final state, otherwise w is rejected.
- The production rules for the generative grammar are constructed as in the case of Turing machine.

- The following additional productions are needed in case of LBA

  $a_i\, q_f > \Rightarrow q_f >$ for all $a_i \in \Gamma$

  $< q_f > \Rightarrow < q_f, < q_f \rightarrow q_f$

Examples of languages accepted by LBA

1. $L = \{a^n b^n c^n \mid n \geq 1\}$
2. $L = \{a^{n!} \mid n \geq 0\}$
3. $L = \{ww \mid w \in (a, b)^*\}$
4. $L = \{w^n \mid w \in (a, b)^* n \geq 1\}$
5. $L = \{www^R \mid w \in (a, b)^*\}$

Syllabus Topic : Pushdown Automata

5.6 Pushdown Automata

☞ **Write components of Pushdown Automata.**

- **A pushdown automata** is a way to implement a context free grammar for a regular grammar.
- It can remember on infinite information.
- It is combination of finite state machine and a stack.
- A stack has **two** operations namely **PUSH operation** and **POP operation**.

  Push operation : It is used to insert an element into stack.

  Pop operation : It is used to delete an element from stack.

→ **Top pointer :** Which point to the current input of stack.

PDA has 3 components

1. An input tape
   A control unit
   A stack with infinite size

Appn - can be used as language acceptor
→ compiler design (parser design for syntactic analysis)
→ Online Transaction process system
→ Tower of Hanoi (Recursive soln)

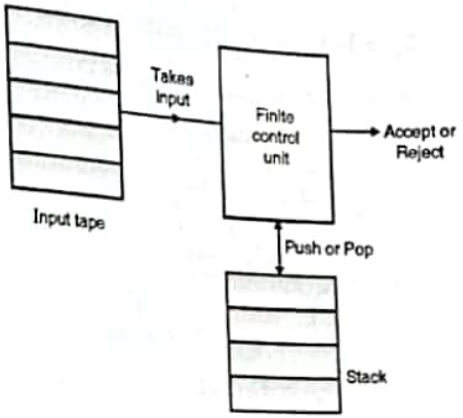


Fig. 5.6.1

Operation

User can give the input to the **input tape** and it will take to **finite control unit**, **stack symbol** only push or pop the input values and can be control led by finite control unit, then it can be accept or reject.

5.6.1 Definition of Push Down Automata(PDA)

☞ **Define a Push Down Automata.**

PDA has 7 tuples

$(Q, \Sigma, \Gamma, \delta, q_0, Z_0, f)$

τ symbol is pronouns "Taw"

Where, Q = Finite number of states

Σ = Input alphabet

Γ = Stack symbol

δ = Transition function

$Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma) \rightarrow Q \times \Gamma^*$

q_0 = Initial state ($q_0 \in Q$)

Z_0 = Initial stack top symbol ($\Gamma \in S$)

F = Set of accepting states ($F \in Q$)

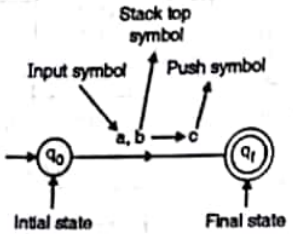


Fig. 5.6.2

For example,

Construct a PDA that accepts $L = \{0^n, 1^n \mid n \geq 0\}$.

Step I :

Problem Description

- If you encounter input '0' and top is null, push '0' into stack. This may iterate.
- If you encounter input '1' and top is '0', pop '0' from stack. This may iterate.
- The process is repeated until black space and Z_0 came.
- This indicates that string is valid, otherwise strength is invalid.

Step II :

Construct transition diagram for PDA

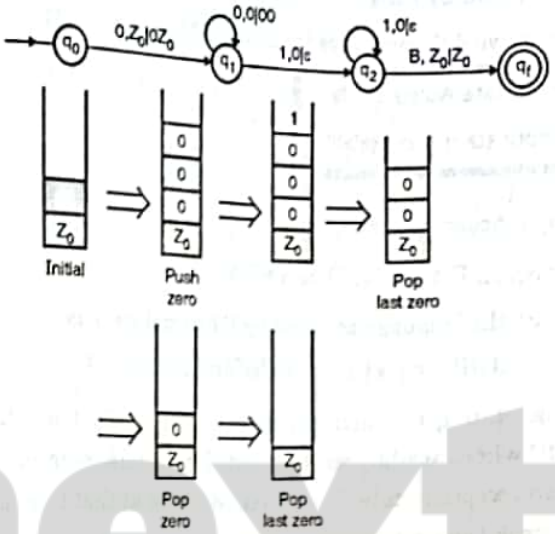


Fig. 5.6.3

Step III :

