

SUBJECT CODE : 3160704

As per New Syllabus of
GUJARAT TECHNOLOGICAL UNIVERSITY
Semester - VI (CE / CSE)

THEORY OF COMPUTATION

Anuradha A. Puntambekar
M.E. (Computer)
Formerly Assistant Professor in
P.E.S. Modern College of Engineering,
Pune

SPECIMEN COPY



THEORY OF COMPUTATION

Subject Code : 3160704

Semester - VI (Computer Engineering / Computer Science & Engineering)

© Copyright with Author

All publishing rights (printed and ebook version) reserved with Technical Publications. No part of this book should be reproduced in any form, Electronic, Mechanical, Photocopy or any information storage and retrieval system without prior permission in writing, from Technical Publications, Pune.

Published by :



Amit Residency, Office No.1, 412, Shaniwar Peth,
Pune - 411030, M.S. INDIA Ph.: +91-020-24495496/97
Email : sales@technicalpublications.org Website : www.technicalpublications.org

Printer :

Yograj Printers & Binders
Sr. No. 10/1A,
Ghule Industrial Estate, Nanded Village Road,
Tal. - Havell, Dist. - Pune - 411041.

ISBN 978-93-90450-60-2



9789390450602

Course 1B

9789390450602 [1]

(ii)

PREFACE

The Importance of Theory of Computation is well known in various engineering fields. Overwhelming response to my books on various subjects inspired me to write this book. The book is structured to cover the key aspects of the subject Theory of Computation.

The book uses plain, lucid language to explain fundamentals of this subject. The book provides logical method of explaining various complicated concepts and stepwise methods to explain the important topics. Each chapter is well supported with necessary illustrations, practical examples and solved problems. All the chapters in the book are arranged in a proper sequence that permits each topic to build upon earlier studies. All care has been taken to make students comfortable in understanding the basic concepts of the subject.

Representative questions have been added at the end of each section to help the students in picking important points from that section.

The book not only covers the entire scope of the subject but explains the philosophy of the subject. This makes the understanding of this subject more clear and makes it more interesting. The book will be very useful not only to the students but also to the subject teachers. The students have to omit nothing and possibly have to cover nothing more.

I wish to express my profound thanks to all those who helped in making this book a reality. Much needed moral support and encouragement is provided on numerous occasions by my whole family. I wish to thank the Publisher and the entire team of Technical Publications who have taken immense pain to get this book in time with quality printing.

Any suggestion for the improvement of the book will be acknowledged and well appreciated.

*Author
A. A. Puntambekar*

Dedicated to God

SYLLABUS

Theory of Computation - (3160704)

Credits C	Examination Marks				Total Marks	
	Theory Marks		Practical Marks			
	ESE (E)	PA(M)	ESE (V)	PA(I)		
5	70	30	0	0	100	

1. Review of Mathematical Theory : Sets, Functions, Logical statements, Proofs, Relations, Languages, Principle of Mathematical Induction, Strong Principle, Recursive Definitions, Structural Induction. (Chapter - 1) - 10
2. Regular Languages and Finite Automata : Regular Expressions, Regular Languages, Application of Finite Automata, Automata with output - Moore machine & Mealy machine, Finite Automata, Memory requirement in a recognizer, Definitions, union- intersection and complement of regular languages, Non Deterministic Finite Automata, Conversion from NFA to FA, \wedge - Non Deterministic Finite Automata, Conversion of NFA - \wedge to NFA, Kleene's Theorem, Minimization of Finite automata, Regular And Non Regular Languages - pumping lemma. (Chapter - 2) - 25
3. Context free grammar (CFG) : Definitions and Examples, Unions Concatenations And Kleene's of Context free language, Regular Grammar for Regular Language, Derivations and Ambiguity, Unambiguous CFG and Algebraic Expressions, BackusNaur Form (BNF), Normal Form - CNF. (Chapter - 3) - 15
4. Pushdown Automata, CFL And NCFL : Definitions, Deterministic PDA, Equivalence of CFG and PDA & Conversion, Pumping lemma for CFL, Intersections and Complements of CFL, Non-CFL. (Chapter - 4) - 15
5. Turing Machine (TM) : TM Definition, Model Of Computation, Turing Machine as Language Acceptor, TM that Compute Partial Function, Church Turning Thesis, Combining TM, Variations Of TM, Non Deterministic TM, Universal TM, Recursively and Enumerable Languages, Context sensitive languages and Chomsky hierarchy. (Chapter - 5) - 15
6. Computable Functions : Partial - Total - Constant Functions, Primitive Recursive Functions, Bounded Minimalization, Regular function, Recursive Functions, Quantification, Minimalization, and μ -Recursive Functions, All Computable Functions Are μ -Recursive. (Chapter - 6)
7. Undecidability : A Language That Can't Be Accepted, and a Problem That Can't Be Decided, Non Recursive Enumerable (RE) Language - Undecidable Problem with RE - Undecidable Problems about TM - Undecidable Problems Involving Context-Free Languages, Post's Correspondence Problem, The Class P and NP. (Chapter - 7)

TABLE OF CONTENTS

Chapter - 1 Review of Mathematical Theory	(1 - 1) to (1 - 34)
1.1 Introduction	1 - 1
1.2 Sets	1 - 1
1.2.1 Operations on Set	1 - 2
1.2.2 Cartesian Product of Two Sets	1 - 3
1.2.3 Cardinality of Sets	1 - 3
1.3 Functions	1 - 4
1.3.1 Basic Terminologies	1 - 4
1.4 Logical Statements	1 - 6
1.5 Relations	1 - 9
1.6 Languages	1 - 12
1.7 Proofs	1 - 15
1.7.1 A Proof about Sets	1 - 15
1.7.2 Proof by Contradiction	1 - 17
1.7.3 Proof by Counter Example	1 - 18
1.8 Mathematical Induction	1 - 19
1.8.1 The Principle of Mathematical Induction (Regular Induction)	1 - 19
1.8.2 The Principle of Mathematical Induction (Strong Induction)	1 - 19
1.9 Recursive Definition	1 - 30
1.10 Short Questions and Answers	1 - 32
Chapter - 2 Regular Languages	(2 - 1) to (2 - 134)
2.1 Regular Expressions	2 - 1
2.2 Regular Languages	2 - 8
2.3 Introduction to Finite Automata	2 - 8
2.3.1 Representation	2 - 9
2.3.2 Control Over Input String	2 - 10
2.4 Memory Requirement In a Recognizer	2 - 11

2.5 Types of Finite Automata.....	2 - 11
2.6 Deterministic Finite Automata.....	2 - 12
2.7 Non Deterministic Finite Automata	2 - 25
2.8 NFA with ϵ	2 - 28
2.9 Conversion of NFA with ϵ to NFA without ϵ	2 - 30
2.10 Conversion of NFA to DFA	2 - 35
2.10.1 NFA to DFA	2 - 36
2.10.2 NFA with ϵ to DFA.....	2 - 42
2.11 Conversion of Regular Expression to FA (Kleen Theorem)	2 - 62
2.11.1 Direct Method for Conversion of FA to RE	2 - 66
2.12 Minimization of Finite Automata.....	2 - 75
2.13 Finite Automata with Output (Moore and Mealy Machine).....	2 - 85
2.13.1 Difference between Moore and Mealy Machine.....	2 - 90
2.13.2 Method for Conversion of Moore Machine to Mealy Machine	2 - 90
2.13.3 Method for Conversion of Mealy Machine to Moore Machine	2 - 97
2.14 Applications of Regular Expressions and Limitations of FA	2 - 104
2.15 Union, Intersection and Complement of Regular Languages	2 - 106
2.16 Pumping Lemma	2 - 122
2.17 Short Questions and Answers	2 - 133

Chapter - 3 Context Free Grammar	(3 - 1) to (3 - 50)
3.1 Definition of Context Free Grammar	3 - 1
3.2 Union, Concatenation and Kleen's CFG	3 - 2
3.3 Regular Grammar	3 - 15
3.3.1 Construction of Regular Grammar from Regular Expression	3 - 15
3.3.2 Right-linear and Left-linear Grammar	3 - 16
3.4 Derivation and Languages.....	3 - 18
3.5 Derivation Trees.....	3 - 22
3.5.1 Relationship between Derivation Trees and Derivation	3 - 24
3.6 Ambiguous CFG.....	3 - 25
3.7 Unambiguous CFG and Algebraic Expressions.....	3 - 28

3.8 Backus Naur Form (BNF).....	3 - 30
3.9 Simplification of Grammar.....	3 - 31
3.9.1 Removal of Useless Symbols.....	3 - 31
3.9.2 Elimination of ϵ Productions from Grammar.....	3 - 32
3.9.3 Removing Unit Productions	3 - 33
3.10 Normal Forms	3 - 35
3.10.1 Chomsky's Normal Form	3 - 35
3.11 Short Questions and Answers.....	3 - 48
Chapter - 4 Pushdown Automata, CFL and NCFL	(4 - 1) to (4 - 40)
4.1 Definition of Push Down Automata (PDA).....	4 - 1
4.2 Deterministic PDA.....	4 - 1
4.3 Equivalence of CFG and PDA	4 - 19
4.3.1 CFG to PDA.....	4 - 19
4.3.2 PDA to CFG	4 - 24
4.4 Non CFL.....	4 - 27
4.4.1 Pumping Lemma for CFL	4 - 27
4.5 Intersections and Complements of CFL.....	4 - 34
4.6 Short Questions and Answers.....	4 - 38
Chapter - 5 Turing Machine	(5 - 1) to (5 - 46)
5.1 Definiton of Turing Machine (TM)	5 - 1
5.2 Model of Computation	5 - 1
5.2.1 Instantaneous Description	5 - 2
5.3 Church's Turing Thesis	5 - 2
5.4 Computing Functions with TM	5 - 3
5.5 Combining TM.....	5 - 18
5.6 Variations of TM	5 - 21
5.7 Non Deterministic TM	5 - 36
5.8 Universal TM.....	5 - 37
5.9 Recursive and Recursively Enumerable Languages	5 - 39
5.9.1 Properties of Recursive and Recursively Languages.....	5 - 39

5.10 Context Sensitive Languages	5 - 40
5.11 Chomsky Hierarchy.....	5 - 43
5.12 Short Questions and Answers.....	5 - 45
Chapter - 6 Computable Functions	(6 - 1) to (6 - 14)
6.1 Partial, Total and Constant Functions.....	6 - 1
6.2 Primitive Recursive Functions.....	6 - 2
6.2.1 Class of Recursive Functions.....	6 - 4
6.3 Recursive Predicate and Bounded Minimalization.....	6 - 8
6.3.1 Bounded Quantifications	6 - 10
6.3.2 Bounded Minimalization,.....	6 - 11
6.4 Regular Function.....	6 - 11
6.5 μ -Recursive Functions.....	6 - 11
6.6 Short Questions and Answers.....	6 - 12
Chapter - 7 Undecidability	(7 - 1) to (7 - 16)
7.1 A Language that can't be Accepted and a Problem that can't be Decided	7 - 1
7.2 Non Recursive Enumerable (RE) Language.....	7 - 1
7.3 Undecidable Problem with RE	7 - 2
7.4 Undecidable Problems involving Context-Free Languages	7 - 6
7.5 Undecidable Problems about TM	7 - 8
7.5.1 Halting Problem	7 - 8
7.5.2 Post's Correspondence Problem	7 - 10
7.6 The Class P and NP	7 - 12
7.6.1 Example of P Class Problem	7 - 13
7.6.2 Example of NP Class Problem.....	7 - 15
Solved Model Question Paper	(M - 1) to (M - 4)

1

Review of Mathematical Theory

1.1 Introduction

Theory of computation is based on mathematical computations. These computations are used to represent various mathematical models. In this subject we will study many interesting models such as finite automata, push down automata, Turing machines. This subject is a fundamental subject and it is very close to the subjects like compilers, operating systems, system software and pattern recognition systems. The automata theory is the base of this subject. Automata theory is theory of models. Working principle of every process can be represented by the model. Such models can be a theoretical or mathematical models. The model helps in representing the concept of every activity.

In this chapter we will discuss basic concept of mathematical objects such as sets, logic functions and so on.

1.2 Sets

Set is defined as collection of objects. These objects are called elements of the set. All the elements are enclosed within curly brackets '{' and '}' and every element is separated by commas. If 'a' is an element of set A then we say that $a \in A$ and if 'a' is not an element of A then we say that $a \notin A$.

Typically set is denoted by a capital letter. The set can be represented using three methods.

1) Listing method

The elements are listed in the set.

For example : A set of elements which are less than 5. Then,

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

2) Describing properties

The properties of elements of a set define the set.

For example : A set of vowels. Hence here vowel is a property of the set which defines the set as :

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

(1 - 1)

3) Recursion method

The recursion occurs to define the elements of the set.

For example : $A = \{x | x \text{ is square of } n\}$ where $n \leq 10$.

This defines the set as :

$$A = \{0, 1, 4, 9, 16, \dots, 100\}$$

Subset : The subset A is called subset of set B if every element of set A is present in set B but reverse is not true. It is denoted by $A \subseteq B$. For example $A = \{1, 2, 3\}$ and $B = \{1, 2, 3, 4, 5\}$ then $A \subseteq B$.

Empty set : The set having no element in it is called empty set. It is denoted by $A = \{\}$ and it can be written as \emptyset (phi).

Null string : The null element is denoted by ϵ or \wedge character. Null element means no value character. But ϵ does mean \emptyset .

Power set : The power set is a set of all the subsets of its elements.

For example : $A = \{1, 2, 3\}$

Then power set : $Q = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$

The number of elements are always equal to 2^n where n is number of elements in original set. As in set A there are 3 elements. Therefore in power set Q there are $2^3 = 8$ elements.

Equal set : The two sets are said to be equal ($A = B$) if $A \subseteq B$ and $B \subseteq A$ i.e. every element of set A is an element of B and every element of B is an element of A.

For example :

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

$|A|$ denotes the length of set A. i.e. number of elements in set A.

For example : if

$A = \{1, 2, 3, 4, 5\}$ then

$$|A| = 5$$

1.2.1 Operations on Set

Various operations that can be carried out on set are -

- i) Union ii) Intersection
- iii) Difference iv) Complement

i) $A \cup B$ is union operation - If

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

$A \cup B = \{1, 2, 3, 4\}$ i.e. combination of both the sets.

ii) $A \cap B$ is intersection operation - If

$$A = \{1, 2, 3\} \quad \text{and} \quad B = \{2, 3, 4\} \quad \text{then}$$

$A \cap B = \{2, 3\}$ i.e. collection of common elements from both the sets.

iii) $A - B$ is the difference operation - If

$$A = \{1, 2, 3\} \quad \text{and} \quad B = \{2, 3, 4\} \quad \text{then}$$

$A - B = \{1\}$ i.e. elements which are there in set A but not in set B.

iv) \bar{A} is a complement operation - If

$$\bar{A} = U - A \quad \text{where } U \text{ is a universal set.}$$

For example :

$$\text{if} \quad U = \{10, 20, 30, 40, 50\}$$

$$A = \{10, 20\}$$

$$\text{then} \quad \bar{A} = U - A$$

$$= \{30, 40, 50\}$$

1.2.2 Cartesian Product of Two Sets

The cartesian product of two sets A and B is a set of all possible ordered pairs whose first component is member of A and whose second component is member of B. The cartesian product is denoted by $A \times B$.

$$\therefore A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}$$

For example : Let $A = \{a, b\}$ and $B = \{0, 1, 2\}$

then the cartesian product of A and B is,

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

1.2.3 Cardinality of Sets

The cardinality of the set is nothing but the number of members in the set.

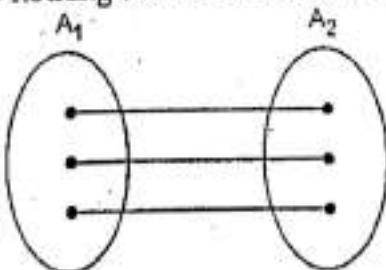


Fig. 1.2.1 Cardinality of two sets

These sets A_1 and A_2 have the same cardinality as there is one to one mapping of the elements of A_1 and A_2 . The different cardinalities of set can be - one to one, one to many, many to one, many to many.

1.3 Functions

Function can be defined as the relationship between two sets. That means using function we can use one element of one set to some other element of another set. A function is a relation but a relation is not necessarily a function.

A function can be denoted using a letter f . If $f(x) = x^3$ then we say that f of x equals to x cube, then we can have

$$\begin{aligned}f(2) &= 8 \\f(5) &= 125 \\f(-1) &= -1\end{aligned}$$

and so on.

1.3.1 Basic Terminologies

If D is a set then we can define function as,

$$f(D) = \{ f(x) \mid x \in D \}$$

If we map some element to some other element then it can be denoted as ,

$$f : D \rightarrow R \quad \text{i.e.} \quad x \rightarrow x^3$$

If set D consists of all the real number then

$D = \{1, 2, \dots\}$ is a domain.

The functions are denoted in terms of its mappings.

For example Let, $D = \{1, 2, 3, 4\}$ and $f(x) = x^3$.

Then the range of f will be $R = \{f(1), f(2), f(3), f(4)\}$

$$= \{1, 8, 27, 64\}$$

If we take Cartesian product of D and R then we obtain

$$F = \{(1, 1), (2, 8), (3, 27), (4, 64)\}$$

Thus a function can be represented using Cartesian product of its domain and range.

→ Example 1.3.1 : Find the domain and range of the following relation. Is this relation a function $\{(2, 9), (3, 14), (4, 21)\}$

Solution : In above given set a relationship between certain x 's and y 's is a relation. Hence all the x values denote the domain and all the y values denote range. Therefore -

$$\text{domain} : \{2, 3, 4\}$$

Range : { 9, 14, 21 }

We can observe the x and y pair to get the relationship. Note that, if $x = 2$ then $y = x^2 + 5 = (2)^2 + 5 = 9$. This holds true for all the x and y pair in given set. Hence we can define a relation as a function as $f(x) = x^2 + 5$.

⇒ Example 1.3.2 : Determine the domain of the given function :

$$y = \frac{x^2 + x - 5}{x^2 + 3x - 10}$$

Solution : Here domain means all the values of x that are allowed to take. For this function, the only care that has to be taken is that, the function should not produce divide by zero error. If the denominator equals to zero, then divide by zero error will occur, hence

$$x^2 + 3x - 10 = 0$$

$$(x+5)(x-2) = 0$$

$$\therefore x = -5 \text{ or } x = 2$$

Hence domain will be all those values of x that are not equal to -5 or 2.

⇒ Example 1.3.3 : Define one-to-one and onto functions. Also explain compositions and inverse of functions.

GTU : Summer-14, 16, Marks 7

Solution : Refer answer of example 1.3.4 for one-to-one and onto functions

Compositions of functions - The composition of functions is a function in which result of one function is given as input to another function. It is denoted by gof or fog . The notation gof is read as "g circle f" or "g composed with f", or "gof f".

For example

If $f(x) = x + 10$ and $g(x) = x - 4$ then

$$\begin{aligned} i) \quad fog(x) &= f(g(x)) = g(x) + 10 \\ &= (x - 4) + 10 \end{aligned}$$

$$fog(x) = x - 6$$

$$\begin{aligned} ii) \quad gof(x) &= g(f(x)) = f(x) - 4 \\ &= (x + 10) - 4 \end{aligned}$$

$$gof(x) = x - 6$$

Inverse of functions -

The inverse of function is a function which is reversal of whatever the original function does as input. It is denoted as $f^{-1}(x)$.

For example :

$$f(x) = x + 50 \text{ then } f^{-1}(x) \text{ is } f'(x) = x - 50.$$

→ Example 1.3.4 : Define one-to-one, onto and bijection function.

Check whether the function $f: R^+ \rightarrow R, f(x) = x^2$ is one to one and onto.

Solution : i) One-to-one : A function $f: A \rightarrow B$ is said to be one to one mapping when any element of A can be mapped with at most one argument of B. For example :

ii) Onto : A function $f: A \rightarrow B$ is said to be onto mapping when every element of A must be mapped with at least one argument of B. For example

iii) Bijection function : A function $f: A \rightarrow B$ is said to be bijective if every element of A can be mapped with exactly one element of B. For example :

For the set of natural numbers N to the set of non negative even number $f: R^+ \rightarrow R, f(x) = x^2$ is one to one and onto.

For instance :

$$f(2) = 4$$

GTU : Summer-11, Marks 7

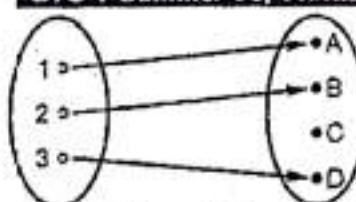


Fig. 1.3.1

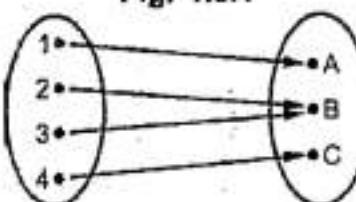


Fig. 1.3.2

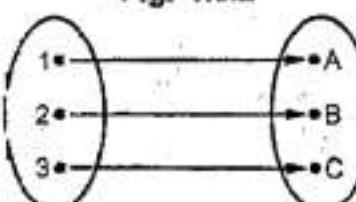


Fig. 1.3.3

1.4 Logical Statements

Logic is a study of reasoning . In the study of formal languages mathematical logic is represented by propositions.

The propositions or statements are declarative sentences. which can be either true or false but not both.

For example :

I am proud of my country.

I like cricket.

Sugar is hot to taste.

You are always welcome.

These are propositions with true or false values. We can join two propositions by 'and', 'but', 'if', 'or'. These are called connectives.

Various connectives that are used are

- | | |
|------------------|-----------------------------------|
| i) Negation | ii) Conjunction |
| iii) Disjunction | iv) Implication v) if and only if |

Let us discuss each one by one -

i) Negation

This proposition is also called NOT proposition. It is denoted by \neg or \sim or having horizontal bar on letter.

If S is a proposition then $\neg S$ defines NOT S . The truth table for this connective is

S	$\neg S$
T	F
F	T

ii) Conjunction (AND)

If A and B are two propositions then conjunction of A and B is denoted by $A \wedge B$. The truth table for this connective is

A	B	$A \wedge B$
T	T	T
T	F	F
F	T	F
F	F	F

iii) Disjunction (OR)

If A and B are two propositions then disjunction of A and B is denoted by $A \vee B$. The truth table for this connective is

A	B	$A \vee B$
T	T	T
T	F	T
F	T	T
F	F	F

iv) Implication

If A and B are two propositions then if A then B is a proposition. It can be denoted by $A \rightarrow B$ or $A \Rightarrow B$.

The truth table for this connective is

A	B	$A \rightarrow B$
T	T	T
T	F	F
F	T	T
F	F	T

For example : Find truth value for following proposition.

"If an apple is not green then it must be red".

Let, A denotes "An apple is not green" and B denotes "It must be red". The statement A is false and B is true. Hence the proposition is true.

v) If and only If

If A and B are two propositions then A if and only if B. It can be denoted as $A \leftrightarrow B$. The truth table will be

A	B	$A \leftrightarrow B$
T	T	T
T	F	F
F	T	F
F	F	T

For example : "I will go to the school if and only if it is not raining". Here statement A is "I will go to the school". The statement B is "It is not raining". Thus the resultant proposition is false.

→ Example 1.4.1 : Draw truth table for following logic formula : $P \rightarrow (\neg P \vee \neg Q)$. Is it a tautology ? A contradiction ? Or neither ? Justify your answer.

GTU : Summer-17, Marks 3

Solution :

P	Q	$\neg P$	$\neg Q$	$\neg P \vee \neg Q$	$P \rightarrow (\neg P \vee \neg Q)$
T	T	F	F	F	F
T	F	F	T	T	T
F	T	T	F	T	T
F	F	T	T	T	T

1.5 Relations

Relationship is a major aspect between two objects, even this is true in our real life. One object can be related with the other object by some relation. Then those two objects form a pair based on this certain relationship.

Definition : The relation R is a collection for the set S which represents the pair of elements.

For example : (a, b) is in R. We can represent their relation as $a R b$. The first component of each pair is chosen from a set called domain and second component of each pair is chosen from a set called range.

Properties of Relations

A relation R on set S is

1. Reflexive if iRi for all i in S.
2. Irreflexive if iRi is false for all i in S.
3. Transitive if iRj and jRk imply iRk .
4. Symmetric if iRj implies jRi .
5. Asymmetric if iRj implies that jRi is false.

Every asymmetric relation must be irreflexive.

For example : if $A = \{a, b\}$ then

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

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

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

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

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

Equivalent Relation

A relation is said to be equivalence relation if it is reflexive, symmetric and transitive, over some set S.

Suppose R is a set of relations and S is the set of elements.

For example : S is the set of lines in a plane and R is the relation of lines intersecting to each other.

A relation is said to be antisymmetric if $(a, b) \in R$ and $(b, a) \in R$ then $a = b$.

→ **Example 1.5.1 :** Determine whether R is equivalence relation or not where

$$A = \{0, 1, 2\}, R = \{(0, 0), (1, 0), (1, 1), (2, 2), (2, 1)\}$$

Solution : The R is reflexive because $(0, 0), (1, 1), (2, 2) \in R$.

Whereas R is not symmetric because $(0, 1)$ is not in R whereas $(1, 0)$ is in R .

Hence the R is not an equivalence relation.

Closures of Relations

Sometimes when the relation R is given, it may not be reflexive or transitive.

By adding some pairs we make the relation as reflexive or transitive.

For example : Let $\{ (a, b), (b, c), (a, a), (b, b) \}$

Now this relation is not transitive because $(a, b), (b, c)$ is there in relation R but (a, c) is not there in R so we can make the transitive closure as

$\{ (a, a), (b, b), (a, b), (b, c), (a, c) \}$

We can even define reflexive closure and symmetric closure in the same way.

→ **Example 1.5.2 :** In the given relation determine the properties (reflexivity, symmetry, transitivity), which ones the relation has : $R = \{(1,1), (2,2), (3,3), (1,2)\}$ and $R = \emptyset$

GTU : Summer-13, Marks 7

Solution : In the given relation for set $= \{1, 2, 3\}$ the relation is $R = \{(1,1), (2,2), (3,3), (1,2)\}$. It is clear that R consists of all the elements of the set which are related to itself i.e. $(1,1)$, $(2,2)$ and $(3,3)$. So the given relation is reflexive relation. If $R = \emptyset$ then it is not reflexive but it is symmetric and transitive.

→ **Example 1.5.3 :** Given the relation R in A as $R = \{(1, 1), (2, 2), (2, 3), (3, 2), (4, 2), (4, 4)\}$ is R (a) Reflexive (b) Symmetric (c) Transitive (d) Antisymmetric ?

GTU : Winter-14, Marks 3

Solution : 1) R is not reflexive as $(3, 3) \notin R$.

2) Not As $(4, 2) \in R$ and $(2, 4) \notin R$

3) Yes R is transitive

4) No. As $(2, 2) \in R, (2, 3) \in R$ but $2 \neq 3$.

→ **Example 1.5.4 :** Find the transitive closure and the symmetric closure of the relation. $\{ (1, 2), (2, 3), (3, 4), (5, 4) \}$

Solution : Given relation is $\{ (1, 2), (2, 3), (3, 4), (5, 4) \}$ then the transitive closure of relation is $\{ (1, 2), (2, 3), (1, 3), (3, 4), (2, 4), (5, 4) \}$

and symmetric closure is

$\{ (1, 2), (2, 1), (2, 3), (3, 2), (3, 4), (4, 3), (5, 4), (4, 5) \}$

→ **Example 1.5.5 :** Consider a relation $R = \{(1, 1), (2, 2), (3, 4), (4, 3)\}$ defined on set $A = \{1, 2, 3, 4\}$. Obtain R^* (reflexive and transitive closure of R).

Solution : Let, $R = \{(1, 1), (2, 2), (3, 4), (4, 3)\}$ then reflexive and transitive closure of R , i.e. R^* is
 $= \{(1, 1), (2, 2), (3, 4), (3, 3), (4, 3), (4, 4)\}$

→ **Example 1.5.6 :** Let $A = \{1, 2, 3, 4, 5, 6\}$ and R be a relation on A such that aRb iff a is a multiple of b . Write R . Check if the relation is
 i) Reflexive ii) Symmetric iii) Asymmetric iv) Transitive

GTU : Winter-16, Marks 7

Solution :

$$R = \{(a, b) : a = i * b\} \text{ where } 1 \leq i <= 6$$

$$R = \{(1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (2, 1), (4, 2), (6, 2), (6, 3)\}$$

- i) R is reflexive as $(a, a) \in R$
- ii) It is not symmetric because $(2, 1) \in R$ but $(1, 2) \notin R$
- iii) It is asymmetric because $(a, b) \in R$ but there is no $(b, a) \in R$
- iv) It is transitive because as if $(a, b) \in R$ and $(b, c) \in R$ then $(a, c) \in R$

Consider $(6, 2) \in R$, $(2, 2) \in R$ Hence $(6, 2) \in R$

→ **Example 1.5.7 :** Define reflexivity, symmetry and transitivity properties of relations.

GTU : Summer-17, Marks 3

Solution : A relation R on set S is

- i) Reflexive if iRi for all i in S
- ii) Symmetric if iRj implies jRi
- iii) transitive if iRj and jRk imply iRk

→ **Example 1.5.8 :** Consider the relation $R = \{(1, 2), (1, 1), (2, 1), (2, 2), (3, 2), (3, 3)\}$ defined over $\{1, 2, 3\}$. Is it reflexive? Symmetric? Transitive? Justify each of your answers.

GTU : Summer-17, Marks 3

Solution : i) Yes it is reflexive

- ii) No. As $(3, 2) \in R$ $(2, 3) \notin R$
- iii) No. As $(3, 2) \in R$ $(2, 1) \in R$ but $(3, 1) \notin R$

→ **Example 1.5.9 :** Define onto function. In each case, a relation on the set $\{1, 2, 3\}$ is given. Of the three properties, reflexivity, symmetry and transitivity, determine which ones the relation has. Give reasons.

- a. $R = \{(1, 3), (3, 1), (2, 2)\}$
- b. $R = \{(1, 1), (2, 2), (3, 3), (1, 2)\}$
- c. $R = \emptyset$

GTU : Summer-18, Marks 4

Solution : a. $R = \{(1, 3), (3, 1), (2, 2)\}$

- i) R is not reflexive as $(1, 1), (3, 3) \notin R$.
- ii) R is symmetric as $R \in (1, 3)$ and $(3, 1)$.
- iii) R is not transitive as $R \in (1, 3)$ and $(3, 1)$ but $\notin (1, 1)$.
- b. $R = \{(1,1), (2,2), (3,3), (1,2)\}$

It is clear that R consist of all the elements of the set which are related to itself i.e. $(1,1), (2,2)$ and $(3,3)$. So the given relation is reflexive relation. If $R = \emptyset$ then it is not reflexive but it is symmetric and transitive.

- c. $R = \emptyset$

R is symmetric and transitive but not reflexive. The \emptyset is empty set.

- 1) R is symmetric if $(x, y) \in R$ then $(y, x) \in R$. But as there is no (x, y) in R there is no (y, x) as well. Hence R is symmetric.
- 2) Similarly if $(x, y) \in R, (y, z) \in R$ then $(x, z) \in R$. But due to empty set $(x, y), (y, z)$ and (x, z) all are not present. Hence R is transitive.
- 3) But as $R \notin (x, x), (y, y), (z, z)$; R is not reflective.

Example 1.5.10 : Define - Equivalence relation. A relation on the set {1, 2, 3} is given as $R = \{(a, b) | a-b \text{ is an even no}\}$. Check whether R is equivalence relation or not. Give reasons.

GTU : Winter-19, Marks 4

Solution : Equivalence relation : Refer section 1.5.

Possible pairs : $(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3)$

Among these pairs : $(1,1), (1,3), (2,2), (3,1), (3,3)$ are the pairs where $|a-b|$ is even.

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

Reflexive : Since $(1,1), (2,2), (3,3)$ are the pairs where $|a-b|$ is even.

Symmetric : $(1,3)$ and $(3,1) \in R$ therefore for every a, b $aRb \Rightarrow bRa \in R$ Hence the relation is symmetric

Transitive : $(1,1) \in R$ and $(1,3) \in R \Rightarrow (1,3)$ that means aRb and $bRc \Rightarrow aRc$ for every pair of a, b and c . Hence relation is transitive.

Therefore R is equivalence relation.

1.6 Languages

Alphabet : An alphabet is a finite collection of symbols. We cannot define symbol formally.

The example of alphabet is,

$$S = \{a, b, c, \dots, z\}$$

The elements a, b, \dots, z are the alphabets.

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

Here 0 and 1 are alphabets, denoted by W.

String : String is finite collection of symbols from alphabet.

For example, if $\Sigma = \{a, b\}$ then various strings that can be formed from Σ are {ab, ab, aa, aaa, bb, bbb, ba, aba ...}. An infinite number of strings can be generated from this set.

- The empty string can be denoted by ϵ .
- The prefix of any string is any number of leading symbols of that string and suffix is any number of trailing.

Symbols :

For example, in string "0011" the prefixes can be 0, 00, 001. Similarly suffixes can be 1, 11, 011.

In string "Mango" the prefix can be 'Man' and the suffix can be 'go'.

Languages in abstract

The abstract languages refer to an abstract mathematical notion which generalizes the characteristics that are common to the regular languages, context free languages and recursively enumerable languages.

Language : The language is a collection of appropriate strings.

The language is defined using an input set. The input set is typically denoted by letter Σ .

For example : $\Sigma = \{\epsilon, 0, 00, 000, \dots\}$

Here the language L is defined as 'any number of Zeros'.

Note that language is formed by appropriate strings and strings are formed by alphabets.

Operations on string

Various operations that can be carried out on strings are

1. Concatenation :

In this operation two strings are combined together to form a single string.

For example, $x = \{\text{white}\}$ $y = \{\text{house}\}$

Then the concatenated string will be $xy = \{\text{white house}\}$.

2. Transpose :

The transpose of operation is also called reverse operation.

For example, $(xa)^T = a(x)^T$.

If $(ababbb)^T = (bbbaba)$.

3. Palindrome :

Palindrome of string is a property of a string in which string can be read same from left to right as well as from right to left.

For example, the string "MadaM" is palindrome because it is same if we read it from left to right or from right to left.

Operations on language

Language is collection of strings. Hence operations that can be carried out on strings are the operations that can be carried out on language.

If L_1 and L_2 are two languages then,

- Union of two languages is denoted as $L_1 \cup L_2$.
- Concatenation of two languages is denoted by $L_1 L_2$.
- Intersection of two languages is denoted by $L_1 \cap L_2$.
- Difference of two language is denoted by $L_1 - L_2$.

► Example 1.6.1 : Consider the string $X = 110$ and $y = 0110$ then find.

- xy
- yx
- x^2
- $\in y$

Solution : Let $x = 110$ and $y = 0110$ then

- xy is the concatenation operation. Hence we will concatenate the two strings from x and y respectively.
 $xy = 1100110$.
- yx suggest the concatenation of string from set y with string from x hence
 $yx = 0110110$.
- $x^2 = x.x$. Hence concatenate the string from x with the string from set x itself.
Hence $x^2 = 110110$.
- $\in y$ means the string belonging to set y which is 0110 .

► Example 1.6.2 : Describe the following language over the input set $A = \{a, b\}$.

- $L_1 = \{a, ab, ab^2\}$
- $L_2 = \{a^n b^n \mid n \geq 1\}$
- $L_3 = \{a^n b^n \mid n > 0\}$

Solution : The language is defined over the set $A = \{a, b\}$.

- L_1 consists of all the strings with letter a followed by any number of b 's.
- L_2 consists of all the strings having equal number of a 's and equal number of b 's such that all the a 's are followed by all the b 's.
- L_3 consists of all the strings with any numbers of a 's followed by at least one b .

1.7 Proofs

We will discuss various forms of proofs with the help of some examples. We will discuss

1. Proofs about sets.
2. Proofs by contradiction.
3. Proofs by counter example.

1.7.1 A Proof about Sets

The set is a collection of elements or items. By giving proofs about the sets we try to prove certain properties of the sets.

For example if there are two expressions A and B and we want to prove that both the expressions A and B are equivalent then we need to show that the set represented by expression A is same as the set represented by expression B. Let $P \cup Q = Q \cup P$ if we map expression A with $P \cup Q$ and expression B with $Q \cup P$ then to prove $A = B$ we need to prove $P \cup Q = Q \cup P$.

This proof is of the kind "if and only if" that means an element x is in A if and only if it is in B. We will make use of sequence of statements along with logical justification in order to prove this equivalence.

Let us prove $P \cup Q = Q \cup P$.

Proving L.H.S.

Sr. No.	Statement	Justification
1.	x is in $P \cup Q$	Given
2.	x is in P or x is in Q	1) And by definition of union
3.	x is in Q or x is in P	2) And by definition of union
4.	x is in $Q \cup P$	3) Rule (2) And by definition of union

Proving R.H.S.

Sr. No.	Statement	Justification
1.	x is in $Q \cup P$	Given
2.	x is in Q or x is in P	1) And by definition of union
3.	x is in P or x is in Q	2) And by definition of union
4.	x is in $P \cup Q$	3) Rule (2) And by definition of union

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

⇒ Example 1.7.1 : Prove that set of all integers is countably infinite.

Solution : A set is countably infinite if it is countable and infinite. And a set is said to be countable if it can be placed in 1-1 correspondence with +ve integers. For all the positive integers just take smallest number in the set and place it first in the list. Then take next smallest element and place it second in the list. Continue this process for all the positive integers. This newly formed set can be mapped with positive integers.

For instance : Consider the set of even numbers {0, 2, 4, 6, 8, 10, ...}. It can be mapped with integers as,

$$\begin{aligned} 0 &\rightarrow 1 \\ 2 &\rightarrow 2 \\ 4 &\rightarrow 3 \\ 6 &\rightarrow 4 \\ 8 &\rightarrow 5 \\ 10 &\rightarrow 6 \end{aligned}$$

and so on.

This shows that set of all integers is countably infinite.

⇒ Example 1.7.2 : Show that any subset of a countable set is countable.

Solution :

Proof : A set S is called countable if there exists an injective function

$$f : S \rightarrow N$$

where N is a set of natural numbers {0, 1, 2, ...}. For any subset of a countable set, if we could map the injective function then it is said to be countable. In other words, for the members of any subset of a countable set if we map a one to one relation with the natural numbers then that subset is countable.

For example - A set of prime numbers is countable. Now consider a subset of this set of prime numbers i.e.

$$A = \{2, 3, 5, 7, 11, 13\}$$

$$\begin{aligned} 2 &\text{ maps to } 1 \\ 3 &\text{ maps to } 2 \\ 5 &\text{ maps to } 3 \\ 7 &\text{ maps to } 4 \\ 11 &\text{ maps to } 5 \\ 13 &\text{ maps to } 6 \end{aligned}$$

Thus we can map the subset to the set of natural numbers. Hence it is proved that any subset of countable set is countable.

→ Example 1.7.3 : Prove for any sets A, B and C if $A \cap B = \emptyset$ and $C \subseteq B$, then $A \cap C = \emptyset$.

Solution : We can use vein diagrams for this prove.

L.H.S. : $A \cap B = \emptyset$ can be represented using vein diagram as follows -

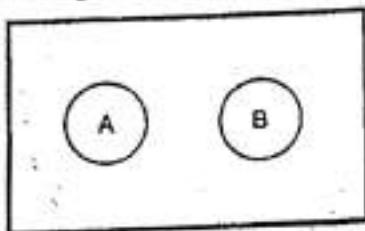


Fig. 1.7.1 $A \cap B = \emptyset$ as no area is shaded

R.H.S. : $C \subseteq B$ can be as shown below.

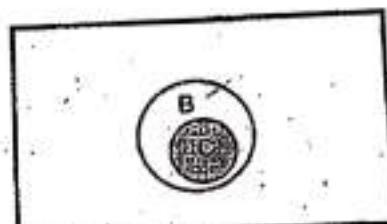


Fig. 1.7.2 $C \subseteq B$ shaded portion represent the subset

Thus we get $A \cap C$ as follows

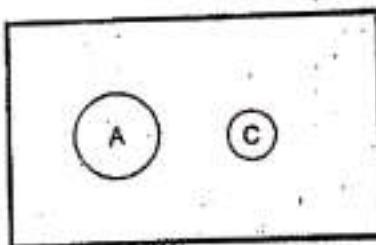


Fig. 1.7.3 $A \cap C = \emptyset$

Thus the above vein diagram represents $A \cap C \neq \emptyset$ is true.

1.7.2 Proof by Contradiction

In this type of proof, for the statement of the form if A then B we start with statement A is not true and thus by assuming false A we try to get the conclusion of statement B. When it becomes impossible to reach to statement B we contradict ourself and accept that A is true.

→ Example 1.7.4 : Prove $P \cup Q = Q \cup P$.

Solution :

- Initially we assume that $P \cup Q = Q \cup P$ is not true. That is $P \cup Q \neq Q \cup P$.
- Now consider that x is in Q , or x is in P . Hence we can say x is in $P \cup Q$ (according to definition of union).
- But this also implies that x is in $Q \cup P$ according to definition of union.

- Hence the assumption which we made initially is false.
- Thus $P \cup Q = Q \cup P$ is proved.

Example 1.7.5 : Prove that $\sqrt{2}$ (square root of 2) is irrational by method of contradiction.

Solution : We assume that $\sqrt{2}$ is rational. That means it can be written as the ratio of two integers m and n as

$$\sqrt{2} = \frac{m}{n} \quad \dots (1)$$

Where m and n have no common factors. Squaring on both sides of equation (1) we get -

$$2 = \frac{m^2}{n^2} \quad \dots (2)$$

$$\text{i.e. } m^2 = 2n^2 \quad \dots (3)$$

This shows that m^2 is even. This is true only when m itself is even. But then m^2 is actually divisible by 4. Hence n^2 and therefore n must be even. So m and n are both even which is contradiction to our assumption that they have no common factors. The square root of 2 can not be rational.

1.7.3 Proof by Counter Example

In order to prove certain statements, we need to see all possible conditions in which that statement remains true. There are some situations in which the statement cannot be true. For example -

There is no such pair of integers such that

$$a \bmod b = b \bmod a$$

Proof : Consider $a = 2$ and $b = 3$. Then clearly

$$2 \bmod 3 \neq 3 \bmod 2$$

Thus the given pair is true for any pair of integers but if $a = b$ then naturally

$$a \bmod b = b \bmod a$$

Thus we need to change the statement slightly. We can say

$$a \bmod b = b \bmod a, \text{ when } a = b.$$

This type of proof is called counter example. Such proof is true only at some specific condition.

Example 1.7.6 : Prove for any integer a and b if a and b are odd, then ab (i.e. Product of two integer) is odd.

Solution : Let any integer a or b if it is odd then it can be represented as $2n+1$.

Hence product of two odd integers

$$\begin{aligned}
 &= (2n + 1) * (2n + 1) \\
 &= 4n^2 + 4n + 1 \\
 &= [4n(n + 1)] + 1 \\
 &= 2(2n(n + 1)) + 1 \\
 &= 2k + 1 \quad \because k = 2n(n + 1). \\
 &= \text{odd integer.}
 \end{aligned}$$

Thus product of any two odd integers is odd is proved.

1.8 Mathematical Induction

Inductive proofs are special proofs based on some observations. It is used to make recursive definitions.

1.8.1 The Principle of Mathematical Induction (Regular Induction)

If for any statement involving a positive integer, n the following are true

1. The statement holds for $n = 1$ and
2. Whenever the statement holds for $n = k$, it must also hold for $n = k + 1$

Then the statement holds for all positive integers, n .

1.8.2 The Principle of Mathematical Induction (Strong Induction)

If for any statement, involving a positive integer n , the following are true

1. The statement holds for $n = 1$ and
2. Whenever the statement holds for all $n \leq k$, it must also hold for $n = k + 1$.

Then the statement holds for all positive integers, n .

If strong induction holds, then regular induction holds.

The only difference between regular induction and strong induction is that in strong induction you assume that every number up to k satisfies the condition that you wish to prove whereas in regular induction, you only assume that some integer k satisfies this condition.

Thus strong induction is an induction where you assume all previous cases satisfy the induction hypothesis.

⇒ Example 1.8.1 : Prove : $1 + 2 + 3 \dots + n = \frac{n(n+1)}{2}$

GTU : Summer-15, Winter-18, Marks 7

Solution : Initially,

1) Basis of induction -

Assume, $n = 1$. Then,

$$\text{L.H.S} = n$$

$$= 1$$

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

$$= \frac{2}{2}$$

$$= 1$$

Now,

2) Induction hypothesis -

Now we will assume $n = K$ and will obtain the result for it. The equation then becomes,

$$1 + 2 + 3 + \dots + K = \frac{K(K+1)}{2} \quad \dots (1)$$

3) Inductive step -

Now we assume that equation is true for $n = K$. And we will then check if it is also true for $n = K + 1$ or not.

Consider the equation assuming $n = K + 1$

$$\text{L.H.S.} = \frac{1 + 2 + 3 \dots + K}{2} + K + 1 \quad \because \text{equation (1)}$$

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

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

$$= \frac{(K+1)(K+2)}{2} \quad \because \text{taking common factor}$$

$$\text{i.e.} = \frac{(K+1)(K+1+1)}{2}$$

$$= \text{R.H.S.}$$

⇒ Example 1.8.2 : Prove : $n! > 2^{n-1}$

Solution : Consider,

1. Basis of induction - Let $n = 1$ then L.H.S. = 1 R.H.S. = $2^{1-1} = 2^0 = 1$ hence $n! > 2^{n-1}$ is proved.

2. Induction hypothesis - Let $n = n + 1$ then

$$k! = 2^{k-1} \text{ where } k > = 1$$

$$\begin{aligned}\text{then } (k+1)! &= (k+1)k! \text{ by definition of } n! \\ &= (k+1)2^{k-1} \\ &= 2 \times 2^{k-1} \\ &= 2^k\end{aligned}$$

→ Example 1.8.3 : Prove that $1^2 + 2^2 + 3^2 + \dots + n^2 = \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$

using mathematical induction.

GTU : Winter-12, Summer-12, Marks 7

Solution : We will first prove it by basis of induction and then will consider induction hypothesis.

1. Basis of induction -

$$\text{Let } n = 1$$

$$\text{L.H.S.} = 1^2 = 1$$

$$\text{R.H.S.} = \frac{1(1+1)(2 \cdot 1 + 1)}{6}$$

$$= \frac{6}{6}$$

$$\text{R.H.S.} = 1$$

As L.H.S. = R.H.S it is proved for $n = 1$.

2. Induction hypothesis - Let, $n = K$.

Then

$$1^2 + 2^2 + \dots + K^2 = \sum_{i=1}^K i^2 = \frac{K(K+1)(2K+1)}{6} \text{ is true} \quad \dots (1)$$

Now, we will try to prove it for $n = K + 1$. Hence we will substitute n by $K + 1$.

$$\begin{aligned}\therefore 1^2 + 2^2 + 3^2 + \dots + (K+1)^2 &= \sum_{i=1}^{K+1} i^2 = \frac{(K+1)(K+1+1)(2(K+1)+1)}{6} \\ \text{L.H.S.} &= 1^2 + 2^2 + 3^2 + \dots + K^2 + (K+1)^2\end{aligned}$$

We will substitute equation (1) and we will get,

$$\begin{aligned}
 &= \frac{K(K+1)(2K+1)}{6} + (K+1)^2 \\
 &= (K+1) \left[\frac{K(2K+1)}{6} + (K+1) \right] \\
 &= (K+1) \left[\frac{K(2K+1) + 6K + 6}{6} \right] \\
 &= (K+1) \left[\frac{2K^2 + K + 6K + 6}{6} \right] \\
 &= (K+1) \left[\frac{2K^2 + 7K + 6}{6} \right] \\
 &= \frac{(K+1)(2K^2 + 4K + 3K + 6)}{6} \\
 &= \frac{(K+1)(2K(K+2) + 3(K+2))}{6} \\
 &= \frac{(K+1)((2K+3)(K+2))}{6} \\
 &= \frac{(K+1)((K+2)(2K+3))}{6} \\
 &= \frac{(K+1)(K+1+1)(2(K+1)+1)}{6} \\
 &= \text{R.H.S.}
 \end{aligned}$$

As L.H.S. = R.H.S., it is true that using $n = K + 1$ the given expression can be proved.

Example 1.8.4 : Prove : $1+4+7+\dots+(3n-2) = \frac{n(3n-1)}{2}$ for $n > 0$

Solution : Consider,

1. Basis of induction -

Let, $n = 1$,

$$\text{L.H.S.} = (3.1-2)$$

$$= 1$$

$$\text{R.H.S.} = \frac{1(3.1-1)}{2} = \frac{2}{2} = 1$$

As L.H.S. = R.H.S. given expression is true for $n = 1$.

2. Induction hypothesis - We assume that the given expression is true for $n = K$.

i.e. $1+4+7+\dots+(3K-2) = \frac{K(3K-1)}{2}$ is true. ... (1)

Now we will prove it for $n = K + 1$.

$$\text{L.H.S.} = 1+4+7+\dots+(3K-2)+(3(K+1)-2)$$

We will substitute R.H.S. of equation (1).

$$\begin{aligned} &= \frac{K(3K-1)}{2} + (3(K+1)-2) \\ &= \frac{K(3K-1) + 2[3(K+1)-2]}{2} \\ &= \frac{3K^2 - K + 6K + 2}{2} \\ &= \frac{3K^2 + 5K + 2}{2} \\ &= \frac{3K^2 + 2K + 3K + 2}{2} \\ &= \frac{(2K+2) + (3K^2 + 3K)}{2} \\ &= \frac{2(K+1) + 3K(K+1)}{2} \\ &= \frac{(K+1)(3K+2)}{2} \\ &= \frac{(K+1)(3(K+1)-1)}{2} \\ &= \text{R.H.S.} \end{aligned}$$

As L.H.S. = R.H.S. the given expression is true for $n = K + 1$.

→ Example 1.8.5 : Prove : $2^n > n^3$ where $n \geq 10$.

GTU : Winter-14, Marks 7

Solution : 1) Basis of induction -

Initially we assume,

$$n = 10 \quad \text{then}$$

$$2^{10} > (10)^3 \quad \text{is true}$$

2) Induction hypothesis -

Now assume, $n = K$ then

$$2^K > K^3$$

3) Inductive step -

Now we will assume $K = K + 1$ then the equation becomes

$$\begin{aligned} \text{L.H.S.} &= 2^{(K+1)} \\ &= 2 \cdot 2^K \\ &\geq \left(1 + \frac{1}{10}\right)^3 \cdot 2^K \\ &\geq \left(1 + \frac{1}{K}\right)^3 \cdot 2^K \end{aligned}$$

But as $2^K > K^3$ we will replace 2^K by K^3 then

$$\begin{aligned} &\geq \left(1 + \frac{1}{K}\right)^3 \cdot K^3 \\ &\geq \left(\frac{K+1}{K}\right)^3 \cdot K^3 \\ &\geq \frac{(K+1)^3}{K^3} \cdot K^3 \\ &\geq (K+1)^3 \\ &= \text{R.H.S.} \end{aligned}$$

Hence $2^{K+1} \geq (K+1)^3$ is also true.

This proves $2^n > n^3$ for $n \geq 10$.

→ Example 1.8.6 : Prove : $2^n > n$ using principle of mathematical induction.

Solution : 1) Basis of induction - Assume $n = 1$ then,

$$2^n > n \text{ becomes}$$

$$2^1 > 1$$

$$2 > 1$$

This also implies $2^n - n > 0$

2) Induction hypothesis : Assume $n = K$ is true

$$\text{i.e. } 2^K > K$$

$$\text{i.e. } 2^K - K > 0$$

Thus $2^K - K$ implies a positive number. i.e.

$$2^K - K = t \quad \dots (1)$$

3) Inductive step : Assume $K = K + 1$ then

$$2^{K+1} = (K+1)$$

$$2^K \cdot 2 = (K+1)$$

But from equation (1) $2^K = K + t$ then

$$(K+t) \cdot 2 = K + 1$$

i.e. $2K + 2t = K + 1$

$$K + 2t = 1$$

> 0 i.e. as positive number.

This implies $2^{K+1} > K + 1$. Hence $2^n > n$ true.

→ Example 1.8.7 : Using principle of mathematical induction, prove $5^n - 2^n$ is divisible by 3 for every $n \geq 0$.