Transition table for PDA

Transition	Operation Performed
$q_0 : \delta(q_0, 0, Z0) \rightarrow (q_1, 0z0)$	push o
$q_1 : \delta(q_1, 0, 0) \rightarrow (q_1, 00)$	push o
$\delta(q_1, 0, 1) \rightarrow (q_2, \varepsilon)$	pop 0
$q_2 : \delta(q_2, 1, 0) \rightarrow (q_2, \varepsilon)$	pop 0
$\delta(q_2, B, Z_0) \rightarrow (q_f, Z_0)$	Accept
q_f : Accept	

5.6.2 Acceptance by PDA

There are two different ways to define PDA acceptability.

1. Final state Acceptability
2. Empty state Acceptability

1. Final state Acceptability

Let $P = \{Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\}$ be a PDA

then L (P), the language accepted by P by final state is

$$L(P) = \{ w \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \alpha)\}$$

For some state $q \in F$ and any stack string α. That is, starting in the initial ID with w waiting on the input, P consumes w from the input and enters an accepting state. The content of stack at that time is irrelevant.

2. Empty stack Acceptability

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. Then N(P) ; the language accepted by P by empty stack is,

$$N(P) = \{w \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)\}$$

for any state q. That is N(P) is the set of input w, that P can consume and at the same time empty in stack. (N (P) stands for null stack or empty stack).

Syllabus Topic : PDA and CFG

5.6.3 Push Down Automata (PDA) and Context Free Grammar (CFG)

Push Down Automata (PDA) to Context Free Grammar (CFG)

Theorem : If L is a CFL, then we can construct a PDA accepting language L by empty store i.e. L = N(A)

Method

Let L = L(G), where $G = (V_N, \Sigma, P, s)$ in a context free grammar we can construct a PDA as,

$A = (\{q\}, \Sigma, V_N \cup \Sigma, \delta, q, S, \Phi)$

Where δ is defined by following rules.

$R_1 = \delta(q, \varepsilon, A) = \{(q, \alpha) \mid A \to \alpha \text{ is in P}\}$

$R_2 = \delta(q, a, a) = \{(q, \varepsilon) \mid \text{for every a in } \Sigma$

Rule 1

Pushdown symbols in A are variable or terminals. If PDA reads a variable A on the top of PDS (Push Down Stack), it makes a ε move by placing RHS of any A production

Rule 2

If PDA reads a terminal a on PDS (Push Down Stack) and if it matches with current input symbol then PDA erase a.

Example 5.6.1 : Construct a PDAA which is equivalent to the following given CFG.

$S \to 0CC, C \to 0S, C \to 1S, C \to 0$

Test whether 010^4 is accepted by N(A)

Solution :

Step 1

The target PDA a is as following :

$A = (\{q\}, \{0, 1\}, \{S, C, 0, 1\}, \delta, q, s, \phi)$

δ is defined by following rules :

R_1 : $\delta(q, \varepsilon, S) = \{(q, 0CC)\}$

$\delta(q, \varepsilon, C) = \{(q, 0S), (q, 1S), (q, 0)\}$

R_2 : $\delta(q, 0, 0) = \{(q, \varepsilon)\}$

$\delta(q, 1, 1) = \{(q, \varepsilon)\}$

Step 2

Check if 010^4 can get from this production,

$S \Rightarrow 0CC$

$\Rightarrow 01SC$

$\Rightarrow 010CCC$

$\Rightarrow 010^4$

Step 3

Construct PDA

$(q, 010^4, S) \vdash (q, 010^4, 0CC) \vdash (q, 10^4, CC)$

$\vdash (q, 010^4, 1SC) \vdash (q, 0^4, SC)$

$\vdash (q, 0^4, 0CCC) \vdash (q, 0^3, CCC)$

$\vdash (q, 0^3, 0CC) \vdash (q, 0^2, CC)$

$\vdash (q, 0^2, 0C) \vdash (q, 0, C)$

$\vdash (q, 0, 0) \vdash (q, \varepsilon, \varepsilon)$

Thus $010^4 \in N(A)$

□□□

CHAPTER 6

Undecidability

University Prescribed Syllabus

The Church-Turing thesis, Universal Turing Machine, Halting Problem, Introduction to Unsolvable Problems.