Solution : Let,

1) Basis of induction :

Initially we assume,

$$n = 1 \text{ then}$$

$$5^n - 2^n = 5 - 2 = 3 \text{ which is divisible by 3.}$$

2) Inductive step : Now we will assume $n = n + 1$ then the expression becomes

$$\begin{aligned} 5^{n+1} - 2^{n+1} \\ &= 5^n \cdot 5 - 2^n \cdot 2 \\ &\quad \boxed{} \\ &= 5^n \cdot 5 - 5 \cdot 2^n + 3 \cdot 2^n \quad \text{taking 5 common} \\ &= 5 \cdot (5^n - 2^n) + 3 \cdot 2^n \end{aligned}$$

The $5^n - 2^n$ is divisible by 3 which is proved in basis of induction. The other term $3 \cdot 2^n$ is clearly divisible by 3. Hence $5 \cdot (5^n - 2^n) + 3 \cdot 2^n$ is divisible by 3. Hence $5^n - 2^n$ is divisible by 3 is proved.

→ Example 1.8.8 : By principle of induction, prove that $10^{2n} - 1$ is divisible by 11 for all $n \geq 1$.

Solution : Let,

1) Basis of induction : Initially we assume,

$$n = 1 \text{ then}$$

$$10^{2n} - 1$$

$$\begin{aligned}
 &= 10^{2(1)} - 1 = 100 - 1 \\
 &= 99 \quad \text{which is divisible by 11.}
 \end{aligned}$$

2) Inductive step : Now we will assume $n = n + 1$ then the expression becomes

$$\begin{aligned}
 &10^{2(n+1)} - 1 \\
 &= 10^{2n+2} - 1 \\
 &= 10^2 \cdot 10^{2n} - 1
 \end{aligned} \tag{1}$$

As $(10^{2n} - 1)$ is divisible by 11, the equation (1) is divisible by 11.

Example 1.8.9 : Using principle of mathematical induction, prove that for every $n \geq 1$,
 $7 + 13 + 19 + \dots + (6n + 1) = n(3n + 4)$ GTU : Summer-11,14. Marks 7

Solution : 1) Basis of induction : Assume $n = 1$, then

$$\begin{aligned}
 \text{L.H.S.} &= 6n + 1 \\
 &= 6 \cdot 1 + 1 \\
 &= 7
 \end{aligned}$$

$$\begin{aligned}
 \text{R.H.S.} &= n(3n + 4) \\
 &= 1 (3 \cdot 1 + 4) \\
 &= 7
 \end{aligned}$$

$$\therefore \text{L.H.S.} = \text{R.H.S.}$$

2) Induction Hypothesis : Now we will assume that the equation is true for $n = k$.
The equation then becomes -

$$7 + 13 + 19 + \dots + (6k + 1) = k(3k + 4) \tag{1}$$

3) Inductive step By assuming that the equation is true for $n = k$, we will check if equation is true for $n = k + 1$.

$$\begin{aligned}
 \text{L.H.S.} &= \underbrace{7 + 13 + 19 + \dots + (6k + 1)}_{\text{Assume true}} + 6(k + 1) + 1 \\
 &= k(3k + 4) + 4k + 6k + 6 + 1 \\
 &= 3k^2 + 10k + 7 \\
 &= 3k^2 + 3k + 4k + 3k + 3 + 4 \\
 &= (k + 1)(3k + 3 + 4) \\
 &= (k + 1)(3(k + 1) + 4) \\
 &= \text{R.H.S.}
 \end{aligned}$$

\therefore Taking common factor

$$\therefore \text{L.H.S.} = \text{R.H.S.}$$

Thus given equation is proved to be true.

→ Example 1.8.10 : Prove that for every integer $n \geq 0$ the number $4^{2n+1} + 3^{n+2}$ is multiple of 13.

Ans. : We will prove this using method of induction.

Basis : Assume $n = 1$

$$\begin{aligned} P(1) &= 4^{2(1)+1} + 3^{1+2} \\ &= 4^3 + 3^3 \\ &= 91 \text{ which is } = 13 \times 7 \end{aligned}$$

Hence $4^{2n+1} + 3^{n+2}$ is multiple of 13 when $n = 1$.

Inductive step : Assume $n = k$

$$\begin{aligned} P(k) &= 4^{2k+1} + 3^{k+2} \\ P(k) &= 13m \end{aligned} \quad \dots(1)$$

where m is some integer.

Thus we assume that given expression is true for $P(k)$. Now we will prove that is also true for $P(k+1)$.

Let, $n = k+1$ then

$$P(k+1) = 4^{2(k+1)+1} + 3^{(k+1)+2}$$

We can simplify it as -

$$\begin{aligned} &= 4^{(2k+1)+2} + 3^{(k+2)+1} \\ &= 4^2 \cdot 4^{2k+1} + 4^2 \cdot \underbrace{(3^{k+2} - 3^{k+2})}_{\downarrow} + 3^1 \cdot 3^{k+2} \\ &\text{equal to zero} \end{aligned}$$

Taking common factors as 4^2 and 3^{k+2} we get

$$= 4^2 (4^{2k+1} + 3^{k+2}) + 3^{k+2} (-4^2 + 3)$$

$$= 4^2 (13m) + 3^{k+2} (-13)$$

∴ equation (1)

$$P(k+1) = 13(4^2 m - 3^{k+2})$$

That is multiple of 13, for $P(k+1)$.

This proves that given integer is multiple of 13.

Example 1.8.11 : Prove that for any every $n >= 0$, $n(n^2 + 5)$ is divisible by 6.

GTU : Winter-14, Marks 7

Solution : 1) Basis of Induction

Initially we assume,

$$n = 1 \text{ then}$$

$$1(1^2 + 5) = 6 \text{ which is divisible by 6.}$$

2) Inductive step : Now, suppose it is true for $n = k$. Then for $n = k + 1$ we have

$$\begin{aligned} & (k+1)((k+1)^2 + 5) \\ &= (k+1)(k^2 + 2k + 1 + 5) \\ &= (k+1)(k^2 + 2k + 6) \\ &= k^3 + 3k^2 + 8k + 6 \\ &= (k^3 + 5k) + (3k^2 + 3k + 6) \\ &= (k^3 + 5k) + 3(k^2 + k + 2) \\ &= k(k^2 + 5k) + 3(k^2 + k + 2) \end{aligned}$$

The $k(k^2 + 5k)$ is already divisible by 6.

Now consider the term $3(k^2 + k + 2)$

$$= 3(k(k+1)+2)$$

$k(k+1)$ is a product of two consecutive number, as one of them must be even $k(k+1)$ is multiple of 2. Then $k(k+1)+2$ is also multiple of 2.

$\therefore 3(k(k+1)+2)$ is multiple of 6

$\therefore (k+1)((k+1)^2 + 5)$ is ultimately multiple of 6. This proves that $n(n^2 + 5)$ is divisible by 6.

Example 1.8.12 : Use the principle of mathematical induction to prove that $1 + 3 + 5 + \dots + r = n^2$ for all $n > 0$ where r is an odd integer and n is the number of terms in the sum. (Note : $r = 2n - 1$)

GTU : Winter-16, Marks 7

Solution : Let, L.H.S. is $1 + 3 + 5 + \dots + r$

$$\text{Let } r = 2n - 1$$

Hence L.H.S. becomes $1 + 3 + 5 + \dots + (2n - 1)$

Basis of Induction

Assume $n = 1$

then L.H.S. = 1

and R.H.S. = $1^2 = 1$

Hence L.H.S. = R.H.S. is proved

Inductive step : Assume $P(k)$ is true for some $k \in \mathbb{N}$

$$P(k) = 1 + 3 + 5 + \dots + (2k-1) = k^2$$

Now to prove $P(k+1)$ is true, we have $1 + 3 + 5 + \dots + (2k-1) + (2k+1)$

$$= k^2 + (2k+1) = k^2 + 2k + 1 = (k+1)^2$$

This proves that $P(k+1)$ is also true when $P(k)$ is true.

→ **Example 1.8.13 :** Write principle of mathematical induction. Prove that for every $n \geq 1$,

$$\sum_{i=1}^n \frac{1}{i(i+1)} = n/(n+1).$$

GTU : Summer-18, Marks 7

Solution : Basis of induction

Let us assume $n = 1$.

$$\begin{aligned} \text{L.H.S.} &= \sum_{i=1}^1 \frac{1}{i(i+1)} = \frac{1}{1(1+1)} \\ &= \frac{1}{2} \end{aligned}$$

$$\begin{aligned} \text{R.H.S.} &= \frac{n}{(n+1)} \\ &= \frac{1}{(1+1)} = \frac{1}{2} \end{aligned}$$

Thus, L.H.S. = R.H.S.

Inductive step : Assume that the induction holds true for $n = k$.

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

Now assume $n = k+1$, then

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

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

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

Thus the equation holds true for $n = k + 1$.

From above cases,

$$\sum_{i=1}^n \frac{1}{i(i+1)} = \frac{n}{(n+1)} \text{ is true}$$

1.9 Recursive Definition

- To construct recursively defined function we need following things
- 1. Initial condition or basis : It basically denotes initial value of the function.
- 2. Recursion : Using a fixed procedure compute the value of the function at $n + 1$ using one or more values of that function.
- To construct recursively defined set we need following things -
- 1. Initial condition or basis : Give one or more element of the set
- 2. Recursion : Give the rule for generating elements of the set in-terms of previously defined elements.

→ Example 1.9.1 : Let, $f(n) = 2^n$ where n is natural number. Give recursive definition of the function.

Solution : 1. Initial condition : $f(0) = 1$

2. Recursion : $f(n + 1) = 2 \cdot f(n)$ for $n > 0$

The (2) step i.e. recursion will work as follows -

$$2^1 = 2^{0+1} = 2^0 \cdot 2 = 2$$

$$2^2 = 2^{1+1} = 2^1 \cdot 2 = 4$$

$$2^3 = 2^2 \cdot 2 = 4 \cdot 2 = 8$$

and so on.

→ Example 1.9.2 : Give recursive definition of $f(n) = n!$

Solution : (1) Initial condition : $f(0) = 1$

(2) Recursion : $f(n + 1) = (n + 1) \cdot f(n)$

It can be derived as

$$1! = f(1) = (0 + 1) \cdot f(0) = 1$$

$$2! = f(2) = (1 + 1) \cdot f(1) = 2$$

$$3! = f(3) = (2 + 1) \cdot f(2) = 3 \cdot 2 = 6$$

⇒ Example 1.9.3 : Give recursive definition for the set S of {0, 1} with no more than single 1.

Solution : (1) Initial condition : $\lambda, 1 \in S$

(2) Recursive : If $x \in S$ then $0x$ and $x0 \in S$

⇒ Example 1.9.4 : Give Recursive definition of each of the following sets

a. The set T is positive integer divisible by 2 or 7.

b. The set U of all string in {0, 1} containing the substring 00.

GTU : Winter-14, Marks 7

Solution : a) $2 \in T$;

$7 \in T$;

For every $n \in T$ and every positive integer i , $in \in T$.

b) $00 \in U$;

For every $x \in U$ all strings $0x, x0, 1x, x1$ are in U .

⇒ Example 1.9.5 : Give recursive definition of the following

i) The set S of all integers (positive or negative) divisible by 10.

Solution : i) $0 \in S$;

For every x both $x + 10$ and $x - 10$ are in S

⇒ Example 1.9.6 : Give recursive definitions of the following sets.

a. The set T of all strings of the form $a^i b^j$ where $j \leq i \leq 2j$

b. The set Q of all strings of the form $a^i b^j$ where $i \geq 2j$

Solution : a) (1) Initial condition : λ or $\in T$;

(2) Recursive step : For every $x \in T$ both axb and $aaxb$ are in T .

b) (1) Initial condition : λ or $\in Q$;

(2) Recursive step : For every $x \in Q$ both ax and $aaxb$ are in Q .

⇒ Example 1.9.7 : Give recursive definition for the strings of the form $0^n 1^n$. Where $n \in N$ the set of natural numbers.

Solution : (1) Initial condition : $\lambda \in T$

(2) Recursive step : If $x \in T$ then $Cx1 \in T$.

⇒ Example 1.9.8 : Recursively define the set S equal to set of odd integers.

Solution : (1) Initial condition : $1 \in S$

(2) Recursive step : If $x \in S$ then $(2x + 1) \in S$.

⇒ Example 1.9.9 : Give the recursive definition of PAL of palindrome over any alphabet Σ .
GTU : Summer-11, Marks 7

Solution : Consider $\Sigma = \{a, b\}$ then the recursive definition of PAL of palindrome over Σ is

Base : $a, b \in \text{PAL}$

Recursion : $aua, bub, \epsilon \in \text{PAL}$ if $u \in \text{PAL}$.

The kleen closure is also called star closure. This is set of strings of any length (including null string ϵ). Each string is obtained from input set Σ . The kleen star of the empty language ϕ is the empty string ϵ .

⇒ Example 1.9.10 : Let $\Sigma = \{a, b\}$ obtain $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \dots$

Solution : $\Sigma^* = \{\epsilon, a, b, aa, bb, ab, ba, aba, aaa, bbb, baba, abaab, \dots\}$ is a set of strings of any length.

The positive closure Σ^+ can be defined as $\Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \dots$ that means it consists of all the strings of any length except a null string.

⇒ Example 1.9.11 : Let $\Sigma = \{a, b\}$ obtain $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \dots$

Solution : $\Sigma^+ = \{a, b, aa, bb, ab, ba, aba, aaa, bbb, baba, abaab, \dots\}$ is a set of strings of any length except null string (epsilon).

Note that $\Sigma^* = \Sigma^+ + \epsilon$.

1.10 Short Questions and Answers

Q.1 Define the term set ?

Ans. : The set is a collection of objects. These objects are called the elements of the set. The set elements are enclosed within the curly bracket. Typically the set is denoted by capital letter.

Q.2 What is recursion method of set representation ?

Ans. : The recursion occurs to define the elements of the set.

For example

$A = \{x \mid x \text{ is square of } n\}$ where $n \leq 10$. This defines the set as

$$A = \{0, 1, 4, 9, \dots, 100\}$$

Q.3 Give the definition of - function

Ans. : If D is a set then we can define the function as

$$F(D) = \{f(x) \mid x \in D\}$$

$D = \{1, 2, 3, \dots\}$ is a domain.

Q.4 What is composition of function ?

Ans. : The composition of function is a function in which result of one function is given as input to another function. It is denoted as $g \circ f$ or $f \circ g$.

Q.5 Enlist various connectives used in logical statements ?

Ans. : Various connectives used in logical statements are i) Negation ii) Conjunction iii) Disjunction iv) Implication v) if and only if.

Q.6 What is relation ?

Ans. : The relation R is a collection for the set S which represents the pair of elements. For example - (a,b) is in R . We can represent their relation as aRb .

Q.7 What is equivalence relation ?

Ans. : A relation is said to be equivalence if it is reflexive, symmetric and transitive.

Q.8 Explain the palindrome property of string

Ans. : Palindrome is a property of string in which the string can be read same from left to right as well as from right to left.



Regular Languages

2.1 Regular Expressions

Let Σ be an alphabet which is used to denote the input set. The regular expression over Σ can be defined as follows.

Definition :

1. ϕ is a regular expression which denotes the empty set.
2. ϵ is a regular expression and denotes the set $\{\epsilon\}$ and it is a null string.
3. For each ' a ' in Σ ' a ' is a regular expression and denotes the set $\{a\}$.
4. If r and s are regular expressions denoting the languages L_1 and L_2 respectively, then

$r+s$ is equivalent to $L_1 \cup L_2$ i.e. union.

rs is equivalent to $L_1 L_2$ i.e. concatenation.

r^* is equivalent to L_1^* i.e. closure.

The r^* is known as kleen closure or closure which indicates occurrence of r for ∞ number of times.

For example if $\Sigma = \{a\}$ and we have regular expression $R = a^*$, then R is a set denoted by $R = \{\epsilon, a, aa, aaa, aaaa, \dots\}$

That is R includes any number of a 's as well as empty string which indicates zero number of a 's appearing, denoted by ϵ character.

Similarly there is a positive closure of L which can be shown as L^+ . The L^+ denotes set of all the strings except the ϵ or null string. The null string can be denoted by ϵ or \wedge

If $\Sigma = \{a\}$ and if we have regular expression $R = a^+$ then R is a set denoted by

$$R = \{a, aa, aaa, aaaa, \dots\}$$

We can construct L^* as

$$L^* = \epsilon \cdot L^+$$

Let us try to use regular expressions with the help of some examples.

⇒ **Example 2.1.1 :** Write the regular expression for the language accepting all combinations of a 's over the set $\Sigma = \{a\}$.

Solution : All combinations of a 's means a may be single, double, triple and so on. There may be the case that a is appearing for zero times, which means a null string. That is we expect the set of $\{\epsilon, a, aa, aaa, \dots\}$. So we can give regular expression for this as

$$R = a^*$$

That is kleen closure of a .

⇒ **Example 2.1.2 :** Design the regular expression (r.e.) for the language accepting all combinations of a 's except the null string over $\Sigma = \{a\}$

Solution : The regular expression has to be built for the language

$$L = \{a, aa, aaa, \dots\}$$

This set indicates that there is no null string. So we can denote r.e. as

$$R = a^+$$

As we know, positive closure indicates the set of strings without a null string.

⇒ **Example 2.1.3 :** Design regular expression for the language containing all the strings containing any number of a 's and b 's.

Solution : The regular expression will be

$$\text{r.e.} = (a + b)^*$$

This will give the set as $L = \{\epsilon, a, aa, ab, b, ba, bab, abab, \dots\}$ any combination of a and b .

The $(a + b)^*$ means any combination with a and b even a null string.

⇒ **Example 2.1.4 :** Construct the regular expression for the language containing all strings having any number of a 's and b 's, except the null string.

Solution : r.e. = $(a+b)^*$

This regular expression will give the set of strings of any combination of a 's and b 's except a null string.

⇒ **Example 2.1.5 :** Construct the r.e. for the language accepting all the strings which are ending with 00 over the set $\Sigma = \{0, 1\}$.

Solution : The r.e. has to be formed in which at the end, there should be 00. That means

$$\text{r.e.} = (\text{any combination of } 0\text{'s and } 1\text{'s}) 00$$

$$\text{i.e.} \quad \text{r.e.} = (0+1)^* 00$$

Thus the valid strings are 100, 0100, 1000, 10100 ... strings ending with 00.

- Example 2.1.6 : Write r.e. for the language accepting the strings which are starting with 1 and ending with 0, over the set $\Sigma = \{0, 1\}$.

Solution : The first symbol in r.e. should be 1 and the last symbol should be 0.

$$\text{So, } R = 1(0+1)^*0$$

Note that the condition is strictly followed by keeping starting and ending symbols correctly. In between them there can be any combination of 0 and 1 including a null string.

- Example 2.1.7 : If $L = \{\text{The language starting and ending with } a \text{ and having any combination of } b's \text{ in between, then what is } r ?\}$

Solution : The regular expression

$$r = a b^* a$$

- Example 2.1.8 : Describe in simple English the language represented by the following regular expression

$$r = (a + ab)^*$$

Solution : We will first try to find out the set of strings, which can be possible by this r ,

$$L(r) = \{a, ab, abab, abab, aaa, \dots\}$$

The language is beginning with a but not having consecutive (in a clump) b 's.

- Example 2.1.9 : Write regular expression to denote the language L over Σ^* , where $\Sigma = \{a, b, c\}$ in which every string will be such that any number of a 's is followed by any number of b 's is followed by any number of c 's.

Solution : As we have seen any number of a 's means a^* any number of b 's means b^* any number of c 's means c^* . Since as given in problem statement, b 's appear after a 's and c 's appear after b 's. So the regular expression could be -

$$r = a^* b^* c$$

- Example 2.1.10 : Write a regular expression to denote a language L over Σ^* , where $\Sigma = \{a, b, c\}$ such that every string will have atleast one a followed by atleast one b followed by atleast one c .

Solution : Now, in this problem the condition is slightly changed to "atleast". That means the null string is not allowed at all. So, we can write

$$R = a^* b^* c^*$$

→ Example 2.1.11 : Write r.e. to denote a language L which accepts all the strings which begin or end with either 00 or 11.

Solution : The r.e. can be categorized into two subparts.

$$R = L_1 + L_2$$

L_1 = The strings which begin with 00 or 11.

L_2 = The strings which end with 00 or 11.

Let us find out L_1 and L_2 .

$$L_1 = (00 + 11) \text{ (any number of 0's and 1's)}$$

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

Similarly,

$$L_2 = \text{(any number of 0's and 1's)} (00 + 11)$$

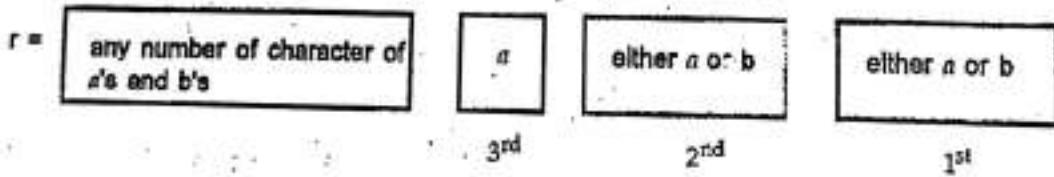
$$= (0 + 1)^* (00 + 11)$$

Hence

$$R = [(00+11)(0+1)^*] + [(0+1)^*(00+11)]$$

→ Example 2.1.12 : Write a r.e. to denote a language L over Σ^* , where $\Sigma = \{a, b\}$ such that the 3rd character from right end of the string is always a.

Solution : The r.e. contains the third symbol from right end as a



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

Thus the valid strings are babb or baba or babba or abab and so on.

→ Example 2.1.13 : Construct r.e. for the language L which accepts all the strings with atleast two b's over the $\Sigma = \{a, b\}$.

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

Atleast two b's are maintained but the two b's are surrounded by any combination of a's and b's.

→ Example 2.1.14 : Construct r.e. for the language which consists of exactly two b's over the set $\Sigma = \{a, b\}$

Solution : Now, this problem is similar to the previous problem. But only difference is that there should be exactly two b's.

$$R = a^* b a^* b a^*$$

a^* indicates either the string contains any number of a's or a null string. Thus we can derive any string having exactly two b's and any number of a's.

→ Example 2.1.15 : Write r.e. which contains L having strings which should have atleast one 0 and atleast one 1.

Solution : The required expression will be

$$R = [(0+1)^* 0 (0+1)^* 1 (0+1)^*] + [(0+1)^* 1 (0+1)^* 0 (0+1)^*]$$

→ Example 2.1.16 : Construct r.e. which denotes a language L over the set $\Sigma = \{0\}$ having even length of string.

Solution : Since $\Sigma = \{0\}$ there are strings in L of even length i.e.

$$L = [\epsilon, 00, 0000, 000000, \dots]$$

So we can give the regular expression as

$$R = (00)^*$$

Thus kleen closure indicates the recursion of two zeros which will ultimately give the even length of the string.

→ Example 2.1.17 : Write r.e. which denotes a language L over the set $\Sigma = \{1\}$ having odd length of strings.

Solution : $R = 1 (11)^*$

The odd length strings are $L = \{1, 111, 11111, \dots\}$. Note the difference between previous example and this one.

→ Example 2.1.18 : Describe the language denoted by following regular expression

$$\text{r.e.} = (b^* (aaa)^* b^*)^*$$

Solution : The language can be predicted from the r.e. by finding out the meaning of it. We can split the r.e. as

$$\text{r.e.} = (\text{any combination of } b\text{'s}) (aa)^* (\text{any combination of } b\text{'s}).$$

$L = \{\text{The language consists of the strings in which } a\text{'s appear tripled, there is no restriction on number of } b\text{'s}\}$

→ Example 2.1.19 : Construct regular expression for the language L over the set $\Sigma = \{a, b\}$ in which the total number of a 's are divisible by 3.

Solution : The total number of a 's are divisible by 3. So the valid strings can be

$$L = \{baab, bababa, ababab, \dots\}$$

The regular expression can be

$$\text{r.e.} = (b^* a b^* a b^* a b^*)^*$$

→ Example 2.1.20 : Write the r.e. to denote the language L over $\Sigma = \{a, b\}$ such that all the strings do not contain the substring "ab".

Solution : The $L = \{\epsilon, a, b, bb, aa, ba, \dots\}$

$$\text{The r.e.} = (b^* a^*)$$

In this regular expression if we substitute $a^* = \epsilon$ we get all combination of b 's and similarly if we substitute $b^* = \epsilon$ then we will get all combinations of a 's. We have strictly maintained a condition for not to have ab as substring by giving the regular expression as $b^* a^*$.

→ Example 2.1.21 : Write regular expressions for the following languages of all strings in $\{0, 1\}^*$

- i) Strings that do not end with 01.
- ii) Strings with odd number of 1's.

GTU : Summer-11, Marks 5

Solution : i) The string that do not end with 01 means, it ends with 0, 1, 00, 11 or 10. Hence

$$\text{r.e.} = \epsilon + 0 + 1 + [(0 + 1)^* (00 + 11 + 10)]$$

$$\text{ii) r.e. } (0^* 1 0^* 1 0^*)^* 10^*$$

→ Example 2.1.22 : Write regular expressions for the following languages of all strings in $\{0, 1\}^*$

- i) Strings that contains odd number of 0's (zeroes)
- ii) Strings that begin or end with 00 or 11.

GTU : Winter-12, Marks 5

$$\text{Solution : i) r.e.} = 1^* (0 1^* 0 1)^* 01^*$$

$$\text{ii) r.e.} = (0 + 1)^* (00 + 11)$$

→ Example 2.1.23 : Write regular expressions for the following languages of all strings in $\{0, 1\}^*$

- i) Strings that start with 1 and do not end with 10.
- ii) Strings with length 6 or less.

GTU : Winter-13, Marks 5

Solution : i) r.e. = $1 (0 + 1)^* (00 + 11 + 01 + 1)$
ii) r.e. = $[\epsilon + 1 + 0 + (0 + 1)(0 + 1) + (0 + 1)(0 + 1)(0 + 1) + (0 + 1)(0 + 1)(0 + 1)(0 + 1) + (0 + 1)(0 + 1)(0 + 1)(0 + 1)(0 + 1) + (0 + 1)(0 + 1)(0 + 1)(0 + 1)(0 + 1)]$

→ **Example 2.1.24 :** Find a regular expression corresponding to each of the following subsets of $\{0, 1\}^*$

- The language of all strings that do not contain the substring 110
- The language of all strings that contain both 101 and 010 as substring
- The language of all strings in which both the number of 0's and number of 1's are odd.

GTU : Winter-14, Marks 7

Solution : i) r.e. = $0^* (100^*)^* 1^*$
ii) r.e. = $(0 + 1)^* [(101 (0 + 1)^* 010) + (010 (0 + 1)^* 101) + (1010) + (0101)] (0 + 1)^*$
iii) r.e. = $0 (00 + 010 + 100 + 101)^* + 1 (11 + 101 + 011 + 110)^* + 01 + 10$

→ **Example 2.1.25 :** Write Regular Expressions corresponding to each of the following subsets of $\{0, 1\}^*$

- The language of all strings in $\{0, 1\}^*$ that containing at least two 0's.
- The language of all strings containing both 101 and 010 as substrings.
- The language of all strings that do not end with 01.

GTU : Summer-16, Marks 7

Solution : i) $(0 + 1)^* 0(0 + 1)^* 0(0 + 1)^*$

ii) r.e. = $(0+1)^*[(101 (0+1)^* 010) + (010 (0+1)^* 101) + (1010) + (0101)] (0 + 1)^*$

iii) The strings that do not end with 01 means it end with 0, 1, 00, 11 or 10.

Hence r.e. = $\epsilon + 0 + 1 + [(0+1)^* (00+11+10)]$

→ **Example 2.1.26 :** Write a regular expression for language L over $\{0, 1\}$ such that every string in L

- Begins with 00 and ends with 11.
- Contains alternate 0 and 1.

GTU : Winter-16, Marks 7

Solution : i) r.e. = $00(0+1)^* 11$

ii) r.e. = $(01)^*$

→ **Example 2.1.27 :** Find a regular expression corresponding to each of the following subsets of $\{0, 1\}^*$.

- The language of all strings that begin or end with 00 or 11.
- The language of all strings containing both 11 and 010 as substrings.

GTU : Summer-18, Marks 3

Solution : i) r.e. = $[(00+11)(0+1)^*] + [(0+1)^* (00+11)]$

ii) $[(0+1)^* 11(0+1)^* 010(0+1)^*] + [(0+1)^* 010(0+1)^* 11(0+1)^*]$

→ Example 2.1.28 : Find a regular expression corresponding to each of the following subsets of $\{0, 1\}^*$.

i) The language of all strings that begin or end with 00 or 11.

(Refer example 2.22 (ii))

ii) The language of all strings beginning with 1 and ending with 0.

GTU : Winter-18, Marks 4

Solution : i) r.e. = $(00 + 11)(0 + 1)^*(00 + 11)$

ii) $1(0 + 1)^* 0$.

→ Example 2.1.29 : Write regular expression over the alphabets {a, b} consisting strings :

- Second last character as 'a'

- Starting with 'a' and ending with 'b'.

GTU : Summer-19, Marks 4

Solution : i) r.e. = $(a+b)^* a(a+b)$ ii) r.e. = $a(a+b)^* b$

→ Example 2.1.30 : Find regular expression and also derive the words corresponding to the language defined recursively below over $\Sigma = \{a, b\}$.

i) $a \in L$ ii) For any $x \in L$ xa and xb are elements of L

GTU : Winter-19, Marks 3

Solution : i) Regular expression = a. That means the regular expression contains the word containing single a.

ii) Regular expression = $(a+b)^* a + (a+b)^* b = (a+b)^+$

That means the regular expression contains the word that may either ends with single a or single b.

2.2 Regular Languages

The regular languages are the languages that are represented by regular expressions. The regular languages are the languages that can be modeled using finite automata.

2.3 Introduction to Finite Automata

The finite state system represents a mathematical model of a system with certain input. The model finally gives certain output. The input when given to the machine it is processed by various states, these states are called as intermediate states.

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

where,

Q is a finite set of states, which is non empty.

Σ is input alphabet, indicates input set.

q_0 is an initial state and q_0 is in Q i.e. $q_0 \in Q$.

F is a set of final states.

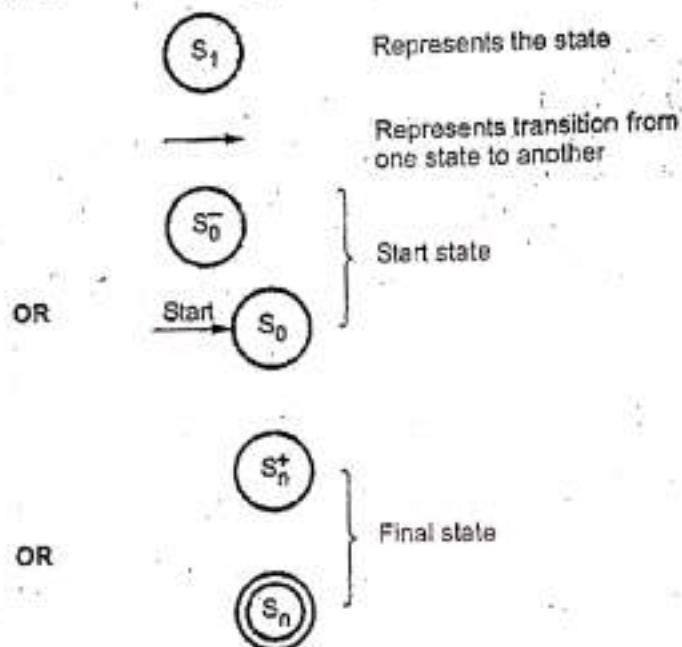
δ is a transition function or a mapping function. Using this function the next state can be determined.

2.3.1 Representation

The strings and languages can be accepted by a finite automata, when it reaches to a final state. There are two preferred notations for describing automata :

1. Transition diagram

The notations used in transition diagram are -



For example :

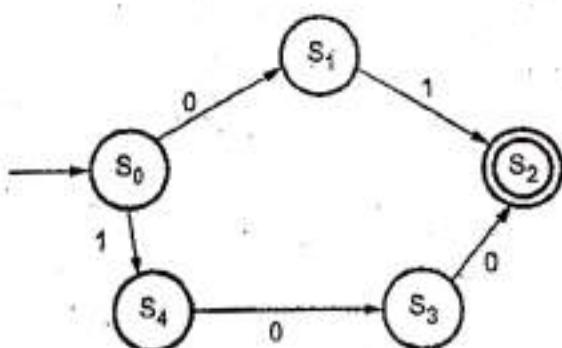


Fig. 2.3.1 Transition diagram

2. Transition table

This is a tabular representation of finite automata. For transition table the transition function is used.

For example :

Input States \	a	b
States /		
q_0	q_1	-
q_1	-	q_2
q_2	q_2	-

The rows of the table corresponds to states and columns of the table corresponds to inputs.

2.3.2 Control Over Input String

The finite automata can be represented as :

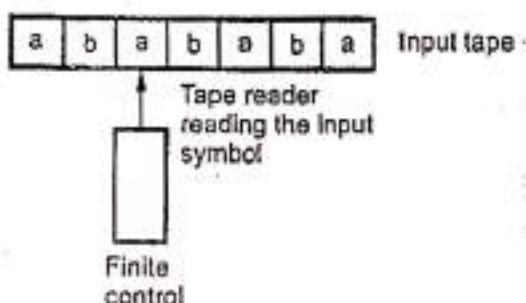


Fig. 2.3.2 Model for finite automata

The finite automata can be represented using :

- i) **Input tape** - It is a linear tape having some number of cells. Each input symbol is placed in each cell.
- ii) **Finite control** - The finite control decides the next state on receiving particular input from input tape. The tape reader reads the cells one by one from left to right and at a time only one input symbol is read.

For example, suppose current state is q , and suppose reader is reading the symbol 'a' then it is finite control which decides what will be the next state at input 'a'. The transition from current state q with input w to next state q' producing w' will be represented as,

$$(q, w) \vdash (q', w')$$

If w is a string and M is a finite automata, then w is accepted by the FA.

iff $(w, s) \models (q, \epsilon)$

with q as final state.

The set of strings accepted by a FA given by M then M is accepted by language L . The acceptance of M by some language L is denoted by $L(M)$.

A machine M accepts a language L iff,

$$L = L(M).$$

2.4 Memory Requirement in a Recognizer

For recognizing any language there are two conventions that are used. First - restrict to read the input in single pass from left to right. Seconds make tentative decisions after each input symbol.

The advantage of using these conventions is that we can keep track of how much information is read and making decisions at appropriate places lead us to accept any valid string belonging to that language.

For example - Consider a language L containing all the strings that end with 11.

In this language initially we may read any number of 0's or/and 1's. But before reaching to final state we must read two consecutive 1's. We can represent the language L as

$$L = \{0,1\}^* \{11\}$$

The finite automata can be designed as

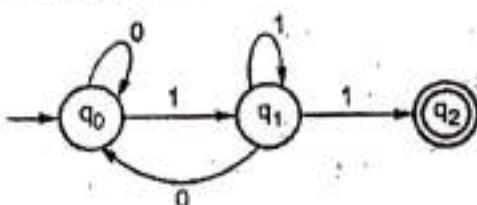


Fig. 2.4.1

Here by reading first '1' of 11 we change the state. From q_0 to q_1 and on reading second '1' of 11 we change the state to q_2 and make q_2 as final state.

2.5 Types of Finite Automata

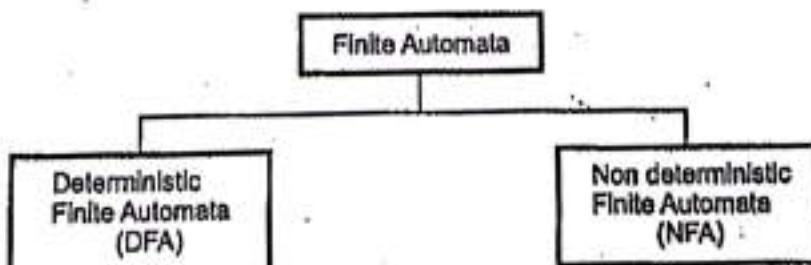


Fig. 2.5.1 DFA and NFA

There are two types of finite automata -

- i) Deterministic Finite Automata. (DFA)
- ii) Non deterministic Finite Automata. (NFA)

The deterministic finite automata is deterministic in nature, in the sense, each move in this automata is uniquely determined on current input and current state. On the other hand, it is non deterministic in nature, in NFA i.e. non deterministic finite automata.

2.6 Deterministic Finite Automata

GTU : Winter-11, 12, 13, Summer-11, 15, 16, Marks 7

A deterministic finite automation is a collection of following things -

- 1) The finite set of states which can be denoted by Q .
- 2) The finite set of input symbols Σ .
- 3) The start state q_0 such that $q_0 \in Q$.
- 4) A set of final states F such that $F \subseteq Q$.

5) The mapping function or transition function denoted by δ . Two parameters are passed to this transition function : One is current state and other is input symbol. The transition function returns a state which can be called as next state.

For example, $q_1 = \delta(q_0, a)$ means from current state q_0 , with input a the next state transition is q_1 .

In short, the DFA is a five tuple notation denoted as :

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

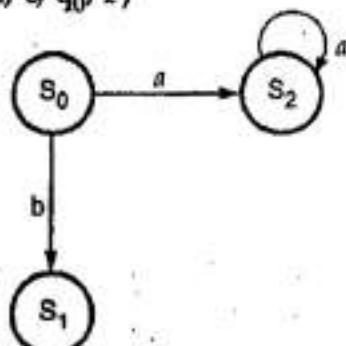


Fig. 2.6.1 Deterministic finite automata

The name of DFA is A which is a collection of above described five elements.

→ Example 2.6.1 : Design a FA which checks whether the given binary number is even.

Solution : The binary number is made up of 0's and 1's when any binary number ends with 0 it is always even and when a binary number ends with 1 it is always odd. For example,

0100 is an even number, it is equal to 4.

0011 is an odd number, it is equal to 3.

And so while designing FA we will assume one start state, one state ending in 0 and other state for

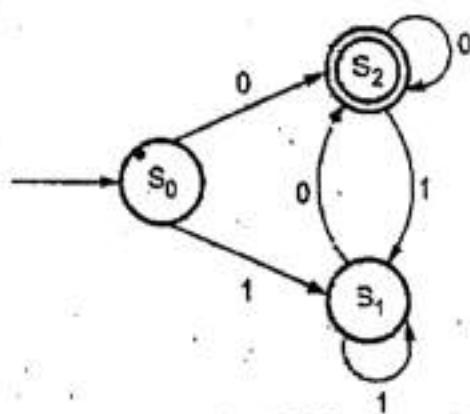


Fig. 2.6.2

ending with 1. Since we want to check whether given binary number is even or not, we will make the state for 0, as the final state.

The FA indicates clearly S_1 is a state which handles all the 1's and S_2 is a state which handles all the 0's. Let us take some input.

$$01000 \Rightarrow CS_2 1000$$

$$C1S_1 000$$

$$C10S_2 00$$

$$C100S_2 0$$

$$C1000S_2$$

Now at the end of input we are in final or in accept state so it is an even number. Similarly, let us take another input

$$1011 \Rightarrow S_0 |- 1011$$

$$1S_1 |- 011$$

$$10S_2 |- 11$$

$$101S_1 |- 1$$

$$1011S_1 |-$$

Now we are at the state S_1 , which is a non final state. Here the symbol " |- " denotes the symbols to be scanned. On L.H.S. of " |- " the already read symbols are given and at the R.H.S. of " |- " the symbols to be scanned are given.

Another idea to represent FA with the help of transition table.

States \ Input	0	1
→ S_0	S_2	S_1
S_1	S_2	S_1
(S_2)	S_2	S_1

Transition table

Thus the table indicates in the first column all the current states. And under the column 0 and 1 the next states are shown.

The first row of transition table can be read as : when current state is S_0 , on input 0 the next state will be S_2 and on input 1 the next state will be S_1 . The arrow marked to S_0 indicates that it is a start state and circle marked to S_2 indicates that it is a final state.

→ Example 2.6.2 : Design FA which accepts odd number of 1's and any number of 0's.

Solution : In the problem statement, it is indicated that there will be a state which is meant for odd number of 1's and that will be the final state. There is no condition on number of 0's.

At the start if we read input 1 then we will go to state S_1 which is a final state as we have read odd number of 1's. There can be any number of zeros at any state and therefore the self loop is applied to state S_2 as well as to state S_1 . For example, if the input is 10101101, in this string there are any number of zeros but odd number of ones.

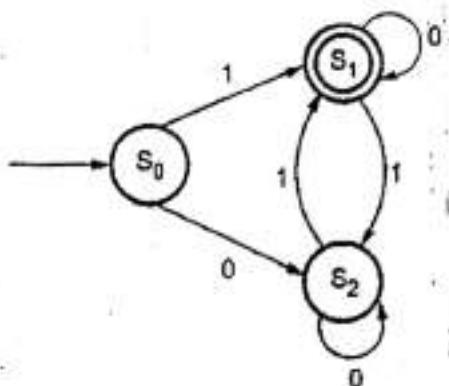


Fig. 2.6.3

→ Example 2.6.3 : Design FA which checks whether the given unary number is divisible by 3.

Solution :

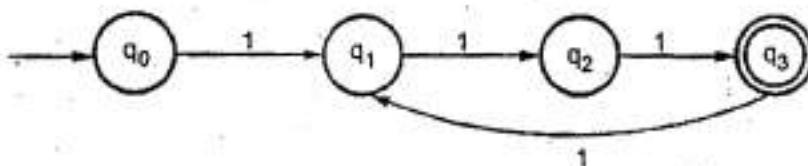


Fig. 2.6.4

The unary number is made up of ones. The number 3 can be written in unary form as 111, number 5 can be written as 11111 and so on. The unary number which is divisible by 3 can be 111 or 111111 or 111111111 and so on. Refer the given transition table.

Consider a number 111111 which is equal to 6 i.e. divisible by 3. So after complete scan of this number we reach to final state q_3 .

Start 111111

State q_0

1 q_1 11111

11 q_2 1111

111 q_3 111

1111 q_1 11

11111 q_2 1

111111 q_3 → Now we are in final state.

Input \ State	1
→ q_0	q_1
q_1	q_2
q_2	q_3
q_3	q_1

Example 2.6.4 : Design FA to check whether given decimal number is divisible by three.

Solution : To determine whether the given decimal number is divisible by three, we need to take the input number digit by digit. Also, while considering its divisibility by three, we have to consider that the possible remainders could be 1, 2 or 0. The remainder 0 means, it is divisible by 3. Hence from input set {0, 1, 2, ..., 9} (since decimal number is a input), we will get either remainder 0 or 1 or 2 while testing its divisibility by 3. So we need to group these digits according to their remainders. The groups are as given below -

remainder 0 group : $S_0 : (0, 3, 6, 9)$

remainder 1 group : $S_1 : (1, 4, 7)$

remainder 2 group : $S_2 : (2, 5, 8)$

We have named out these states as S_0, S_1 and S_2 . The state S_0 will be the final state as it is remainder 0 state.

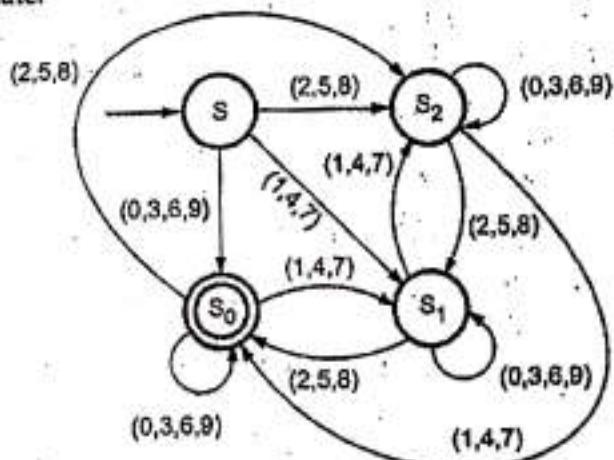


Fig. 2.6.5

Let us test the above FA, if the number is 36 then it will proceed by reading the number digit by digit.

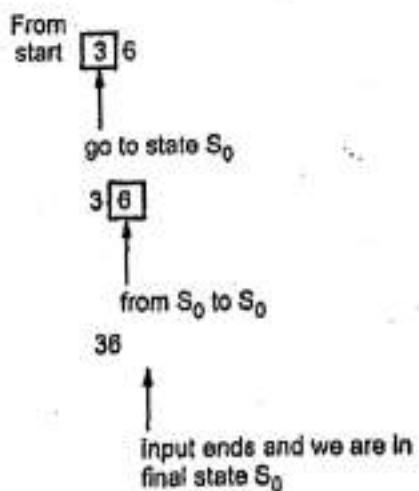


Fig. 2.6.6

Hence the number is divisible 3, isn't it? Similarly if number is 121 which is not divisible by 3, it will be processed as

S 121

1 S₁ 21

12 S₀ 1

121 S₁ which is remainder 1 state.

Example 2.6.5 : Draw an DFA that recognize the language of all strings of 0's and 1's of length at least 1 that, if they were interpreted as binary representation of integers, would represent evenly divisible by 3. Your DFA should accept the string 0 but no other strings with leading 0's.

GTU : Winter-11, Marks 5

Solution : The input number is a binary number. Hence the input set will be $\Sigma = \{0, 1\}$. The start state is denoted by S, the remainder 0 is by S₀, remainder 1 by S₁, and remainder 2 by S₂.

Let us take some number 010 which is equivalent to 2. We will scan this number from MSB to LSB.

0	1	0
↑	↑	↑
S ₀	S ₁	S ₂

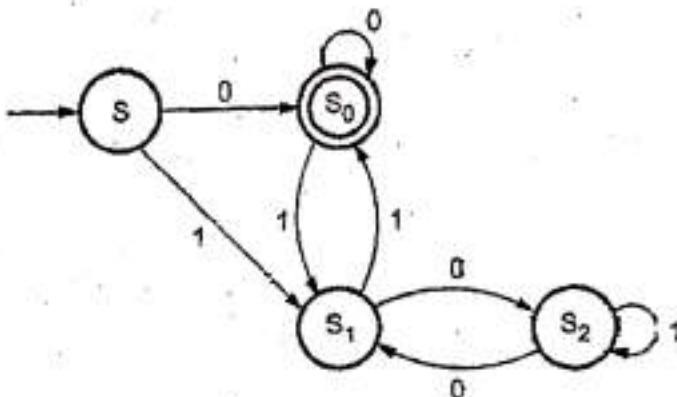


Fig. 2.6.7

Then we will reach to state S₂ which is remainder 2 state. Similarly, for input 1001 which is equivalent to 9 we will be in final state after scanning the complete input.

1	0	0	1
↑	↑	↑	↑
S ₁	S ₂	S ₁	S₀

Thus the number is really divisible by 3.

Example 2.6.6 : Design FA which accepts even number of 0's and even number of 1's.

Solution : This FA will consider four different stages for input 0 and 1. The stages could be :

- even number of 0 and even number of 1,
- even number of 0 and odd number of 1,
- odd number of 0 and even number of 1,
- odd number of 0 and odd number of 1.

Let us try to design the machine :

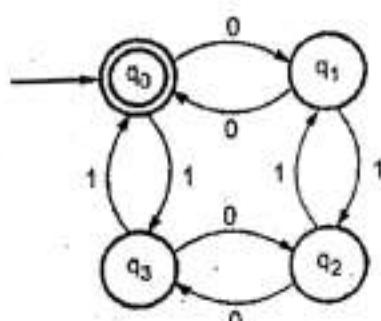


Fig. 2.6.8

Here q_0 is a start state as well as final state. Note carefully that a symmetry of 0's and 1's is maintained. We can associate meanings to each state as :

q_0 : State of even number of 0's and even number of 1's.

q_1 : State of odd number of 0's and even number of 1's.

q_2 : State of odd number of 0's and odd number of 1's.

q_3 : State of even number of 0's and odd number of 1's.

→ Example 2.6.7 : Design FA to accept the string that always ends with 00.

Solution :

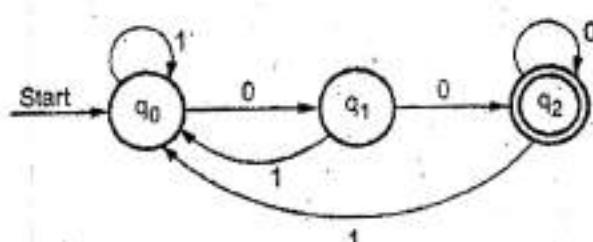


Fig. 2.6.9

If the input is 01001100 it will be processed as :

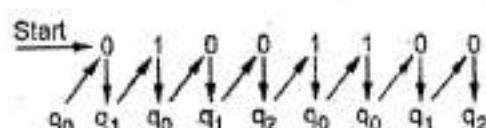


Fig. 2.6.10

The q_2 is a final state, hence the input is accepted.

→ Example 2.6.8 : Design FA to accept L , where $L = \{ \text{Strings in which } a \text{ always appears tripled} \}$ over the set $\Sigma = \{a, b\}$.

Solution : For this particular language the valid strings are $aabb$, $baaaaab$, $bbaaab$ and so on. The a always appears in a clump of 3. The TG (transition graph) will look like this -

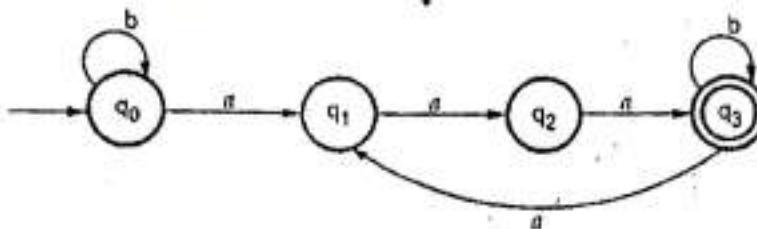


Fig. 2.6.11

Note that the sequence of triplets a is maintained to reach to the final state.

→ **Example 2.6.9 :** Design FA to accept L where all the strings in L are such that total number of a 's in them are divisible by 3.

Solution : As we have seen earlier, while testing divisibility by 3, we group the input as remainder 0, remainder 1 and remainder 2.

Hence,

S_0 : State of remainder 0.

S_1 : State of remainder 1.

S_2 : State of remainder 2.

Note the difference between previous example and this, here there is no condition as a should be in clump but total number of a 's in a string are divisible by 3. Hence b 's are allowed in between.

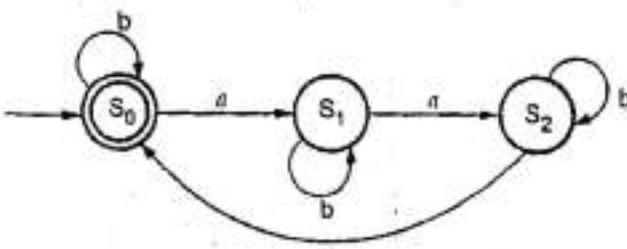


Fig. 2.6.12

→ **Example 2.6.10 :** Design DFA to accept odd and even numbers represented using binary notation.

Solution : The binary number that ends with 0 is an even number and binary number that ends with 1 is an odd number. Hence the DFA is shown in Fig. 2.6.13.

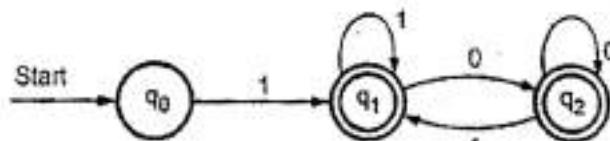


Fig. 2.6.13

→ **Example 2.6.11 :** Write a DFA to accept the language $L = \{L : |W| \bmod 5 \neq 0\}$.

Solution : The string which we obtain should not be divisible by 5. Hence the, DFA will be,

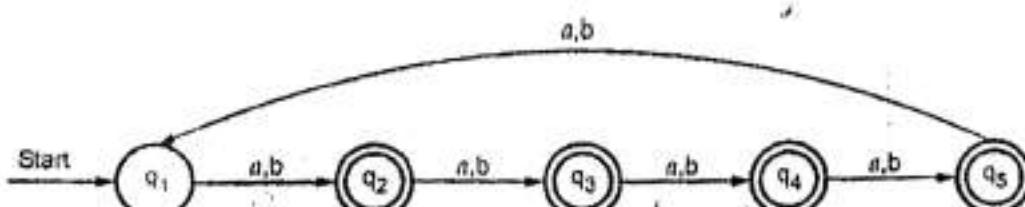


Fig. 2.6.14

The transition table can be drawn as follows -

State	Input	
	a	b
q ₁	q ₂	q ₂
q ₂	q ₃	q ₃
q ₃	q ₄	q ₄
q ₄	q ₅	q ₅
q ₅	q ₁	q ₁

We can consider the string "abbb". This string is a valid string as it is not divisible by 5 i.e $abbb \text{ mod } 5 \neq 0$. That also means that we should reach to final state on acceptance of this string. The simulation of this string for given DFA is :

$$\begin{aligned}\delta(q_1, abbb) &\vdash \delta(q_2, bbb) \\ &\vdash \delta(q_3, bb) \\ &\vdash \delta(q_4, b) \\ &\vdash \delta(q_5, \epsilon)\end{aligned}$$

where q₅ is a final state.