Syllabus Topic : The Church-Turing Thesis

6.1 Church Turing Thesis

☞ **Write Short note on Church Turing Thesis.**

- Many mathematicians were finding if there is a procedure, algorithm for solving a problem that can be implemented on Turing machine Alonzo church also worked on same problem and he solved and some problem Chomsky unrestricted grammar recursive recues function theory by λ calculus.
- If a computational problem P is solvable by human being then it is solved by computer and if its solvable by computer then it is solved by Turing machine.
- Mr. Church formalise simple hypothesis over it. If a computational problem P is solvable by human being then it is directly solve by Turing machine.

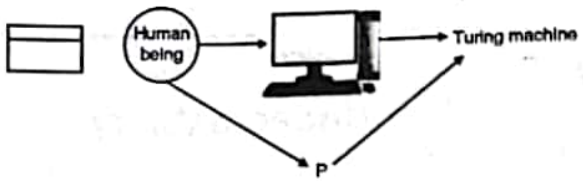


Fig. 6.1.1

Above formalism leads to,

1. **Turing theory (Weak form) :** A turing machine can compute anything that can be composed by digital computer.
2. **Turing theory (strong form) :** A turing machine can compute anything that can be computed.

Define string,

D = {p : P is a polynomial with an integral root }

Consider turing machine,

M = {The input is a polynomial over variable $x_1, x_2, \ldots x_n$ }

- Evaluate P on n tuple of integers.
- If p ever evaluates to 0, accept
- M recognized D but does not decide D.

The Church Turing thesis explain similarities between recursive function and computable one,

1. Recursive function is a mathematical concept.
2. Computable function is more of constructivism.

Now it is universally accepted by computer scientists that turing machine is a mathematical model of an algorithm.

Syllabus Topic : Universal Turing Machine

6.2 Universal Turing Machine (UTM)

Write Short note on UTM.

- The limitations of Turing machine execute only one program at a time hence it must be constructed for every new computation to be performed for a every input output relation.

- This is why Turing machine introduce as a solution could be used to work like a simulator a computer with an arbitrary program.
- This Turing machine is capable to simulate the action of any Turing machine on any input, is called ***Universal Turing Machine.***

6.2.1 Attributes of UTM

- Reprogrammable machine
- Simulates any other Turing Machine

UTM gives following information in its tape,

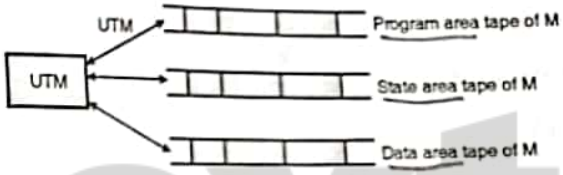


Fig. 6.2.1

1. The description of in terms of its machine.
2. The initial configuration of M i.e. starting state and the start symbol, (state area of tape).
3. The processing data (input string) to be fed to M (Data are tape of M) UTM algorithm.
   1. Scan the state area of the tape and read the symbol (O are tape of M) from the same area that M reads to start with and also the starting or initial state of M.
   2. Move side tape to the program area contains the description of M and find the row in machines headed by the state symbol read in step 1.
   3. Find the column headed by the input symbol read in step 1 and read the triplet (new symbol (a), new state (a), direction to move) stored as the entry (current state, current input symbol), which is the entry at the intersection of row and column f and in step 2.
   4. Move the tape to reach the appropriate cell in the data area, replace the symbol by new symbol, move the head in required direction, read the next symbol and finally reach the state area and replace the state scanned symbol then goto step 1.

- UTM has, linear tape as any other turing machine hence it cannot be directly put finite machine of M on this tape, because finite machine is a two dimensional matrix, it has to be converted into one directorial form.
- In UTM we encode M as a string of symbols line,

(1) Alphabet encoding

Symbol	a	b	c	d
	↓	↓	↓	↓
Encoding	1	11	111	1111

(2) State encoding

Store	q_1	q_2	q_3	q_4
	↓	↓	↓	↓
Encoding	1	11	111	1111

(3) Head move encoding

Move	L	R
	↓	↓
Encoding	1	11

We can perform transition encoding by using above symbols

e.g. transitions.

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

↓

ncoding 1# 1#1 1#11#1

↑ ↑ ↑ ↑

Separator

Same as in turing on active encoding.