Now consider string "ababa" which is invalid string as it is divisible by 5. That means ababa mod 5 = 0. That means we should reach to non final state on acceptance of this string. The simulation for this string is as given below.

As q₁ is a start we will start from state q₁.

$$\begin{aligned}\delta(q_1, ababa) &\vdash \delta(q_2, bab) \\ &\vdash \delta(q_3, aba) \\ &\vdash \delta(q_4, ba) \\ &\vdash \delta(q_5, a) \\ &\vdash \delta(q_1, \epsilon)\end{aligned}$$

Here q₁ is a non-final state. That means "ababa" is invalid string for the given DFA.

⇒ Example 2.6.12 : Design a DFA $L(M) = \{W \mid W \in \{0,1\}^*\}$ and W is a string that does not contain consecutive 1's.

Solution : When three consecutive 1's occur the DFA will be :

Here two consecutive 1's or single 1 is acceptable, hence :

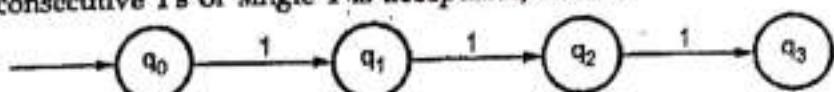


Fig. 2.6.15

The states q_0, q_1, q_2 are final states. Hence DFA M can be represented as,

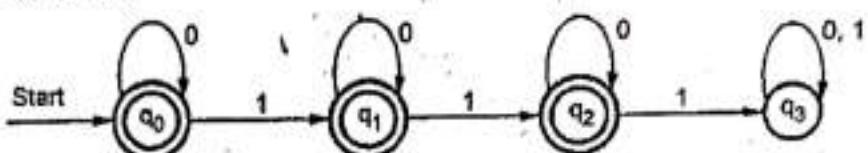


Fig. 2.6.16

$$M = [Q, \Sigma, q_0, F]$$

where

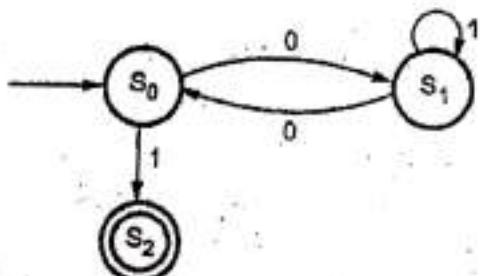
$$Q = \{q_0, q_1, q_2, q_3\}$$

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

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

→ **Example 2.6.13 :** Design a DFA which accepts strings with even number of 0's followed by single 1 over $\Sigma = \{0, 1\}$.

Solution : The DFA can be shown by a transition diagram as :

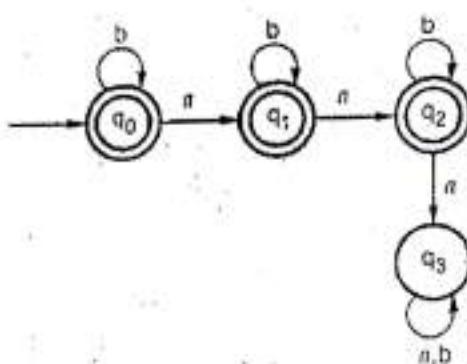


Here state S_1 will accept all the odd number of 0's and S_0 will accept all even number of 0's. It should then be followed by single 1. Hence S_2 is a final state. Consider the input :

0 0 0 1 0 1
0 S_1 0 0 1 0 1
0 0 S_0 0 1 0 1
0 0 0 S_1 1 0 1
0 0 0 1 S_1 0 1
0 0 0 1 0 S_0 1
0 0 0 1 0 1 S_2

→ Example 2.6.14 : Design DFA which accepts all the strings not having more than two a's over $\Sigma = \{a, b\}$.

Solution : In this DFA maximum two a's are accepted. If we try to accept third a then it should not lead us to final state. Such a DFA can be as shown below.



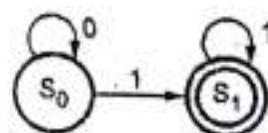
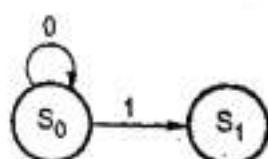
The transition table can be as shown -

States \ Input	a	b
→ q_0	q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_2
q_3	q_3	q_3

Transition table

→ Example 2.6.15 : Design a DFA for accepting all the strings of $\{L = 0^m 1^n | m \geq 0 \text{ and } n \geq 1\}$.

Solution : This is a language in which all the 0's followed by all 1's. But number of zero's and number of 1's are different. The language explicitly mentions that there should be at least one 1. Any number of 0's followed by only one 1 is -



But we have 1^n in the language. Hence the DFA can be as shown in this figure.

The transition table can be

State \ Input	0	1
S ₀	S ₀	S ₁
S ₁	-	S ₁

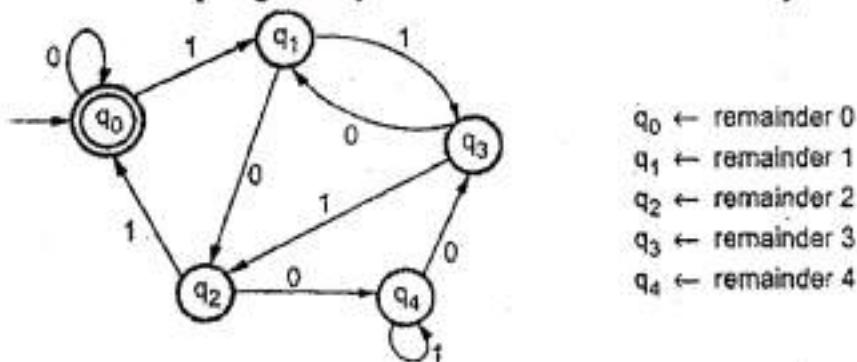
Consider a string 00111, which is a valid string and is accepted by above drawn DFA.

$$\begin{aligned}\delta(S_0, 00111) &\vdash (S_0, 0111) \\ &\vdash (S_0, 111) \\ &\vdash (S_1, 11) \\ &\vdash (S_1, 1) \\ &\vdash (S_1, \epsilon)\end{aligned}$$

Thus we reach to final state S₁ on acceptance of 00111.

► Example 2.6.16 : Give the DFA accepting the following language over alphabet {0,1} L = 'Set of all strings beginning with 1 that, when interpreted as a binary integer, is a multiple of 5.' For example, strings 101, 1010, and 1111 in the language; 0, 100; and 111 not.

Solution : The DFA for accepting binary number which is divisible by 5. is as follows -



Note that for accepting string

10100 i.e. 20

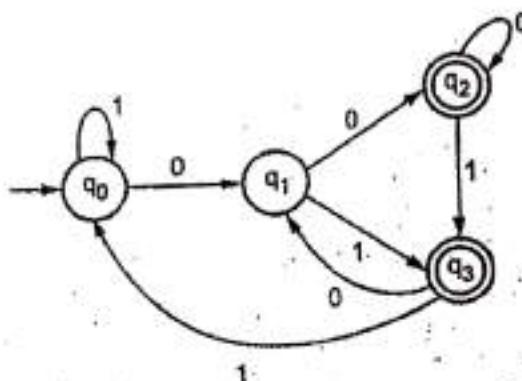
q ₀	10100
1 q ₁	0100
10 q ₂	100
101 q ₀	00
1010 q ₀	0
10100	accept state

→ **Example 2.6.17 :** Write definition of finite automata and draw FA for the strings :

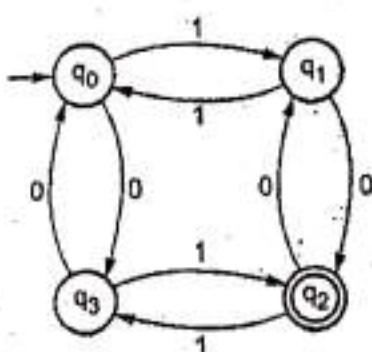
- The strings with next to last symbol as 0
- The string with number of 0's odd and number of 1's odd. **GTU : Summer-11, Marks 7**

Solution : i) The term next to last means last but one. Hence for given languages the string 01, 00 are valid. The set of strings can be {01, 00, 000, 001, 101, ...}

The FA will be



ii)



→ **Example 2.6.18 :** Write definition of finite automata and draw FA for the strings :

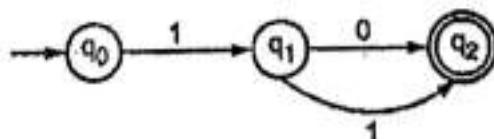
- The string in $\{0,1\}^*$ ending in 10 or 11.
- The string corresponding to regular expression $\{11\}^* \{00\}^*$ **GTU : Winter-12, Marks 7**

Ans. : i) The regular expression for given language is

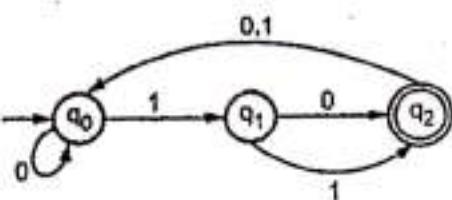
$$\text{r.e. } = (0+1)^*(10+11)$$

The finite automata will be -

Step 1 : We will draw the FA for the posterior part of the given regular expression.



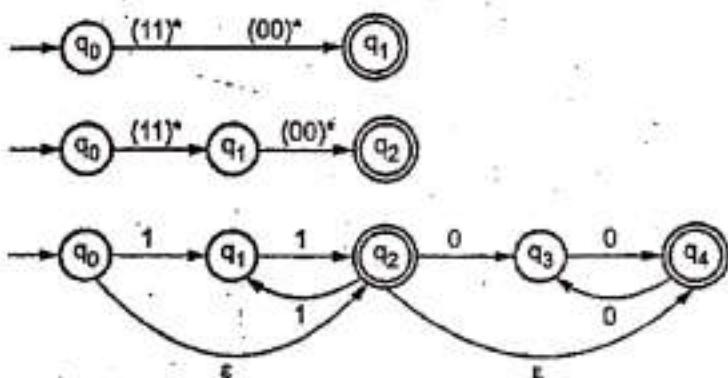
Step 2 : Now, we will add up the transitions for prefix $(0+1)^*$. Hence the required FA will be



ii) r.e. = $(11)^*(00)^*$

The FA will be

Step 1 :

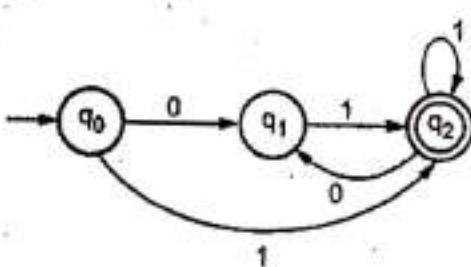


→ **Example 2.6.19 :** Draw FA for accepting :

- The string in $(0, 1)^*$ ending in 1 and not containing substring 00.
- The strings with odd no. of 1's and odd no. of 0's.

GTU : Winter-13, Marks 6, Summer-16, Marks 7

Solution : i) FA for accepting strings ending with 1 and not containing 00.



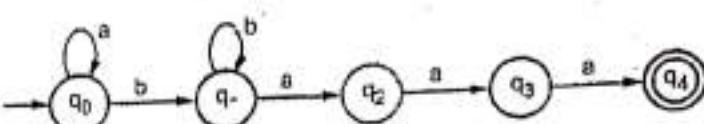
ii) Refer example 2.41 (ii).

→ **Example 2.6.20 :** Attempt the following :

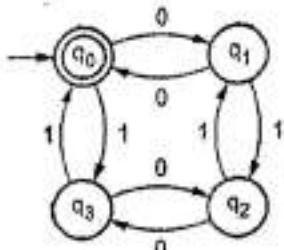
- Draw FA for $(a + b)^* baab$.
- Write a regular expression for the string of 0's and 1's in which number of 0's and 1's are even.

Solution :

i)



ii)



$$\text{r.e.} = (00 + 11)^* + (01(11)^* 0(00)^* 1(11)^* \\ + (10(00)^* 1(11)^* 0(00)^*)$$

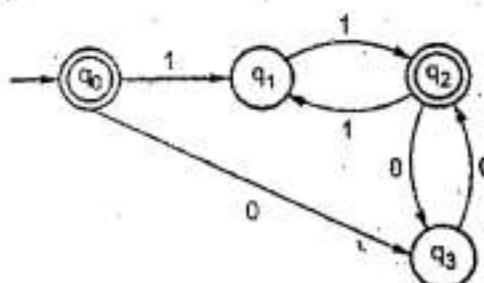
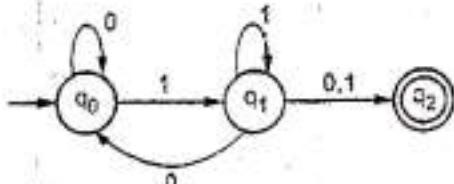
→ **Example 2.6.21 :** Write definition of finite automata and draw FA for the strings :

- The string in $\{0, 1\}^*$ ending in 10 or 11.
- The string corresponding to regular expression $(11)^*(00)^*$.

GTU : Summer-15, Marks 7**Solution :**

i)

iii) This language accepts the strings such as
 $\{\epsilon, 11, 1100, 1111, 111111, 00, 0000, \dots\}$



2.7 Non Deterministic Finite Automata

The concept of Non deterministic Finite Automata is exactly reverse of Deterministic Finite Automata. The Finite Automata is called NFA when there exists many paths for a specific input from current state to next state. The NFA can be shown as in Fig. 2.7.1.

Note that the NFA shows from q_0 for input a there are two next states q_1 and q_2 . Similarly, from q_0 for input b the next states are q_0 and q_1 .

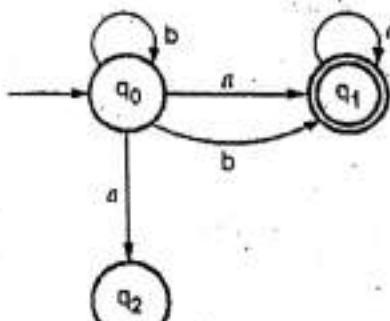


Fig. 2.7.1 Non deterministic finite automata

Thus it is not fixed or determined that with a particular input where to go next. Hence this FA is called non deterministic finite automata.

Consider the input string bba . This string can be derived as

	Input	b	b	a
	Path	q_0	q_0	q_1
or	Input	b	b	a
	Path	q_0	q_0	q_2
or	Input	b	b	a
	Path	q_0	q_1	q_1

Thus you cannot take the decision of which path has to be followed for deriving the given string.

Definition of NFA

The NFA can be formally defined as a collection of 5-tuples.

- 1) Q is a finite set of states.
- 2) Σ is a finite set of inputs.
- 3) δ is called next state or transition function.
- 4) q_0 is initial state.
- 5) F is a final state where $F \subseteq Q$.

There can be multiple final states. Thus the next question might be what is the use of NFA. The NFA is basically used in theory of computations because they are more flexible and easier to use than the DFAs.

Difference between NFA and DFA

Sr. No.	NFA	DFA
1.	NFA stands for non deterministic finite automata.	DFA stands for deterministic finite automata.
2.	NFA can use empty string transition.	DFA can not use empty string transition.
3.	For every input symbol, there can be more than one transition.	For every input symbol there can be only one state transition.
4.	Construction of NFA is simple.	Construction of DFA is difficult.

Example 2.7.1 : Compare FA, NFA and NFA- Δ .

GTU : Winter-17, Marks 3

Solution :

Sr. No.	FA	NFA	NFA - \cup
1.	It is deterministic in nature.	It is non-deterministic in nature.	It is non-deterministic in nature.
2.	It is difficult to design.	It is easy to design due to non-determinism.	It is easy and flexible in design.
3.	It is easy to recognize language accepted by FA.	It is difficult to recognize language accepted by NFA.	Using ϵ transitions the union or concatenated languages are accepted by this NFA.

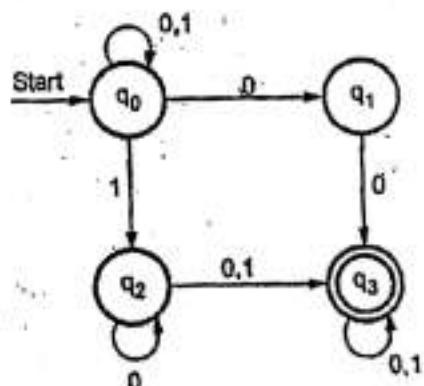
→ **Example 2.7.2 :** Design the NFA transition diagram for the transition table as given below -

State \ Input	0	1
q ₀	{q ₀ , q ₁ }	{q ₀ , q ₂ }
q ₁	{q ₃ }	
q ₂	{q ₂ , q ₃ }	{q ₃ }
q ₃	{q ₃ }	{q ₃ }

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

Solution : The transition diagram can be drawn by using the mapping function as given in table.

$$\begin{aligned}\delta(q_0, 0) &= \{q_0, q_1\} \\ \delta(q_0, 1) &= \{q_0, q_2\} \\ \text{Then, } \delta(q_1, 0) &= \{q_3\} \\ \text{Then, } \delta(q_2, 0) &= \{q_2, q_3\} \\ \delta(q_2, 1) &= \{q_3\} \\ \text{Then, } \delta(q_3, 0) &= \{q_3\} \\ \delta(q_3, 1) &= \{q_3\}\end{aligned}$$



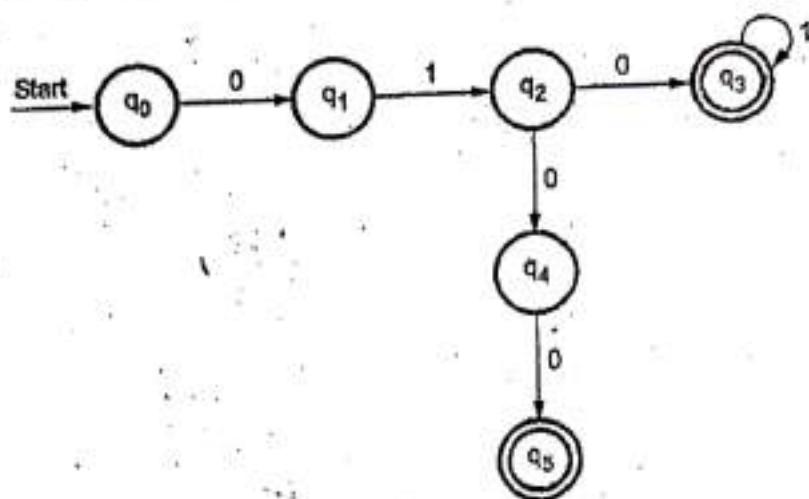
→ **Example 2.7.3 :** Construct NFA for the language

$$L = \{0101^n \cup 0100 \mid n \geq 0\}$$

Over

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

Solution : Here in language L first three symbols are common i.e. 010. Hence the NFA can be drawn as

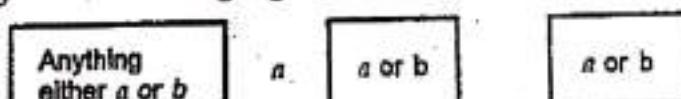


The states q_3 and q_5 are final states accepting 0101^n and 0100 respectively. The NFA then can be denoted by,

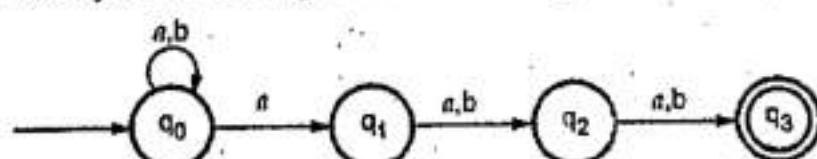
$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \delta, q_0, \{q_3, q_5\})$$

→ **Example 2.7.4 :** Construct a NFA for a language L which accepts all the strings in which the third symbol from right end is always a. Over $\Sigma = \{a, b\}$.

Solution : The strings in such a language are of the form



Thus we get third symbol from right end as 'a' always. The NFA then can be



The above figure is a NFA because in state q_0 with input 'a' we can go to either state q_0 or state q_1 .

2.8 NFA with ϵ

The language L accepted by NFA with ϵ , denoted by $M = (Q, \Sigma, \delta, q_0, F)$ can be defined as :

Let, $M = (Q, \Sigma, \delta, q_0, F)$ be a NFA with ϵ .

where

Q is a finite set of states.

Σ is input set.

δ is a transition or a mapping function for transitions from $Q \times \{\Sigma \cup \epsilon\}$ to 2^Q .

q_0 is a start state.

F is a set of final states such that $F \subseteq Q$.

The string w in L accepted by NFA can be represented as

$L(M) = \{w \mid w \in \Sigma^* \text{ and } \delta \text{ transition for } w \text{ from } q_0 \text{ reaches to } F\}$.

→ Example 2.8.1 : Construct NFA with ϵ which accepts a language consisting the strings of any number of a's followed by any number of b's. Followed by any number of c's.

Solution : Here any number of a's or b's or c's means zero or more in number. That means there can be zero or more a's followed by zero or more b's followed by zero or more c's. Hence NFA with ϵ can be -

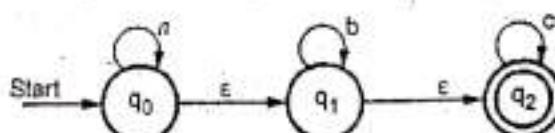


Fig. 2.8.1

Normally ϵ 's are not shown in the input string. The transition table can be -

		Σ			
		Σ			
		a	b	c	ϵ
State	Input	q_0	\emptyset	\emptyset	q_1
	ϵ	\emptyset	q_1	\emptyset	q_2
q_2	ϵ	\emptyset	\emptyset	q_2	\emptyset

We can parse the string aabbcc as follows -

$\delta(q_0, aabbcc)$ $\vdash \delta(q_0, abbcc)$
 $\vdash \delta(q_0, bbcc)$
 $\vdash \delta(q_0, \epsilon bbcc)$
 $\vdash \delta(q_1, bbcc)$
 $\vdash \delta(q_1, bcc)$
 $\vdash \delta(q_1, cc)$
 $\vdash \delta(q_1, \epsilon cc)$
 $\vdash \delta(q_2, cc)$
 $\vdash \delta(q_2, c)$

$$\vdash \delta(q_2, \epsilon)$$

Thus we reach to accept state, after scanning the complete input string.

Definition of ϵ -closure

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

- i) ϵ -closure (p) = p where $p \in Q$.
- ii) If there exists ϵ -closure (p) = $\{q\}$ and $\delta(q, \epsilon) = r$ then ϵ -closure (p) = $\{q, r\}$.

→ **Example 2.8.2 :** Find ϵ -closure for the following NFA with ϵ .

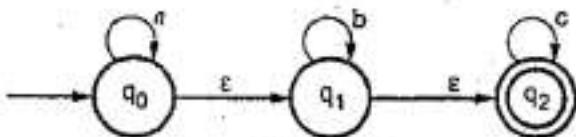


Fig. 2.8.2

Solution :

ϵ -closure (q_0) = $\{q_0, q_1, q_2\}$ means self state + ϵ -reachable states.

ϵ -closure (q_1) = $\{q_1, q_2\}$ means q_1 is a self state and q_2 is a state obtained from q_1 with ϵ input.

ϵ -closure (q_2) = $\{q_2\}$

2.9 Conversion of NFA with ϵ to NFA without ϵ

In this method we try to remove all the ϵ transitions from given NFA. The method will be

1. Find out all the ϵ transitions from each state from Q . That will be called as ϵ -closure $\{q_i\}$ where $q_i \in Q$.
2. Then δ' transitions can be obtained. The δ' transitions 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 can be built.

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

Proof :

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

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

$$F' = F \cup \{q_0\} \text{ if } \epsilon\text{-closure contains state off}$$

$$F \quad \text{otherwise}$$

M' is a NFA without ϵ moves. The δ' function can be denoted by δ'' with some input. For example, $\delta'(q, a) = \delta''(q, a)$ for some q in Q and a from Σ . We will apply the method of induction with input x . The x will not be ϵ because

$$\delta'(q_0, \epsilon) = \{q_0\}$$

$\delta''(q_0, \epsilon) = \epsilon\text{-closure}(q_0)$. Therefore we will assume length of string to be 1.

Basis : $|x| = 1$. Then x is a symbol a .

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

Induction : $|x| > 1$ Let $x = wa$

$$\delta'(q_0, wa) = \delta'(\delta'(q_0, w), a)$$

By inductive hypothesis,

$$\delta'(q_0, w) = \delta''(q_0, w) = p$$

Now we will show that $\delta'(p, a) = \delta(q_0, wa)$

$$\text{But } \delta'(p, a) = \cup_{q \text{ in } p} \delta'(q, a) = \cup_{q \text{ in } p} \delta''(q, a)$$

$$\text{As } p = \delta''(q_0, w)$$

$$\text{We have, } \cup \delta''(q, a) = \delta''(q_0, wa)$$

$$q \text{ in } p$$

Thus by definition δ''

$$\delta'(q_0, wa) = \delta''(q_0, wa)$$

Rule for conversion

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

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

→ Example 2.9.1 : Convert the given NFA with ϵ to NFA without ϵ .

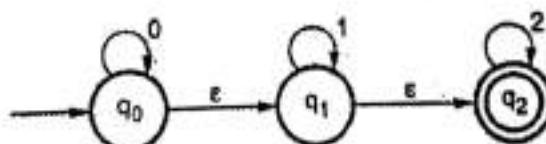


Fig. 2.9.1

Solution : We will first obtain ϵ -closure of each state i.e. we will find out ϵ -reachable states from current state.

Hence

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

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

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

As ϵ -closure(q_0) means with null input (no input symbol) we can reach to q_0 , q_1 or q_2 . In a similar manner for q_1 and q_2 ϵ -closures are obtained. Now we will obtain δ' transitions for each state on each input symbol.

$$\begin{aligned}\delta'(q_0, 0) &= \epsilon\text{-closure}(\delta(\delta(q_0, \epsilon), 0)) \\&= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0), 0)) \\&= \epsilon\text{-closure}(\delta(q_0, q_1, q_2), 0) \\&= \epsilon\text{-closure}(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)) \\&= \epsilon\text{-closure}(q_0 \cup \phi \cup \phi) \\&= \epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\} \\ \delta'(q_0, 1) &= \epsilon\text{-closure}(\delta(\delta(q_0, \epsilon), 1)) \\&= \epsilon\text{-closure}(\delta(q_0, q_1, q_2), 1) \\&= \epsilon\text{-closure}(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)) \\&= \epsilon\text{-closure}(\phi \cup q_1 \cup \phi) \\&= \epsilon\text{-closure}(q_1) \\ \delta'(q_0, 1) &= \{q_1, q_2\} \\ \delta'(q_1, 0) &= \epsilon\text{-closure}(\delta(\delta(q_1, \epsilon), 0)) \\&= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1), 0)) \\&= \epsilon\text{-closure}(\delta(q_1, q_2), 0) \\&= \epsilon\text{-closure}(\delta(q_1, 0) \cup \delta(q_2, 0)) \\&= \epsilon\text{-closure}(\phi \cup \phi) \\&= \epsilon\text{-closure}(\phi) \\&= \phi \\ \delta'(q_1, 1) &= \epsilon\text{-closure}(\delta(\delta(q_1, \epsilon), 1)) \\&= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1), 1)) \\&= \epsilon\text{-closure}(\delta(q_1, q_2), 1)\end{aligned}$$

$$\begin{aligned}
 &= \text{closure}(\delta(q_1, \cdot) \cup \delta(q_2, \cdot)) \\
 &= \text{closure}(q_1 \cup \emptyset) \\
 &= \text{closure}(q_1) \\
 &= \{q_1, q_2\} \\
 \delta'(q_2, 0) &= \text{closure}(\delta(\delta(q_2, \epsilon), 0)) \\
 &= \text{closure}(\delta(\text{closure}(q_2), 0)) \\
 &= \text{closure}(\delta(q_2, 0)) \\
 &= \text{closure}(\emptyset) \\
 &= \emptyset \\
 \delta'(q_2, 1) &= \text{closure}(\delta(\delta(q_2, \epsilon), 1)) \\
 &= \text{closure}(\delta(\text{closure}(q_2), 1)) \\
 &= \text{closure}(\delta(q_2, 1)) \\
 &= \text{closure}(\emptyset) \\
 \delta'(q_2, 2) &= \emptyset \\
 \delta'(q_0, 2) &= \text{closure}(\delta(\delta(q_0, \epsilon), 2)) \\
 &= \text{closure}(\delta(\text{closure}(q_0), 2)) \\
 &= \text{closure}(\delta(q_0, q_1, q_2), 2) \\
 &= \text{closure}(\delta(q_0, 2) \cup \delta(q_1, 2) \cup \delta(q_2, 2)) \\
 &= \text{closure}(\emptyset \cup \emptyset \cup q_2) \\
 &= \text{closure}(q_2) \\
 &= \{q_2\} \\
 \delta'(q_1, 2) &= \text{closure}(\delta(\delta(q_1, \epsilon), 2)) \\
 &= \text{closure}(\delta(\text{closure}(q_1), 2)) \\
 &= \text{closure}(\delta(q_1, q_2), 2) \\
 &= \text{closure}(\delta(q_1, 2) \cup \delta(q_2, 2)) \\
 &= \text{closure}(\emptyset \cup q_2) \\
 &= \{q_2\}
 \end{aligned}$$

$$\begin{aligned}
 \delta'(q_2, 2) &= \text{e-closure}(\delta(\delta(q_2, \epsilon), 2)) \\
 &= \text{e-closure}(\delta(\text{e-closure}(q_2), 2)) \\
 &= \text{e-closure}(\delta(q_2, 2)) \\
 &= \text{e-closure}(q_2) \\
 &= \{q_2\}
 \end{aligned}$$

Now we will summarize all the computed δ' transitions -

$$\delta'(q_0, 0) = \{q_0, q_1, q_2\}, \quad \delta'(q_0, 1) = \{q_1, q_2\}, \quad \delta'(q_0, 2) = \{q_2\}$$

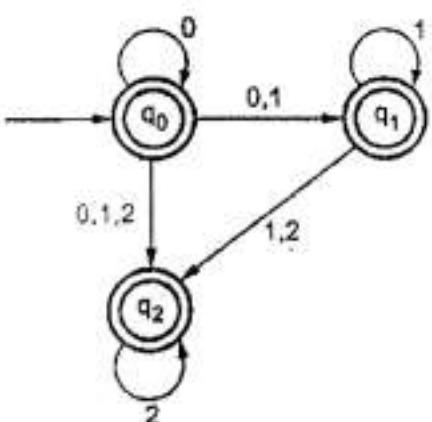
$$\delta'(q_1, 0) = \emptyset, \quad \delta'(q_1, 1) = \{q_1, q_2\}, \quad \delta'(q_1, 2) = \{q_2\}$$

$$\delta'(q_2, 0) = \emptyset, \quad \delta'(q_2, 1) = \emptyset, \quad \delta'(q_2, 2) = \{q_2\}$$

From this we can write the transition table as -

State \ Input	0	1	2
q ₀	{q ₀ , q ₁ , q ₂ }	{q ₁ , q ₂ }	{q ₂ }
q ₁	∅	{q ₁ , q ₂ }	{q ₂ }
q ₂	∅	∅	{q ₂ }

The NFA will be -



Here q₀, q₁ and q₂ is a final state because e-closure(q₀), e-closure(q₁) and e-closure(q₂) contains final state q₂.

Fig. 2.9.2

→ Example 2.9.2 : Give recursive definitions of the extended transition functions, δ^* (i.e., for strings) for DFA and NFA.

GTU : Summer-17, Marks 4

Solution : The δ' transitions mean e-closure on δ moves. The definition of e-closure is as follows :

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

- ϵ -closure (p) = p where $p \in Q$
- If there exists ϵ -closure (p) = $\{q\}$ and $\delta(q, \epsilon) = r$ then ϵ -closure (p) = $\{q, r\}$.

Review Question

- Define NFA - Δ . Explain how to convert NFA - Δ into NFA and FA with suitable example.

GTU : Summer-16, Marks 7

2.10 Conversion of NFA to DFA

Here is a theorem which tells you that any NFA can be converted to its equivalent DFA. That is any language L acceptable by NFA can be acceptable by its equivalent DFA. The basic idea in this theorem is that, DFA keeps track of all the states, that NFA could be in reading the same input as the DFA has read.

Let us see the theorem.

Theorem : Let L be a set accepted by non deterministic finite automation. Then there exists a deterministic finite automation that accepts L .

Proof : Let

$M = (Q, \Sigma, \delta, q_0, F)$ be an NFA for language L . Then define DFA M' such that

$$M' = (Q', \Sigma, \delta', q'_0, F')$$

The states of M' are all the subset of M . The $Q' = 2^Q$.

F' be the set of all the final states in M' .

The elements in Q' will be denoted by $[q_1, q_2, q_3, \dots, q_i]$ and the elements in Q are denoted by $\{q_0, q_1, q_2, \dots\}$. The $[q_1, q_2, \dots, q_i]$ will be assumed as one state in Q' if in the NFA q_0 is a initial state it is denoted in DFA as $q'_0 = [q_0]$. We define,

$$\delta'([q_1, q_2, q_3, \dots, q_i], a) = [p_1, p_2, p_3, \dots, p_j]$$

if and only if,

$$\delta(\{q_1, q_2, q_3, \dots, q_i\}, a) = \{p_1, p_2, p_3, \dots, p_j\}$$

This means that whenever in NFA, at the current states $\{q_1, q_2, q_3, \dots, q_i\}$ if we get input a and it goes to the next states $\{p_1, p_2, \dots, p_j\}$ then while constructing DFA for it the current state is assumed to be $[q_1, q_2, q_3, \dots, q_i]$. At this state, the input is a and the next is assumed to be $[p_1, p_2, \dots, p_j]$. On applying δ function on each of the states $q_1, q_2, q_3, \dots, q_i$ the new states may be any of the states from $[p_1, p_2, \dots, p_j]$. The theorem can be proved with the induction method by assuming length of input string x

$$\delta'(q'_0, x) = [q_1, q_2, \dots, q_l]$$

if and only if,

$$\delta(q_0, x) = \{q_1, q_2, q_3, \dots, q_l\}$$

Basis : If length of input string is 0

i.e. $|x| = 0$, that means x is ϵ then $q'_0 = [q_0]$

Induction : If we assume that the hypothesis is true for the input string of length m or less than m . Then if $x a$ is a string of length $m+1$. Then the function δ' could be written as

$$\delta'(q'_0, x a) = \delta(\delta'(q_0, x), a)$$

By the induction hypothesis,

$$\delta'(q'_0, x) = [p_1, p_2, \dots, p_j]$$

if and only if,

$$\delta(q_0, x) = \{p_1, p_2, p_3, \dots, p_j\}$$

By definition of δ'

$$\delta'([p_1, p_2, \dots, p_j], a) = [r_1, r_2, \dots, r_k]$$

if and only if,

$$\delta(\{p_1, p_2, \dots, p_j\}, a) = \{r_1, r_2, \dots, r_k\}$$

Thus

$$\delta'(q'_0, x a) = [r_1, r_2, \dots, r_k]$$

if and only if

$$\delta(q_0, x a) = \{r_1, r_2, \dots, r_k\}$$

is shown by inductive hypothesis.

Thus $L(M) = L(M')$

With the help of this theorem, let us try to solve few examples.

2.10.1 NFA to DFA

We will discuss the method of converting NFA to its equivalent DFA.

Let, $M = (Q, \Sigma, \delta, q_0, F)$ is a NFA which accepts the language $L(M)$. There should be equivalent DFA denoted by $M' = (Q', \Sigma', \delta', q'_0, F')$ such that $L(M) = L(M')$.

The conversion method will follow following steps -

- 1) The start state of NFA M will be the start for DFA M'. Hence add q_0 of NFA (start state) to Q' . Then find the transitions from this start state.
- 2) For each state $[q_1, q_2, q_3, \dots, q_i]$ in Q' the transitions for each input symbol Σ can be obtained as,
 - i) $\delta'([q_1, q_2, \dots, q_i], a) = \delta(q_1, a) \cup \delta(q_2, a) \cup \dots \cup \delta(q_i, a)$
 $[q_1, q_2, \dots, q_k]$ may be some state.
 - ii) Add the state $[q_1, q_2, \dots, q_k]$ to DFA if it is not already added in Q' .
 - iii) Then find the transitions for every input symbol from Σ for state $[q_1, q_2, \dots, q_k]$. If we get some state $[q_1, q_2, \dots, q_n]$ which is not in Q' of DFA then add this state to Q' .
 - iv) If there is no new state generating then stop the process after finding all the transitions.
- 3) For the state $[q_1, q_2, \dots, q_n] \in Q'$ of DFA if any one state q_i is a final state of NFA then $[q_1, q_2, \dots, q_n]$ becomes a final state. Thus the set of all the final states $\in F'$ of DFA.

Example 2.10.1 : Let $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$.

be NFA where $\delta(q_0, 0) = \{q_0, q_1\}$, $\delta(q_0, 1) = \{q_1\}$

$\delta(q_1, 0) = \emptyset$, $\delta(q_1, 1) = \{q_0, q_1\}$

Construct its equivalent DFA.

Solution : Let the DFA $M' = (Q', \Sigma, \delta', q'_0, F')$.

Now, the δ' function will be computed as follows -

As $\delta(q_0, 0) = \{q_0, q_1\}$ $\delta'([q_0], 0) = [q_0, q_1]$

As in NFA the initial state is q_0 , the DFA will also contain the initial state $[q_0]$.

Let us draw the transition table for δ function for a given NFA.

		Input	0	1	
		State	q_0	q_1	
$\delta(q_0, 0)$	\Rightarrow	$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_1\}$	$= \delta(q_0, 0)$
$\delta(q_1, 0)$	\Rightarrow	(q_1)	\emptyset	$\{q_0, q_1\}$	$= \delta(q_1, 0)$

δ Function for NFA

From the transition table we can compute that there are $[q_0]$, $[q_1]$, $[q_0, q_1]$ states for its equivalent DFA. We need to compute the transition from state $[q_0, q_1]$.

$$\begin{aligned}\delta(\{q_0, q_1\}, 0) &= \delta(q_0, 0) \cup \delta(q_1, 0) \\ &= \{q_0, q_1\} \cup \emptyset \\ &= \{q_0, q_1\}\end{aligned}$$

So, $\delta'([q_0, q_1], 0) = [q_0, q_1]$

Similarly,

$$\begin{aligned}\delta(\{q_0, q_1\}, 1) &= \delta(q_0, 1) \cup \delta(q_1, 1) \\ &= \{q_1\} \cup \{q_0, q_1\} \\ &= \{q_0, q_1\}\end{aligned}$$

So, $\delta'([q_0, q_1], 1) = [q_0, q_1]$

As in the given NFA q_1 is a final state, then in DFA wherever q_1 exists that state becomes a final state. Hence in the DFA final states are $[q_1]$ and $[q_0, q_1]$. Therefore set of final states $F = \{[q_1], [q_0, q_1]\}$.

The equivalent DFA is -

State	Input	
	0	1
$\rightarrow q_0$	$[q_0, q_1]$	$[q_1]$
$([q_1])$	\emptyset	$[q_0, q_1]$
$([q_0, q_1])$	$[q_0, q_1]$	$[q_0, q_1]$

Transition table for equivalent DFA

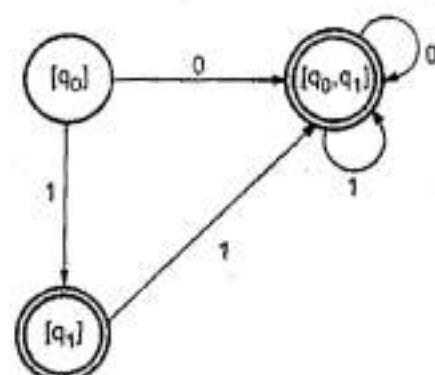


Fig. 2.10.1

Even we can change the names of the states of DFA.

$$A = [q_0]$$

$$B = [q_1]$$

$$C = [q_0, q_1]$$

With these new names the DFA will be as follows -

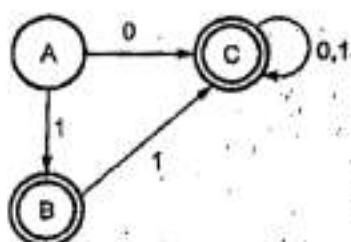


Fig. 2.10.2 An equivalent DFA

→ Example 2.10.2 : Construct DFA equivalent to the given NFA.

State \ Input	0	1
State		
→ p	{p, q}	p
q	r	r
r	s	-
s	s	s

Solution : The NFA M = ({p, q, r, s}, {0, 1}, δ, {p}, {s})

The equivalent DFA will be constructed.

State \ Input	0	1
State		
→ [p]	[p, q]	[p]
[q]	[r]	[r]
[r]	[s]	-
([s])	[s]	[s]
[p, q]	[p, q, r]	[p, r]

Continuing with the generated new states.

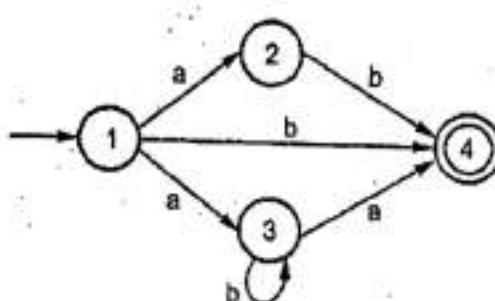
State \ Input	0	1
State	0	1
$\rightarrow [p]$	$[p, q]$	$[p]$
$[q]$	$[r]$	$[r]$
$[r]$	$[s]$	-
$([s])$	$[s]$	$[s]$
$[p, q]$	$[p, q, r]$	$[p, r]$
$[p, q, r]$	$[p, q, r, s]$	$[p, s]$
$[p, r]$	$[p, q, s]$	$[p]$
$([p, q, r, s])$	$[p, q, r, s]$	$[p, r, s]$
$([p, q, s])$	$[p, q, r, s]$	$[p, r, s]$
$([p, r, s])$	$[p, q, s]$	$[p, s]$
$([p, s])$	$[p, q, s]$	$[p, s]$

The final state $F' = \{ [s], [p, q, r, s], [p, q, s], [p, r, s], [p, s] \}$

The transition graph shows two disconnected parts. But part I will be accepted as final DFA because it consists of start state and final states, in part II there is no start state. (See Fig. 2.10.3 on next page)

→ Example 2.10.3 : Convert the following NFA into FA.

GTU : Summer-12, Marks 7



Solution : We will draw the transition table for given NFA.

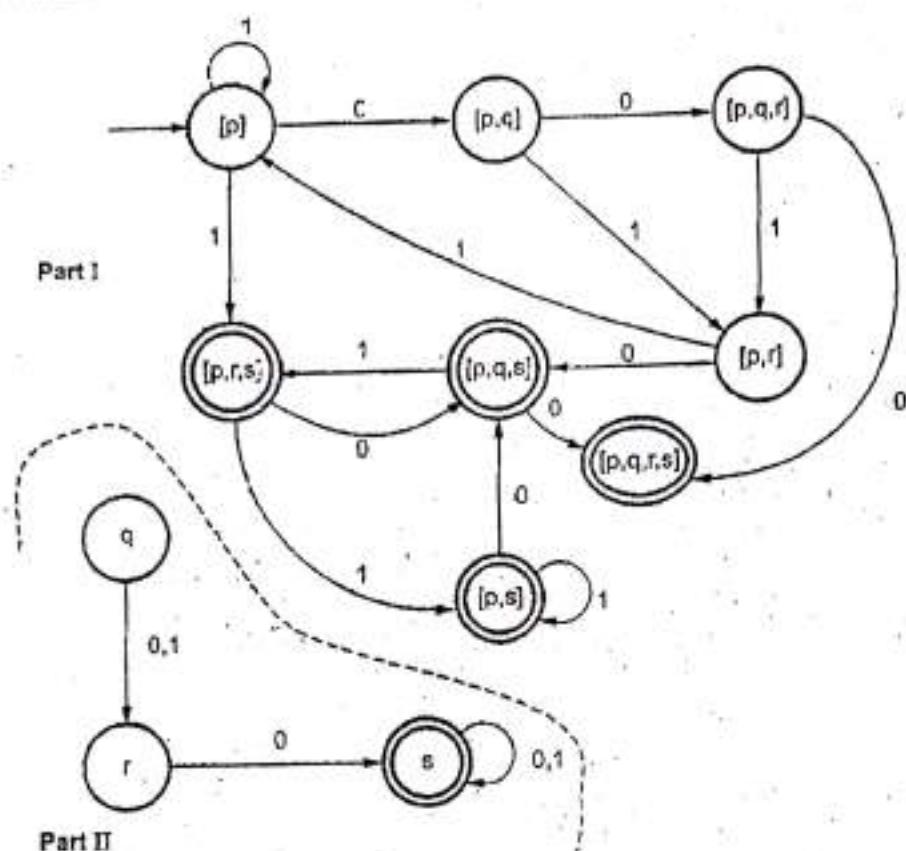


Fig. 2.10.3

State \ Input	a	b
1	{2, 3}	4
2	\emptyset	4
3	4	3
4	\emptyset	\emptyset

Now we will obtain δ transition for each state and input pair.

$$\delta(1, a) = \{2, 3\} = [2, 3] \text{ a new state.}$$

$$\delta(1, b) = [4]$$

$$\delta(2, a) = \emptyset$$

$$\delta(2, b) = [4]$$

$$\delta(3, a) = [4]$$

$$\delta(3, b) = \{3\}$$

$$\delta(4, a) = \delta(4, b) = \emptyset$$

As the new state $[2, 3]$ is being generated. We will obtain value of δ function for this state.

$$\delta([2, 3], a) = \{4\}$$

$$\delta([2, 3], b) = \{3, 4\} = [3, 4] \leftarrow \text{New state}$$

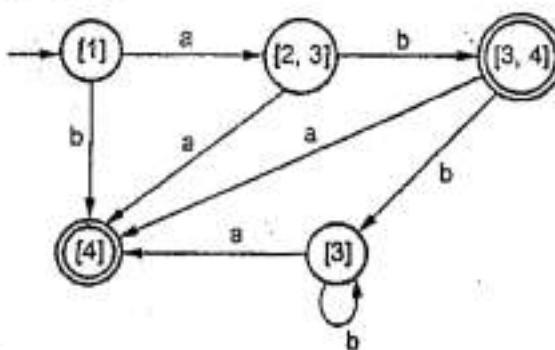
$$\delta([3, 4], a) = \{4\}$$

$$\delta([3, 4], b) = \{3\}$$

As no new state getting generated, the DFA can be represented using transition table as -

Input State \	a	b
[1]	[2, 3]	{4}
[2]	\emptyset	[4]
[3]	[4]	[3]
(4)	\emptyset	\emptyset
[2, 3]	[4]	[3, 4]
(3, 4)	[4]	[3]

The transition diagram will be -



Note that state 2 can be eliminated as it is a dead state.

2.10.2 NFA with ϵ to DFA

Method for converting NFA with ϵ to DFA

Step 1 : Consider $M = (Q, \Sigma, \delta, q_0, F)$ is a NFA with ϵ . We have to convert this NFA with ϵ to equivalent DFA denoted by

$$M_D = (Q_D, \Sigma, \delta_D, q_0, F_D)$$

Then obtain,

ϵ -closure(q_0) = $\{p_1, p_2, p_3, \dots, p_n\}$ then $[p_1, p_2, p_3, \dots, p_n]$ becomes a start state of DFA.

Now $[p_1, p_2, p_3, \dots, p_n] \in Q_D$

Step 2 : We will obtain δ transitions on $[p_1, p_2, p_3, \dots, p_n]$ for each input.

$$\delta_D([p_1, p_2, \dots, p_n], a) = \epsilon\text{-closure}(\delta(p_1, a) \cup \delta(p_2, a) \cup \dots \cup \delta(p_n, a))$$

$$= \bigcup_{i=1}^n \epsilon\text{-closure}(\delta(p_i, a))$$

where a is input $\in \Sigma$.

Step 3 : The states obtained $[p_1, p_2, p_3, \dots, p_n] \in Q_D$. The states containing final state in p_i is a final state in DFA.

Now let us see some examples of conversion based on this procedure.

► Example 2.10.4 : Convert the given NFA into its equivalent DFA -

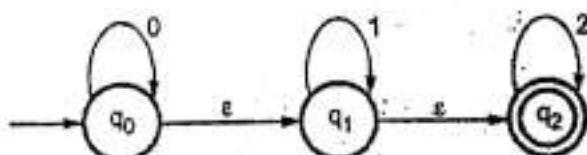


Fig. 2.10.4

Solution : Let us obtain ϵ -closure of each state.

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

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

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

Now we will obtain δ' transition. Let $\epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\}$ call it as state A.

$$\begin{aligned} \delta'(A, 0) &= \epsilon\text{-closure}\{\delta(\{q_0, q_1, q_2\}, 0)\} \\ &= \epsilon\text{-closure}\{\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)\} \\ &= \epsilon\text{-closure}\{q_0\} \\ &= \{q_0, q_1, q_2\} \end{aligned}$$

i.e. state A

$$\begin{aligned} \delta'(A, 1) &= \epsilon\text{-closure}\{\delta(\{q_0, q_1, q_2\}, 1)\} \\ &= \epsilon\text{-closure}\{\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)\} \end{aligned}$$

$$= \epsilon\text{-closure } \{q_1\}$$

Call it as state B

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

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

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

$$= \epsilon\text{-closure } \{q_2\}$$

Call it as state C.

$$= \{q_2\}$$

Thus we have obtained

$$\delta'(A, 0) = A$$

$$\delta'(A, 1) = B$$

$$\delta'(A, 2) = C$$

i.e.

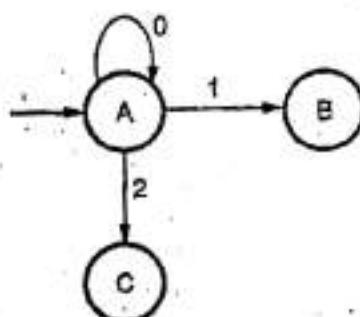


Fig. 2.10.5

Now we will find transitions on states B and C for each input.

Hence

$$\delta'(B, 0) = \epsilon\text{-closure } \{\delta(q_1, q_2), 0\}$$

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

$$= \epsilon\text{-closure } \{\phi\}$$

$$= \phi$$

$$\delta'(B, 1) = \epsilon\text{-closure } \{\delta(q_1, q_2), 1\}$$

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

$$= \epsilon\text{-closure } \{q_1\}$$

$$= \{q_1, q_2\} \text{ i.e. state B itself.}$$

$$\delta'(B, 2) = \epsilon\text{-closure } \{\delta(q_1, q_2), 2\}$$

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

$$\begin{aligned}
 &= \epsilon\text{-closure } \{ q_2 \} \\
 &= \{ q_2 \} \text{ i.e. state C.}
 \end{aligned}$$

Hence

$$\delta'(B, 0) = \emptyset$$

$$\delta'(B, 1) = B$$

$$\delta'(B, 2) = C$$

The partial transition diagram will be

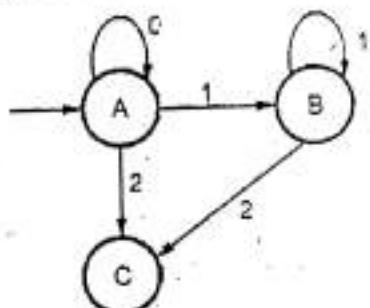


Fig. 2.10.6

Now we will obtain transitions for C :

$$\delta'(C, 0) = \epsilon\text{-closure } \{ \delta(q_2, 0) \}$$

$$= \epsilon\text{-closure } \{ \emptyset \}$$

$$= \emptyset$$

$$\delta'(C, 1) = \epsilon\text{-closure } \{ \delta(q_2, 1) \}$$

$$= \epsilon\text{-closure } \{ \emptyset \}$$

$$= \emptyset$$

$$\delta'(C, 2) = \epsilon\text{-closure } \{ \delta(q_2, 2) \}$$

$$= q_2$$

Hence the DFA is

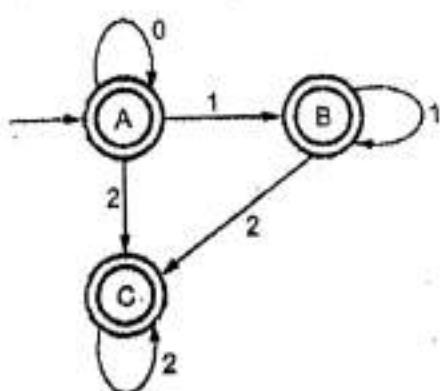


Fig. 2.10.7

As $A = \{q_0, q_1, q_2\}$ in which final state q_2 lies hence A is final state in. $B = \{q_1, q_2\}$ the state q_2 lies hence B is also final state in $C = \{q_2\}$, the state q_2 lies hence C is also a final state.