Transition $\delta(q_1,a) = (q_2, b, L)$ $\delta(q_2,b) = (q_3, C, R)$

Encoding ↓ ↓ ↓

1#1#11 ##11 #1 # # 11 # 11 # 111 # 111 # 11

↑

Separator

turing machine encodes transition in binary format of the simulator machine M.

1 # 1 # 11 # 11 # 1 ## 11 # 11 # 111 # 111 # 11 # 11 ## ....

A turing machine is described with a binary string of 0's and 1's therefore the set of Turing machines forms a language. Each string of this language is the binary encode of a Turing machine.

E.g. Langue to TURING MACHINE

L_2 { # 1 # 1 # 11 # 1 , Turing machine 1

1 ## 1 ## 1 # 1111 Turing machine 2

....

}

6.2.2 Composite Turing Machine and Iterated Turing Machine

☞ **Write Short note on Composite and Iterated Turing Machine.**

- Two or more turing machine can be obtained to solve a collection of a simpler problems, so that the output of one Turing machine forms the input to the next Turing machine and so on. This is called as **composition.**
- This is used to break the complicated job into number of jobs implementing each separately and then combing them together to get answer for the job required to be done.

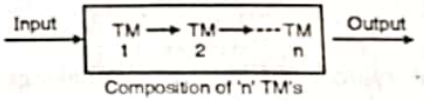


Fig. 6.2.2

- Another any having a combination in turing machine is by applying its own output as input repetitively. This is called **iteration or recursion**.

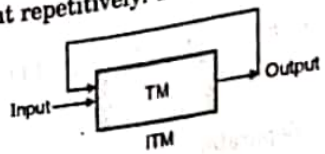


Fig. 6.2.3

Syllabus Topic : Introduction to Unsolvable Problems

6.3 Introduction to Unsolvable Problems

6.3.1 Recursive and Recursive Enumerable Languages

- We are well known about recursive function, how we will see about and R.E. languages.
- Language accepted by turing machine M is set of all strings of input symbols which are accepted by turing machine 'M'. for each string of I/P symbols there are three possibilities with 'M'.
  1. Turing machine will halt on final state (Accept)
  2. Turing machine will halt on non final state
  3. Turing machine will enter an infinite loop.

1. Recursive enumerable language

☞ **Write a short note on Recursive Enumerable Language.**

- A language L is recursive enumerable if there exists some truing machine 'M' to accept L.
- If there string of input symbols which belongs to language L will be accepted by turing machine M.

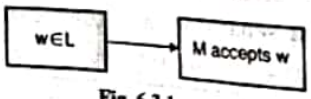


Fig. 6.3.1

- If string of input symbols which does not belongs to long L, the Turing machine will be either take half in non final state or enters in an infinite loop.

- All the languages which are accepted by turing machine are recursive enumerable.

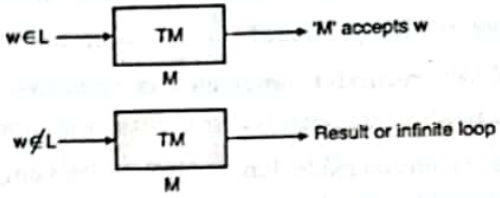


Fig. 6.3.2

2. Recursive language

A language L is recursive language which is decidable if some Turing machine M decides it. In this case M satisfies following conditions,

1. M accepts w if $w \in L$
2. M rejects w if $w \neq L$

In recursive language, Turing machine will decide the language is in particular format or not. For e.g. if there is CFL, so there are rules for recognizing it. turing machine will decide that language is CFL or not using all the symbols of CFLs.

So this language is recursive.

Every recursive language is also recursively enumerable (r.e)

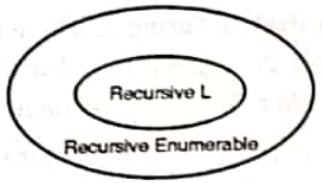


Fig. 6.3.3

If language is recursive then it is also recursively enumerable, but if the language is recursively enumerable language that we can not say it is recursive language.

Recursive and Recursive Enumerable languages are closed under intersection complementation and union. It's property states that,

1. If L is recursive, it's complement L′ is also recursive.
2. The union of two recursive languages is recursive and union of two recursive enumerable language is also recursive enumerable.
3. If L is recursive enumerable language and its compliment L′ is also recursive enumerable then L is recursive language.
4. If L_1 and L_2 are two recursive enumerable languages then $L_1 \cup L_2$ and $L_1 \cap L_2$ is also recursive enumerable.