Example 2.10.5 : Convert the given NFA with ϵ to its equivalent DFA.

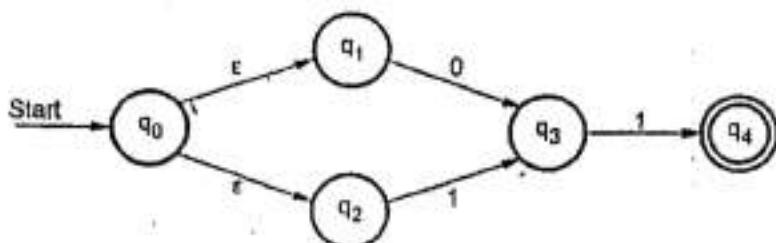


Fig. 2.10.8

Solution :

$$\epsilon\text{-closure } \{q_0\} = \{q_0, q_1, q_2\}$$

$$\epsilon\text{-closure } \{q_1\} = \{q_1\}$$

$$\epsilon\text{-closure } \{q_2\} = \{q_2\}$$

$$\epsilon\text{-closure } \{q_3\} = \{q_3\}$$

$$\epsilon\text{-closure } \{q_4\} = \{q_4\}$$

Now, Let ϵ -closure $\{q_0\} = \{q_0, q_1, q_2\}$ be state A .

$$\text{Hence } \delta'(A, 0) = \epsilon\text{-closure } \{\delta(\{q_0, q_1, q_2\}, 0)\}$$

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

$$= \epsilon\text{-closure } \{q_3\}$$

$$= \{q_3\} \text{ call it as state } B.$$

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

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

$$= \epsilon\text{-closure } \{q_3\}$$

$$= q_3 = B.$$

The partial DFA will be

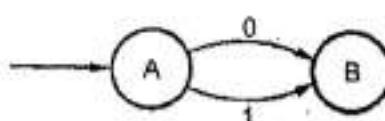


Fig. 2.10.9

Now,

$$\delta'(B, 0) = \epsilon\text{-closure} \{ \delta(q_3, 0) \}$$

$$= \emptyset$$

$$\delta'(B, 1) = \epsilon\text{-closure} \{ \delta(q_3, 1) \}$$

$$= \epsilon\text{-closure} \{ q_4 \}$$

$$= \{ q_4 \} \text{ i.e. state C}$$

$$\delta'(C, 0) = \epsilon\text{-closure} \{ \delta(q_4, 0) \}$$

$$= \emptyset$$

$$\delta'(C, 1) = \epsilon\text{-closure} \{ \delta(q_4, 1) \}$$

$$= \emptyset$$

The DFA will be,

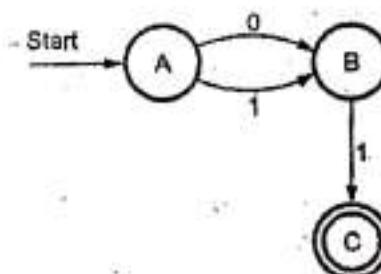


Fig. 2.10.10

→ Example 2.10.6 : Convert following NFA- Δ to NFA and FA.

q	$\delta(q, \Delta)$	$\delta(q, 0)$	$\delta(q, 1)$
A	{B}	{A}	\emptyset
B	{D}	{C}	\emptyset
C	\emptyset	\emptyset	{B}
D	\emptyset	{D}	\emptyset

GTU : Summer-15, Marks 8, Winter-17, Marks 7

Solution : We will first draw the transition graph for given table. As final state is not given we assume state D as final state.

Now we will obtain ϵ -closure for each state.

$$\epsilon\text{-closure}(A) = \{A, B, D\}$$

$$\epsilon\text{-closure}(B) = \{B, D\}$$

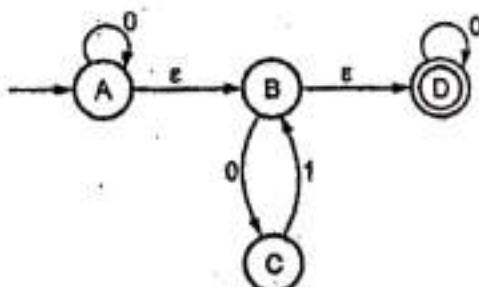


Fig. 2.10.11

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

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

Now we will obtain δ' transitions for each state and for each input symbol.

$$\begin{aligned}\delta'(A, 0) &= \epsilon\text{-closure } (\delta(\delta(A, \epsilon), 0)) \\ &= \epsilon\text{-closure } (\delta(\epsilon\text{-closure } (A), 0)) \\ &= \epsilon\text{-closure } (\delta(A, B, D), 0) \\ &= \epsilon\text{-closure } (\delta(A, 0) \cup \delta(B, 0) \cup \delta(D, 0)) \\ &= \epsilon\text{-closure } (A \cup C \cup D) \\ &= \epsilon\text{-closure } (A) \cup \epsilon\text{-closure } (C) \cup \epsilon\text{-closure } (D) \\ &= \{A, B, D\} \cup \{C\} \cup \{D\} = \{A, B, C, D\}\end{aligned}$$

$$\begin{aligned}\delta'(A, 1) &= \epsilon\text{-closure } (\delta(\delta(A, \epsilon), 1)) \\ &= \epsilon\text{-closure } (\delta(\epsilon\text{-closure } (A), 1)) \\ &= \epsilon\text{-closure } (\delta(A, B, D), 1) \\ &= \epsilon\text{-closure } (\delta(A, 1) \cup \delta(B, 1) \cup \delta(D, 1)) \\ &= \epsilon\text{-closure } (\Phi \cup \Phi \cup \Phi) \\ &= \Phi\end{aligned}$$

$$\begin{aligned}\delta'(B, 0) &= \epsilon\text{-closure } (\delta(\delta(B, \epsilon), 0)) \\ &= \epsilon\text{-closure } (\delta(\epsilon\text{-closure } (B), 0)) \\ &= \epsilon\text{-closure } (\delta(B, D), 0) \\ &= \epsilon\text{-closure } (\delta(B, 0) \cup \delta(D, 0)) \\ &= \epsilon\text{-closure } (C \cup D) \\ &= \epsilon\text{-closure } (C) \cup \epsilon\text{-closure } (D) \\ &= \{C, D\}\end{aligned}$$

$$\begin{aligned}\delta'(B, 1) &= \epsilon\text{-closure } (\delta(\delta(B, \epsilon), 1)) \\ &= \epsilon\text{-closure } (\delta(\epsilon\text{-closure } (B), 1)) \\ &= \epsilon\text{-closure } (\delta(B, D), 1) \\ &= \epsilon\text{-closure } (\delta(B, 1) \cup \delta(D, 1)) \\ &= \epsilon\text{-closure } (\Phi \cup \Phi) \\ &= \Phi\end{aligned}$$

$$\begin{aligned}
 \delta'(C, 0) &= \text{e-closure}(\delta(\delta(C, \epsilon), 0)) \\
 &= \text{e-closure}(\delta(\text{e-closure}(C), 0)) \\
 &= \text{e-closure}(\delta(C, 0)) \\
 &= \text{e-closure}(\Phi) \\
 &= \Phi \\
 \delta'(C, 1) &= \text{e-closure}(\delta(\delta(C, \epsilon), 1)) \\
 &= \text{e-closure}(\delta(\text{e-closure}(C), 1)) \\
 &= \text{e-closure}(\delta(C, 1)) \\
 &= \text{e-closure}(B) \\
 &= \{B, D\} \\
 \delta'(D, 0) &= \text{e-closure}(\delta(\delta(D, \epsilon), 0)) \\
 &= \text{e-closure}(\delta(D, 0)) \\
 &= \text{e-closure}(D) \\
 &= \{D\} \\
 \delta'(D, 1) &= \text{e-closure}(\delta(\delta(D, \epsilon), 1)) \\
 &= \text{e-closure}(\delta(\text{e-closure}(D), 1)) \\
 &= \text{e-closure}(\delta(D, 1)) \\
 &= \text{e-closure}(\Phi) \\
 &= \Phi
 \end{aligned}$$

Input States \	0	1
{A}	{A, B, C, D}	Φ
{B}	{C, D}	Φ
{C}	Φ	{B, D}
{D}	{D}	Φ

Fig. 2.10.12

The transition table for this NFA will be -

The above NFA can be converted to DFA. It is as follows -

$$\delta([A], 0) = [A, B, C, D]. \text{ We assume it as state } [ABCD]$$

$$\delta([B], 0) = [C, D]. \text{ We consider it as state } [CD]$$

$$\delta([C], 1) = [B, D]. \text{ We consider it as state } [BD]$$

As new states [ABCD] [CD] and [BD] are generated, we will obtain input transitions on it.

$$\delta([ABCD], 0) = [ABCD]$$

$$\delta([ABCD], 1) = [BD]$$

$$\delta([CD], 0) = [D]$$

$$\delta([CD], 1) = [BD]$$

$$\delta([BD], 0) = [CD]$$

$$\delta([BD], 1) = \Phi$$

No more new states are generated.

The transition table will be

Input State State \ Input	0	1
[A]	[ABCD]	Φ
[B]	[CD]	Φ
[C]	Φ	[BD]
[D]	[D]	Φ
[ABCD]	[ABCD]	[BD]
[CD]	[D]	[BD]
[BD]	[CD]	Φ

Fig. 2.10.13

The FA will be drawn from above table.

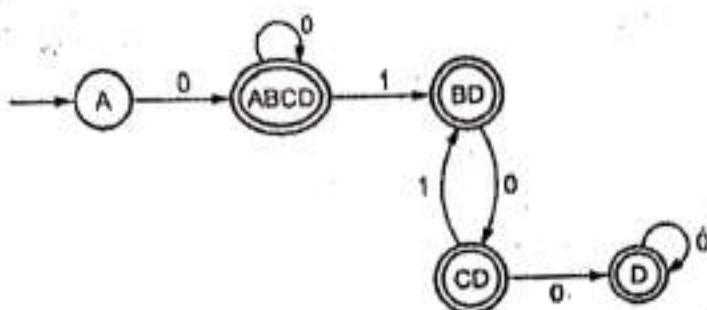


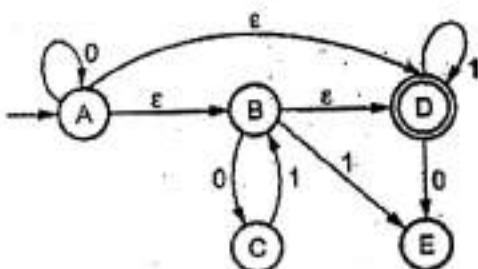
Fig. 2.10.14

→ Example 2.10.7 : Convert following NFA - Λ to NFA and FA.

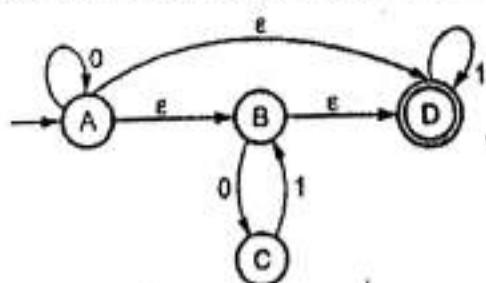
GTU : Winter-12, Marks 8

Q	$\delta(q, \Lambda)$	$\delta(q, 0)$	$\delta(q, 1)$
A	{B, D}	{A}	\emptyset
B	\emptyset	{C}	{E}
C	\emptyset	\emptyset	{B}
D	\emptyset	{E}	{D}
E	\emptyset	\emptyset	\emptyset

Solution : We will first draw the transition graph for given table. We will assume E as a final state.



But since E is a dead state, we will eliminate it. Then the FA will become



For remaining part of solution refer similar example on page 2-46.

→ Example 2.10.8 : Convert the following NFA- Λ into FA. GTU : Summer-14, Marks 7

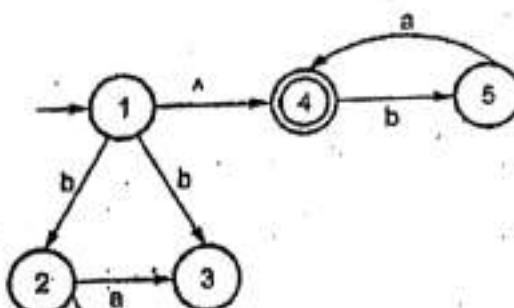


Fig. 2.10.15

Solution : Let us obtain ϵ -closure of each state.

$$\epsilon\text{-closure } (1) = \{1, 4\} \text{ call it state A}$$

$$\epsilon\text{-closure } (2) = \{2\} \text{ call it state B}$$

$$\epsilon\text{-closure } (3) = \{3\} \text{ call it state C}$$

$$\epsilon\text{-closure } (4) = \{4\} \text{ call it state D}$$

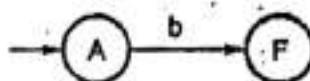
$$\epsilon\text{-closure } (5) = \{5\} \text{ call it state E}$$

Now, we will obtain δ transitions for each state and for each input symbol.

$$\begin{aligned}\delta(A, a) &= \epsilon\text{-closure}\{\delta(1, 4), a\} \\ &= \epsilon\text{-closure}\{\delta(1, a) \cup \delta(4, a)\} \\ &= \epsilon\text{-closure}\{\emptyset\} = \emptyset\end{aligned}$$

$$\begin{aligned}\delta(A, b) &= \epsilon\text{-closure}\{\delta(1, 4), b\} \\ &= \epsilon\text{-closure}\{\delta(1, b) \cup \delta(4, b)\} \\ &= \epsilon\text{-closure}\{\delta(2, 3) \cup 5\} \\ &= \{2, 3, 5\} \text{ call it as state F.}\end{aligned}$$

Partial DFA is



$$\begin{aligned}\delta(B, a) &= \epsilon\text{-closure}\{\delta(2, a)\} \\ &= \epsilon\text{-closure}\{3\} \\ &= \{3\} \text{ i.e. C}\end{aligned}$$

$$\delta(B, b) = \epsilon\text{-closure}\{\delta(2, b)\}$$

$$= \emptyset$$

$$\delta(C, a) = \text{e-closure } \{\delta(3, a)\}$$

$$\delta(C, a) = \emptyset$$

$$\delta(C, b) = \text{e-closure } \{\delta(3, b)\}$$

$$= \emptyset$$

$$\delta(D, a) = \text{e-closure } \{\delta(4, a)\}$$

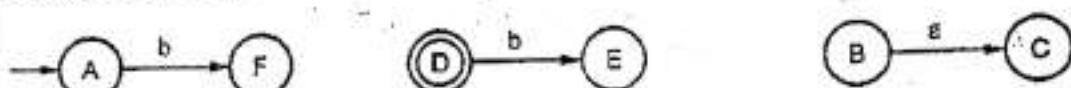
$$= \text{e-closure } \{\emptyset\}$$

$$= \emptyset$$

$$\delta(D, b) = \text{e-closure } \{\delta(4, b)\}$$

$$= \{5\} \text{ i.e. E}$$

Partial DFA will be



$$\delta(E, a) = \text{e-closure } \{\delta(5, a)\}$$

$$= \text{e-closure } \{4\}$$

$$= \{4\} \text{ i.e. D}$$

$$\delta(E, b) = \text{e-closure } \{\delta(5, b)\}$$

$$= \emptyset$$

$$\delta(F, a) = \text{e-closure } \{\delta(2, 3, 5), a\}$$

$$= \text{e-closure } \{\delta(2, a) \cup \delta(3, a) \cup \delta(5, a)\}$$

$$= \text{e-closure } \{3, 4\}$$

$$= \{3, 4\} \text{ i.e. G.}$$

$$\delta(F, b) = \text{e-closure } \{\delta(2, 3, 5), b\}$$

$$= \text{e-closure } \{\delta(2, b) \cup \delta(3, b) \cup \delta(5, b)\}$$

$$= \text{e-closure } \{\emptyset \cup \emptyset \cup \emptyset\}$$

$$= \emptyset$$

$$\delta(G, a) = \text{e-closure } \{\delta(3, 4), a\}$$

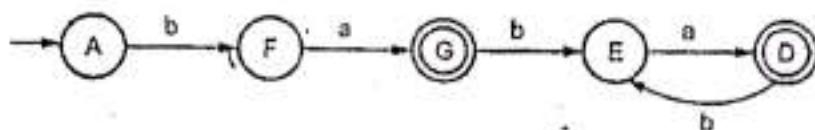
$$= \text{e-closure } \{\delta(3, a) \cup \delta(4, a)\}$$

$$= \text{e-closure } \{\emptyset \cup \emptyset\}$$

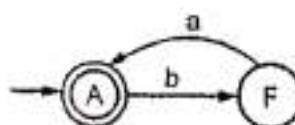
$$= \emptyset$$

$$\begin{aligned}
 \delta(G, b) &= \epsilon\text{-closure } \{\delta(3, 4), b\} \\
 &= \epsilon\text{-closure } \{\delta(3, b) \cup \delta(4, b)\} \\
 &= \epsilon\text{-closure } \{\emptyset \cup 5\} \\
 &= \epsilon\text{-closure } \{5\} \\
 &= \{5\} \text{ i.e. E}
 \end{aligned}$$

The DFA will be



The minimized DFA will be



→ Example 2.10.9 : Consider the NFA- Δ depicted in following table :

	Δ	a	b	c
$\rightarrow p$	Φ	{p}	{q}	{r}
q	{p}	{q}	{r}	Φ
r	{q}	{r}	Φ	{p}

i) Compute the Δ -closure of each state.

ii) Convert the NFA- Δ to a DFA.

GTU : Summer-17, Marks 7

Solution : We will draw transition graph for given NFA with ϵ .

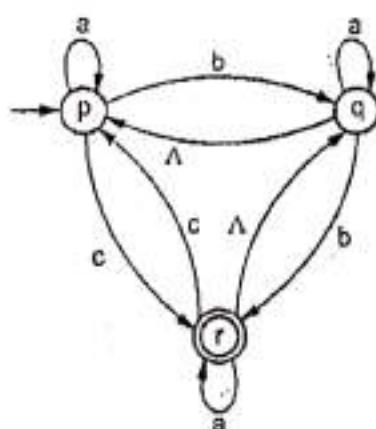


Fig. 2.10.16

i) ϵ -closure of each state

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

$$\epsilon\text{-closure}(q) = \{q, p\}$$

$$\epsilon\text{-closure}(r) = \{p, q, r\}$$

ii) Conversion to DFA

Let, $\epsilon\text{-closure}(p) = \{p\}$ call it as state A

Now let us apply transitions on state A

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

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

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

$$= \{p\}$$

$$\delta'(A, a) = A$$

$$\delta'(A, b) = \epsilon\text{-closure}(\delta(A, b))$$

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

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

$$= \{p, q\} \text{ call it as state B}$$

$$\delta'(A, c) = \epsilon\text{-closure}(\delta(A, c))$$

$$= \epsilon\text{-closure}(\delta(p, c))$$

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

$$= \{p, q, r\} \text{ call it as state C}$$

$$\delta'(A, c) = C$$

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

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

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

$$= \epsilon\text{-closure}(\{p\} \cup \{q\})$$

$$= \{p, q\}$$

$$\delta'(B, a) = B$$

$$\delta'(B, b) = \epsilon\text{-closure}(\delta(B, b))$$

$$= \epsilon\text{-closure}(\delta(p, q), b)$$

$$\begin{aligned}
 &= \epsilon\text{-closure}(q, r) \\
 &= \epsilon\text{-closure}(q) \cup \epsilon\text{-closure}(r) \\
 &= \{p, q, r\}
 \end{aligned}$$

$$\delta'(B, b) = C$$

$$\begin{aligned}
 \delta'(B, c) &= \epsilon\text{-closure}(\delta(B, c)) \\
 &= \epsilon\text{-closure}(\delta(p, q), c) \\
 &= \epsilon\text{-closure}(\delta(p, c) \cup \delta(q, c)) \\
 &= \epsilon\text{-closure}(r) \\
 &= \{p, q, r\}
 \end{aligned}$$

$$\delta'(B, a) = C$$

$$\begin{aligned}
 \delta'(C, a) &= \epsilon\text{-closure}(\delta(C, a)) \\
 &= \epsilon\text{-closure}(\delta(p, q, r), a) \\
 &= \epsilon\text{-closure}(\delta(p, a) \cup \delta(q, a) \cup \delta(r, a)) \\
 &= \epsilon\text{-closure}\{p, q, r\} \\
 &= \{p, q, r\}
 \end{aligned}$$

$$\delta'(C, b) = C$$

$$\begin{aligned}
 \delta'(C, b) &= \epsilon\text{-closure}(\delta(C, b)) \\
 &= \epsilon\text{-closure}(\delta\{p, q, r\}, b) \\
 &= \epsilon\text{-closure}(\delta(p, b) \cup \delta(q, b) \cup \delta(r, b)) \\
 &= \epsilon\text{-closure}(q \cup r) \\
 &= \{p, q, r\}
 \end{aligned}$$

$$\delta'(C, c) = C$$

$$\begin{aligned}
 \delta'(C, c) &= \epsilon\text{-closure}(\delta(C, c)) \\
 &= \epsilon\text{-closure}(\delta(p, q, r), c) \\
 &= \epsilon\text{-closure}(\delta(p, c) \cup \delta(q, c) \cup \delta(r, c))
 \end{aligned}$$

$$\delta'(C, c) = \{p, r\} \text{ call it as D}$$

$$\begin{aligned}
 \delta'(D, a) &= \epsilon\text{-closure}(\delta(D, a)) \\
 &= \epsilon\text{-closure}(\delta\{p, r\}, a))
 \end{aligned}$$

$$\begin{aligned}
 &= \epsilon\text{-closure}(\delta(p, a) \cup \delta(r, a)) \\
 &= \epsilon\text{-closure}(p \cup r) \\
 &= \epsilon\text{-closure}(p) \cup \epsilon\text{-closure}(r) \\
 &= \{p\} \cup \{p, q, r\} \\
 &= \{p, q, r\}
 \end{aligned}$$

$$\therefore \delta'(D, a) = C$$

$$\begin{aligned}
 \delta'(D, b) &= \epsilon\text{-closure}(\delta(D, b)) \\
 &= \epsilon\text{-closure}(\delta(p, r), b) \\
 &= \epsilon\text{-closure}(\delta(p, b) \cup \delta(r, b)) \\
 &= \epsilon\text{-closure}(q \cup r) \\
 &= \{p, q, r\}
 \end{aligned}$$

$$\therefore \delta'(D, b) = C$$

$$\begin{aligned}
 \delta'(D, c) &= \epsilon\text{-closure}(\delta(D, c)) \\
 &= \epsilon\text{-closure}(\delta(p, r), c) \\
 &= \epsilon\text{-closure}(\delta(p, c) \cup \delta(r, c)) \\
 &= \epsilon\text{-closure}(r \cup p) \\
 &= \{p, q, r\}
 \end{aligned}$$

$$\therefore \delta'(D, c) = C$$

→ Example 2.10.10 : Fig. 2.10.18 shows NFA-^A. Draw an FA accepting the same language.