Hence recursive languages is decidable, computable and solvable language

e.g. RL and CFL and Recursive Enumerable language known as Trurry recognizable or partially decidable or semi decidable language because if $w \notin L$, the Turing Machine either rejects or goes in loop.

6.3.2 Unsolvability

☞ **Explain the concept of unsolvability.**

☞ **What do you mean by decidability and undecidabgility ? Explain it with Examples.**

The Turing Machine plays an important role in computation of functions and relating classes. Even though Turing Machine computability is great, or solvable there is limitation of Turing Machine. Turing Machine faces decidable and undecidable or unsolvable problems of language.

Decidable problem finds solution, is definite means either yes or No, but, undecidable problem gives undecidable solution means sometime yes and sometime No.

Decidability and Undecidability is analogous to solvability and unsolvability e.g. Does earth move around the sun ? Does winter come after rainy season ? Solution of these problems are solvable or decidable and we can see another e.g. Will tomorrow be a rainy day ? You cannot give confirm

answer for this question. May or may not. It totally depends on nature. So these type of problems are undecidable or unsolvable.

Decidable problems in language like,

- Does FA accept regular language ?
- If L_1 and L_2 are two regular languages, are they closed under union, concatenation or kleene closure ?
- If L_1 and L_2 are two CFL, then $L_1 \cup L_2$ is CFL ?
- Similarly we can see undecidable problems in language like,
  1. For given CFG G is ambiguous or not?
  2. For two given CFL L_1 and L_2 whether $L_1 \cap L_2$ is CFL or not ?

Undecidable problems

The undecidable problem in computation can be understood by modifying or taking specific case and operating algorithms.

Theorem 1 :

There exists a language over Σ that is not recursively enumerable.

Proof :

Let, $L = T(M)$

Where, $L \rightarrow$ Recursive Enumerable language

$T(M) \rightarrow$ Turing Machine for M

$\Sigma \rightarrow$ Finite set of input

$\Sigma^* \rightarrow$ countable set of input (One to one correspondence between Σ^* and M)

As turing machine is defined with 7 tuple,

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

Where, M is encoded as a string and Q, Σ, Γ etc are finite set. So th set I of all Turing Machine is countable.

- Let l be set of all languages over Σ and $l \subset \Sigma^*$ which is infinite

i.e. l is uncountable or not in one to correspondence with N.

$$l = \{ L_1, L_2, \ldots \}$$ which is finite set

and $$\Sigma^* = \{ w_1, w_2, \ldots \}$$

such that any random language state L_i can be represented as infinite binary sequence,

as $X_{i1}, X_{i2} \ldots\ldots X_{in}$

where, $$X_{ij} = \begin{cases} 1 & w_j \in L_j \\ 0 & \text{otherwise} \end{cases}$$

Using this representation L_i as an infinite binary sequence.

$L_1 : x_{11}\ x_{12}\ x_{13} \ldots\ldots x_{1j} \ldots\ldots$

$L_2 : x_{21}\ x_{22}\ x_{23} \ldots\ldots x_{2j} \ldots\ldots$

:

$L_i : x_{i1}\ x_{i2}\ x_{i3} \ldots\ldots x_{ij} \ldots\ldots$

Representation of l.

Let $L \subset \Sigma^*$ defined by $y_1, y_2, y_3 \ldots\ldots$

Where $y_i = 1 - x_{ji}$

if $x_{ii} = 0$

$y_i = 1$

if $x_{ii} = 1$

$y_i = 0$

As per our assumption $L \subset \Sigma^*$ represented by infinite binary sequence should be L_k for random k natural number but $L \neq L_k$ since $w_k \in L$.

iff $w_k \notin L_k$

This contradicts our assumption that l is uncountable because I is countable.

As I is countable L should have some member not corresponding to any Turing Machine in I.

This proves the existence of language over Σ is not recursively enumerable.

We now prove a turing machine language, A_{TM}.

$A_{TM} = \{(M, w) \mid M \text{ is a turing machine and accepted } w\}$

Theorem 2 :

A_{TM} Machine is undecidable.

Proof :

We can prove that Alternating Turing Machine is undecidable by contradiction.

1. Let's we have to prove that Alternating Turing Machine is recursively enumerable or not construct a Turing Machine U which has input (M, w). Simulate M on w. If M enters in accepting state. Turing Machine U accepts input (M, w). So that we can say Alternating Turing Machine is recursively enumerable.
2. Now assume H be a Turing machines which decides Alternating Turing Machine and construct Turing Machine D that uses H as subroutine.

D = 1) Input (M)

2) Construct the string (M, (M))

3) Run H on input (M, (M))

a) If H accepts, then reject

b) If H rejects, then accept.

Now, D ((M)) described as follows,

$$D((M)) = \begin{cases} \text{accpet if M does not accept (M)} \\ \text{reject if M accepts (M)} \end{cases}$$

By referring action of D on the input (CD). According to the construction of D

$$D((D)) = \begin{cases} \text{accpet (D) does not accept by D} \\ \text{reject (D) accept by D} \end{cases}$$

Construction of D is stated contradiction hence Alternating Turing Machine is undecidable.

In this theorem Turing machine U is used in the sense of universal Turing machine since it is simulating any other Turing Machine.

Syllabus Topic : Halting Problem

6.4 Halting Problem of Turing Machine

☞ **Write a short note on Halting Problem.**

☞ **Prove A_{TM} = {(M,w)|The Turing machine M halts on input w} is undecidable.**

- In this section we introduce one more technique to prove the undecidability of halting problem in turing machine. Basically it asks a question "is it possible to tell whether a given machine will halt for some given input ?".
- There are two possibilities of Turing Machine configuration.
  1. After finite number of states Turing Machine M will halt.
  2. The Turing Machine M will never reaches to a halt state, despite its long runs.
- The halting problem is undecidable.

Suppose Turing Machine, H which decides whether computation of Turing Machine M will ever halt or given description d_M of M and the string w of M or not.

Then for every input (w, d_M) to H.

1. If M halts for input w, H is in accept state
2. If M does not halt for input w is in reject state.

Fig. 6.4.1

- Now consider Turing Machine D, which takes d_M as the input and proceeds as follows. Copy the input d_M duplicate d_M on its tape give input to fit with one modification, whenever H is in accept halt, D will loop forever. D calls to run on M.

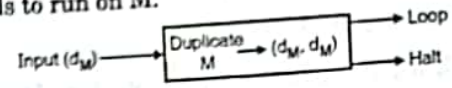


Fig. 6.4.2

For input $w = d_M$

1. Turing Machine B loops if M halts
2. Turing Machine B halts if M does not halt

- This can be described as,

$$D(M) = \begin{cases} \text{Accept if M does not accept (M)} \\ \text{Reject (If M accepts M)} \end{cases}$$

Since D itself behave such a M then replace D by M. Fig. 6.4.3 we get,

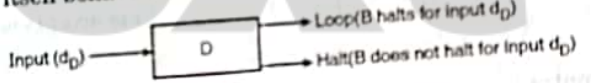


Fig. 6.4.3

Thus, D halts for input d_D if and only if D does not halt for halt for in d_D. This is a contradiction. Hence machine H can decide that any other Tur Machine Halt does not exist.

Theorem : A_{TM} = {(M, w)|The Turing machine M halts on input w undecidable.

Proof : A_{TM} is decidable and get contradiction.

Let, H be the algorithm or Turing Machine such that,

$$T(H) = ATM$$

Let, $$H(M, w) = \begin{cases} \text{Accept if M accepts w} \\ \text{Reject if M does not accept w} \end{cases}$$

We construct a Turing Machine D as follows,

1. For machine D, (M, w) is an input
2. The Turing Machine, H acts on (M, w)
3. If H rejects (M, w) then D rejects (M, w)
4. If H accepts (M, w), simulate the Turing Machine M on the input string w until M halts.
5. .. D accepts (M, w) only and only if M accepts w otherwise D rejects (M, w)

Turing Machine M halts on W when H accepts (M, w)

In this case either an accepting state q or state q′ such that δ (q′, a) is undefined till some symbol a in w is reached.

Cases : 1) D accepts (M, w)

2) D rejects (M, w)

This shows from definition of D that,

$$T(D) = \{ (M, w) \mid \text{The turing machine accepts w} \}$$
$$= A_{TM}$$

This is contradiction as A_{TM} is undecidable.

Hence Halting problem of turning machine is undecidable.

Examples :

1. Universal Turing machine is one of the uniform halting problems. It cannot determine Turing Machine halt or not for any arbitrary word.
2. Empty Turing Machine is also one of the unsolvable halting problems. It cannot determine whether or not M halts for input of the empty word.

❑❑❑