GTU : Summer-18, Marks 7

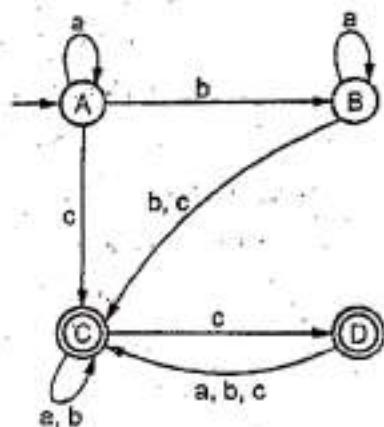


Fig. 2.10.17

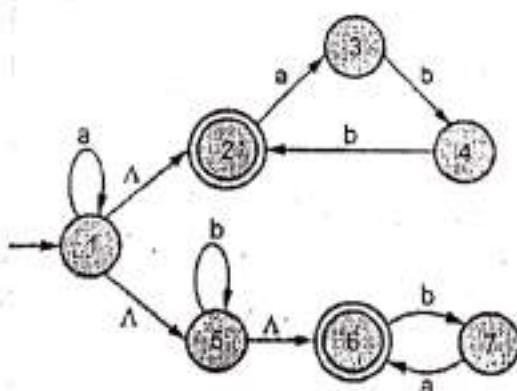


Fig. 2.10.18

Solution :

$$\epsilon\text{-closure (1)} = \{1, 2, 5, 6\}$$

$$\epsilon\text{-closure (2)} = \{2\} = \{2\}$$

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

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

$$\epsilon\text{-closure (5)} = \{5\} = \{5, 6\}$$

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

$$\epsilon\text{-closure (7)} = \{7\} = \{7\}$$

Now,

$$\epsilon\text{-closure (1)} = \{1, 2, 5, 6\} \text{ be state A}$$

$$\delta'(A, a) = \epsilon\text{-closure } \{\delta(1, 2, 5, 6), a\}$$

$$= \epsilon\text{-closure } \{(1) \cup (3)\}$$

$$= \epsilon\text{-closure (1)} \cup \epsilon\text{-closure (3)}$$

$$= \{1, 2, 3, 5, 6\} \rightarrow \text{call it as B}$$

$$\therefore \delta'(A, a) = B$$

$$\delta'(A, b) = \epsilon\text{-closure } \{\delta(1, 2, 5, 6), b\}$$

$$= \epsilon\text{-closure } \{(5) \cup (7)\}$$

$$= \epsilon\text{-closure (5)} \cup \epsilon\text{-closure (7)}$$

$$= \{5, 6, 7\} \rightarrow \text{call it as state C}$$

$$\therefore \delta'(A, b) = C$$

$$\delta'(B, a) = \epsilon\text{-closure } \{\delta(1, 2, 3, 5, 6), a\}$$

$$= \epsilon\text{-closure } \{(1) \cup (3)\}$$

$$\delta'(B, a) = B$$

$$\delta'(B, b) = \epsilon\text{-closure } \{\delta(1, 2, 3, 5, 6), b\}$$

$$= \epsilon\text{-closure } \{(4) \cup (7) \cup (5)\}$$

$$= \epsilon\text{-closure (4)} \cup \epsilon\text{-closure (5)} \cup \epsilon\text{-closure (7)}$$

$$= \{4, 5, 6, 7\} \rightarrow \text{call it as D}$$

$$\delta'(B, b) = D$$

$$\delta'(C, a) = \epsilon\text{-closure } \{\delta(5, 6, 7), a\}$$

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

$$\delta'(C, a) = \{6\} \rightarrow \text{call it as state E}$$

$$\delta'(C, a) = E$$

$$\begin{aligned}\delta'(C, b) &= \epsilon\text{-closure } \{\delta(5, 6, 7), b\} \\ &= \epsilon\text{-closure } \{(5) \cup (7)\}\end{aligned}$$

$$\delta'(C, b) = C$$

$$\begin{aligned}\delta'(D, a) &= \epsilon\text{-closure } \{\delta(4, 5, 6, 7), a\} \\ &= \epsilon\text{-closure } \{6\}\end{aligned}$$

$$\delta'(D, a) = E$$

$$\begin{aligned}\delta'(D, b) &= \epsilon\text{-closure } \{\delta(4, 5, 6, 7), b\} \\ &= \epsilon\text{-closure } \{(2) \cup (5) \cup (7)\} \\ &= \{2, 5, 6, 7\} \rightarrow \text{call it as F}\end{aligned}$$

$$\delta'(D, b) = F$$

$$\begin{aligned}\delta'(E, a) &= \epsilon\text{-closure } \{\delta(6), a\} \\ &= \epsilon\text{-closure } \{\phi\} \\ &= \phi\end{aligned}$$

$$\begin{aligned}\delta'(E, b) &= \epsilon\text{-closure } \{\delta(6), b\} \\ &= \{7\} \rightarrow \text{call it as G}\end{aligned}$$

$$\delta'(E, b) = G$$

$$\begin{aligned}\delta'(F, a) &= \epsilon\text{-closure } \{\delta(2, 5, 6, 7), a\} \\ &= \epsilon\text{-closure } \{(3) \cup (6)\} \\ &= \epsilon\text{-closure } \{3\} \cup \epsilon\text{-closure } \{6\} \\ &= \{3, 6\} \rightarrow \text{call it as H}\end{aligned}$$

$$\delta'(F, a) = H$$

$$\begin{aligned}\delta'(F, b) &= \epsilon\text{-closure } \{\delta(2, 5, 6, 7), b\} \\ &= \epsilon\text{-closure } \{(5) \cup (7)\}\end{aligned}$$

$$\delta'(F, b) = C$$

$$\begin{aligned}\delta'(G, a) &= \epsilon\text{-closure } \{\delta(7), a\} \\ &= \epsilon\text{-closure } \{6\}\end{aligned}$$

$$\delta'(G, a) = E$$

$$\delta'(G, b) = \epsilon\text{-closure}(\delta(7), b)$$

$$= \phi$$

$$\delta'(H, a) = \epsilon\text{-closure}(\delta(3, 6), a)$$

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

$$\delta'(H, a) = \phi$$

$$\delta'(H, b) = \epsilon\text{-closure}(\delta(3, 6), b)$$

$$= \epsilon\text{-closure}(3, b) \cup \epsilon\text{-closure}(6, b)$$

$$= \{4, 7\} \rightarrow \text{call it as I}$$

$$\delta'(H, b) = I$$

$$\delta'(I, a) = \epsilon\text{-closure}(\delta(4, 7), a)$$

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

$$\delta'(I, a) = E$$

$$\delta'(I, b) = \epsilon\text{-closure}(\delta(4, 7), b)$$

$$= \epsilon\text{-closure}\{2\} \cup \epsilon\text{-closure}\{\phi\}$$

$$= \{2\} \rightarrow \text{call it as J}$$

$$\delta'(J, a) = \epsilon\text{-closure}(\delta(2, a))$$

$$= \{3\} \rightarrow \text{call it as K}$$

$$\delta'(J, b) = \epsilon\text{-closure}(\delta(2, b))$$

$$\delta'(J, b) = \phi$$

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

$$\delta'(K, a) = \phi$$

$$\delta'(K, b) = \epsilon\text{-closure}(\delta(3, b))$$

$$\delta'(K, b) = \{4\} \rightarrow \text{call it as L}$$

$$\delta'(L, a) = \epsilon\text{-closure}(\delta(4, a))$$

$$\delta'(L, a) = \phi$$

$$\delta'(L, b) = \epsilon\text{-closure}(\delta(4, b))$$

$$= \epsilon\text{-closure}\{2\}$$

$$= \{2\}$$

$$\delta'(L, b) = J$$

As no new state getting generated DFA is

State \ Input	a	b
State		
A	B	C
B	B	D
C	E	C
D	E	F
E	∅	G
F	H	∅
G	E	∅
H	∅	I
I	∅	J
J	K	∅
K	∅	I
L	∅	J

Here I = L Hence remove state L by replacing it by I

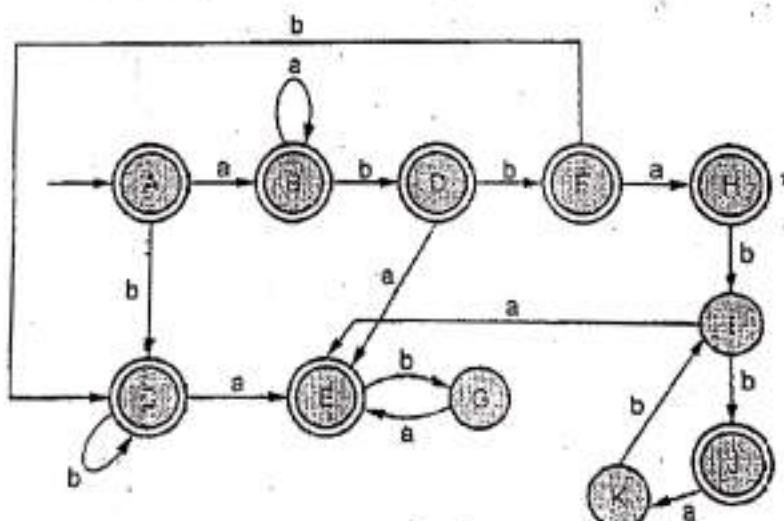


Fig. 2.10.19

Review Question

- Define NFA - Λ . Explain how to convert NFA - Λ into NFA and FA with suitable example.

GTU : Summer-16, Marks 7

2.11 Conversion of Regular Expression to FA (Kleen Theorem)

Theorem 1 : Let r be a regular expression, then there exists a NFA with ϵ transitions that accepts $L(r)$.

Proof : This theorem can be proved by induction method.

The basis of induction will be by considering r has zero operators.

Basis (zero operators) - Now, since r has zero operators, means r can be either ϵ or ϕ or a for some a in input set Σ .

The finite automata for the same can be written as

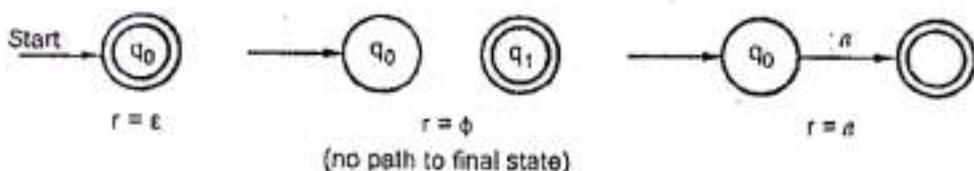


Fig. 2.11.1 Finite automata for given regular expression

Induction : This theorem can be true for n number of operators. The n is greater than or equal to 1. The regular expression contains equal to or more than one operators.

In any type of regular expression there are only three cases possible.

1. Union
2. Concatenation
3. Closure.

Let us see each,

Case 1 : Union case

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

There exists two NFA's $M_1 = (Q_1, \Sigma_1, \delta_1, \{f_1\})$

and $M_2 = (Q_2, \Sigma_2, \delta_2, \{f_2\})$

$L(M_1) = L(r_1)$ means the language states by regular expression r_1 is same which is represented by M_1 . Similarly $L(M_2) = L(r_2)$.

Q_1 represents the set of all the states in machine M_1 .

Q_2 represents the set of all the states in machine M_2 .

We assume that Q_1 and Q_2 are totally different i.e. Q_1 and Q_2 are disjoint.

Let q_0 be new initial state and f_0 be the new final state we will form

$$M = ((Q_1 \cup Q_2 \cup \{q_0, f_0\}), (\Sigma_1 \cup \Sigma_2), \delta, q_0, \{f_0\})$$

The δ is denoted by,

i) $\delta(q_0, \epsilon) = \{q_1, q_2\}$.

ii) $\delta(q, a) = \delta_1(q, a)$ for q in $Q_1 - \{f_1\}$ and a in $\Sigma_1 \cup \{\epsilon\}$.

- iii) $\delta(q, a) = \delta_2(q, a)$ for q in $Q_2 - \{f_2\}$ and a in $\Sigma_2 \cup \{\epsilon\}$
- iv) $\delta(f_1, \epsilon) = \delta_1(f_2, \epsilon) = \{f_0\}$.

All the moves are now present in machine M which is as shown Fig. 2.11.2.

The construction of machine M is shown by the transition from q_0 to f_0 . It must begin by going to q_1 or q_2 on ϵ . If the path goes to q_1 , then it follows the path in machine M_1 and goes to the state f_1 and then to f_0 on ϵ . Similarly, if the path goes to q_2 , then it follows the path in machine M_2 and goes to state f_2 and then to f_0 on ϵ . Thus the $L(M) = L(M_1) \cup L(M_2)$. That means either the path in machine M_1 or M_2 will be followed. This defines $L(M)$ to be union of $L(M_1)$ and $L(M_2)$.

Case 2 : Concatenation case

Consider that there are two regular expressions r_1 and r_2 such that $r = r_1 r_2$. The M_1 and M_2 denotes the two machines such that $L(M_1) = L(r_1)$ and $L(M_2) = L(r_2)$.

The construction of machine M will be

$$M = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, \{q_1\}, \{f_2\})$$

The mapping function δ will be given as

- i) $\delta(q, a) = \delta_1(q, a)$ for q in $Q_1 - \{f_1\}$ and a in $\Sigma_1 \cup \{\epsilon\}$
- ii) $\delta(f_1, \epsilon) = \{q_2\}$
- iii) $\delta(q, a) = \delta_2(q, a)$ for q in Q_2 and a in $\Sigma_2 \cup \{\epsilon\}$

The machine M is shown in the

Fig. 2.11.3

The initial state is q_1 by some input a the next state will be f_1 . And on receiving ϵ the transition will be from f_1 to q_2 and the final state will be f_2 . The transition from q_2 to f_2 will be on receiving some input b .

$$\text{Thus } L(M) = ab$$

That is a is in $L(M_1)$ and b is in $L(M_2)$.

Hence we can prove $L(M) = L(M_1)L(M_2)$.

Case 3 : Closure case

Let $r = r_1^*$ where r_1 be a regular expression.

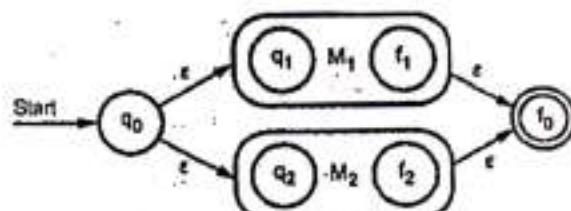


Fig. 2.11.2 The machine M for union

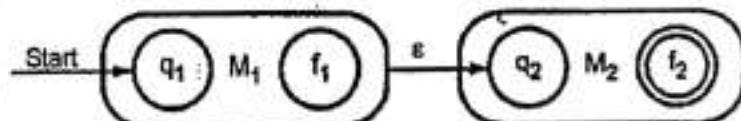


Fig. 2.11.3 Machine M for concatenation

The machine M_1 is such that $L(M_1) = L(r_1)$.

Then construct $M = (Q_1 \cup \{q_0, f_0\}, \Sigma_1, \delta, q_0, \{f_0\})$

The mapping function δ is given by,

$$\text{i)} \delta(q_0, \epsilon) = \delta(f_1, \epsilon) = \{q_1, f_0\}$$

$$\text{ii)} \delta(q, a) = \delta_1(q, a) \text{ for } q \in Q_1 - \{f_1\} \text{ and } a \in \Sigma_1 \cup \{\epsilon\}$$

The machine M will be shown

in Fig. 2.11.4.

The machine M_1 shows that from q_0 to q_1 there is a transition on receiving ϵ similarly, from q_0 to f_0 on ϵ there is a path. The path exists from f_1 to q_1 , a back path. Similarly a transition from f_1 to f_0 final state, on receiving ϵ . The total recursion is possible. Thus one can derive $\epsilon, a, aa, aaa, \dots$ for the input a .

Thus $L(M) = L(M_1)^*$ is proved.

Now based on this proof let us solve some examples. These examples illustrate how to convert given regular expression to NFA with ϵ moves.

Example 2.11.1 : Construct NFA for the regular expression $b + ba^*$.

Solution : The regular expression

$r = b + ba^*$ can be broken into r_1 and r_2 as

$$r_1 = b$$

$$r_2 = ba^*$$

Let us draw the NFA for r_1 , which is very simple.

Now, we will go for $r_2 = ba^*$, this can be broken into r_3 and r_4 where $r_3 = b$ and $r_4 = a^*$. Now the case for concatenation will be applied. The NFA will look like this r_3 will be shown in Fig. 2.11.6.

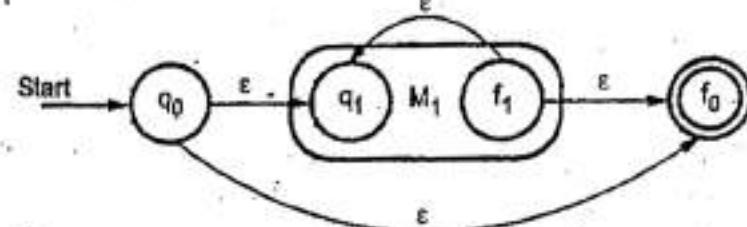


Fig. 2.11.4

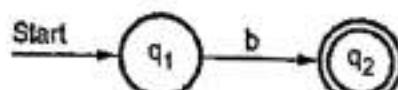


Fig. 2.11.5 For r_1

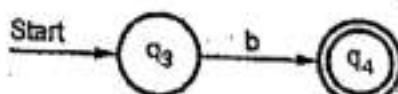


Fig. 2.11.6 or r_3

And r_4 will be shown as

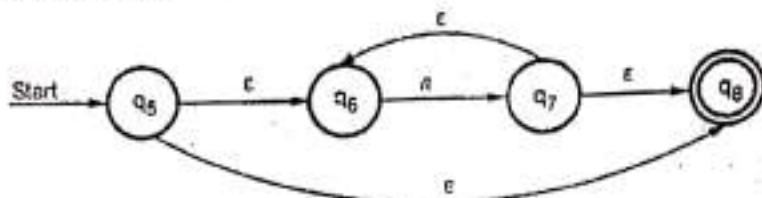


Fig. 2.11.7 For r_4

The r_2 will be $r_2 = r_3 \cdot r_4$

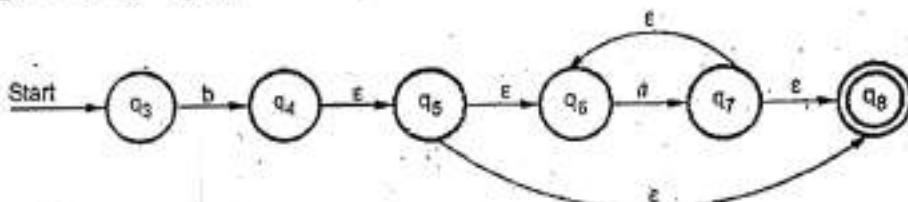


Fig. 2.11.7 (a) For r_2

Now, we will draw NFA for $r = r_1 + r_2$ i.e. $b + ba^*$.

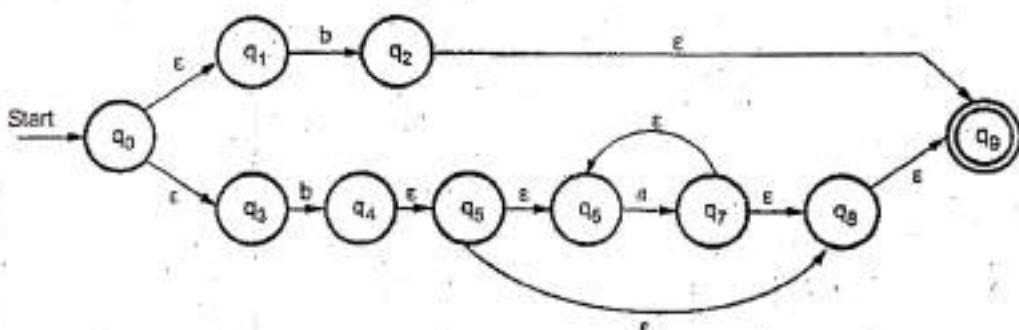


Fig. 2.11.8 For example 2.11.1

► Example 2.11.2 : Construct NFA with ϵ moves for the regular expression $(0+1)^*$.

Solution : The NFA will be constructed step by step by breaking regular expression into small regular expressions.

$$\begin{aligned} r_3 &= (r_1 + r_2) \\ r &= r_3^* \end{aligned}$$

where $r_1 = 0$, $r_2 = 1$

NFA for r_1 will be

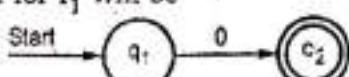


Fig. 2.11.9

NFA for r_2 will be

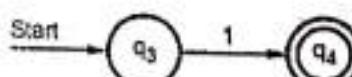


Fig. 2.11.10

NFA for r_3 will be

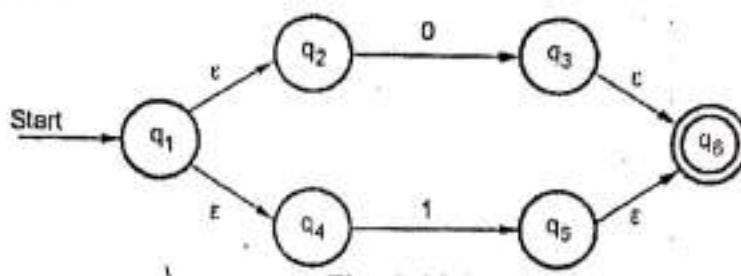


Fig. 2.11.11

And finally

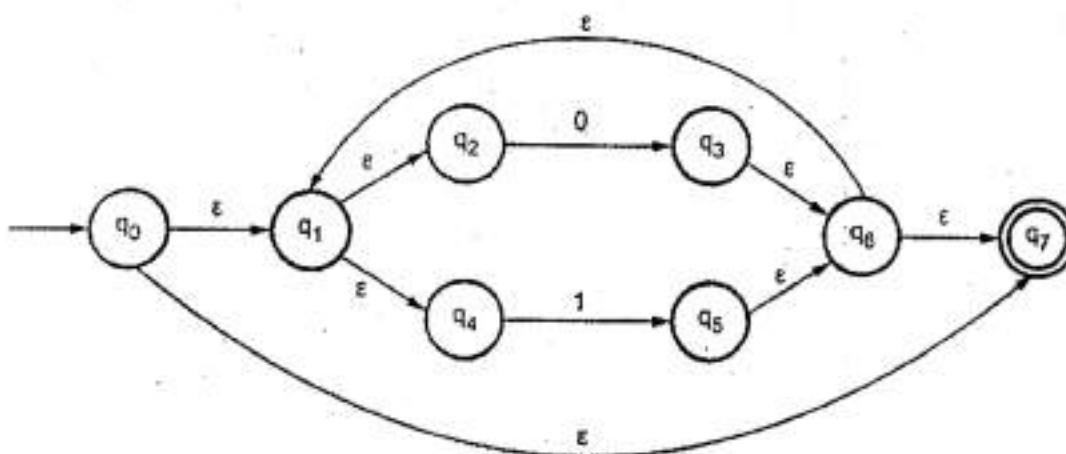


Fig. 2.11.12 For example 2.11.2

2.11.1 Direct Method for Conversion of FA to RE

This method is a direct method for obtaining FA from given regular expression. This is called a subset method. The method is given as below -

Step 1 : Design a transition diagram for given regular expression, using NFA with ϵ moves.

Step 2 : Convert this NFA with ϵ to NFA without ϵ .

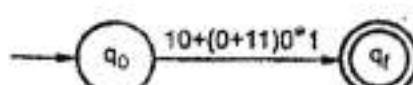
Step 3 : Convert the obtained NFA to equivalent DFA.

Let us understand this method with the help of some example.

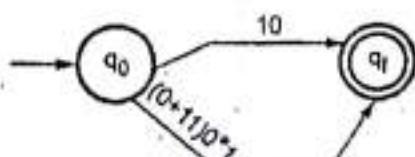
→ **Example 2.11.3 :** Design a FA from given regular expression $10 + (0 + 11)0^*1$.

Solution : First we will construct the transition diagram for given regular expression.

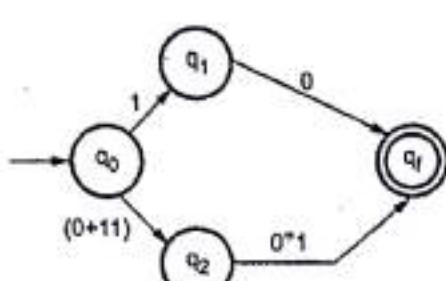
Step 1 :



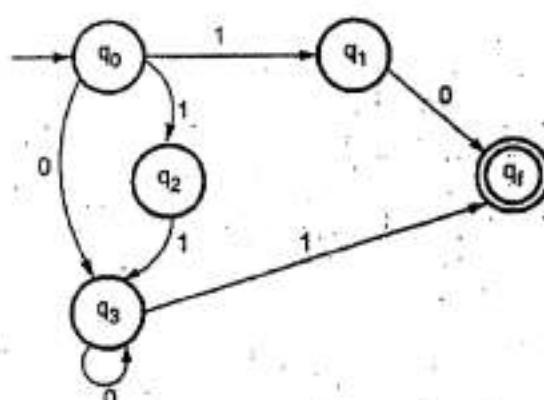
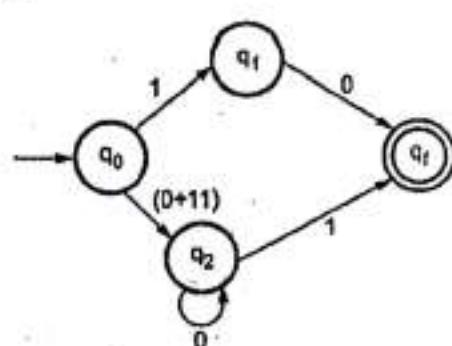
Step 2 :



Step 3 :



Step 4 :



Now we have got NFA without ϵ . Now we will convert it to required DFA for that, we will first write a transition table for this NFA.

State \ Input	0	1
State		
q_0	q_3	$\{q_1, q_2\}$
q_1	q_f	\emptyset
q_2	\emptyset	q_3
q_3	q_3	q_f
q_f	\emptyset	\emptyset

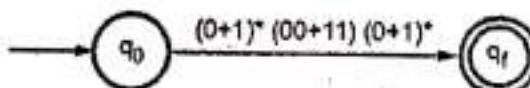
The equivalent DFA will be

State \ Input	0	1
State		
$[q_0]$	$[q_3]$	$[q_1, q_2]$
$[q_1]$	$[q_f]$	\emptyset
$[q_2]$	\emptyset	$[q_3]$
$[q_3]$	$[q_3]$	$[q_f]$
$[q_1, q_2]$	$[q_f]$	$[q_3]$
(q_f)	\emptyset	\emptyset

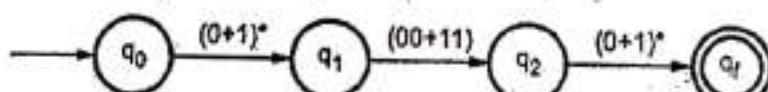
Example 2.11.4 : Construct finite automaton to accept the regular expression $(0 + 1)^* (00 + 11)(0 + 1)^*$.

Solution : The FA for r.e. = $(0 + 1)^* (00 + 11)(0 + 1)^*$ as follows -

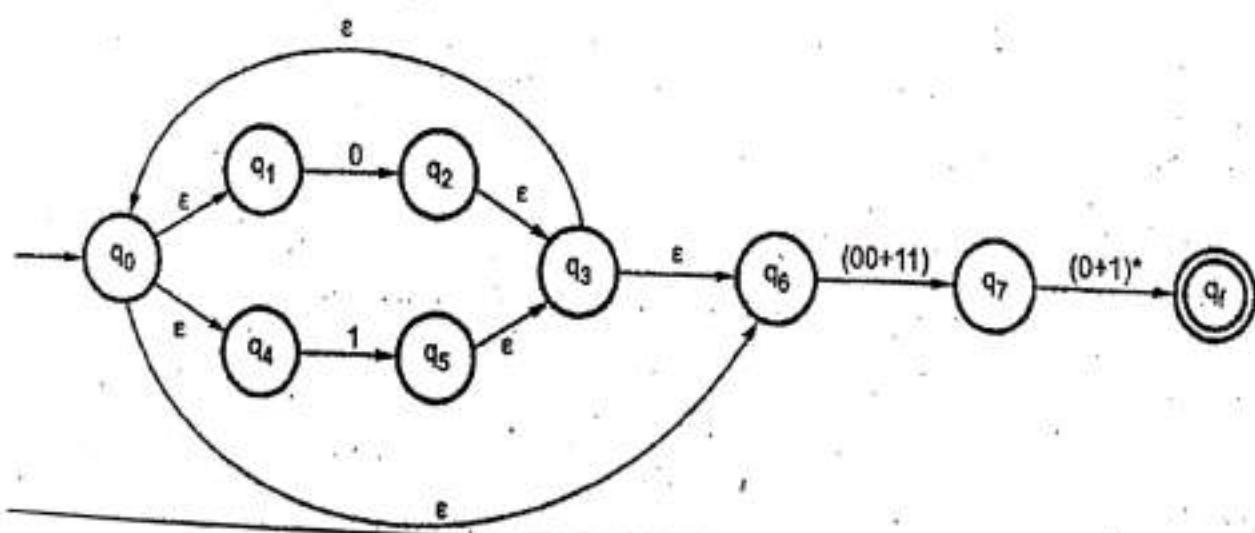
Step 1 :

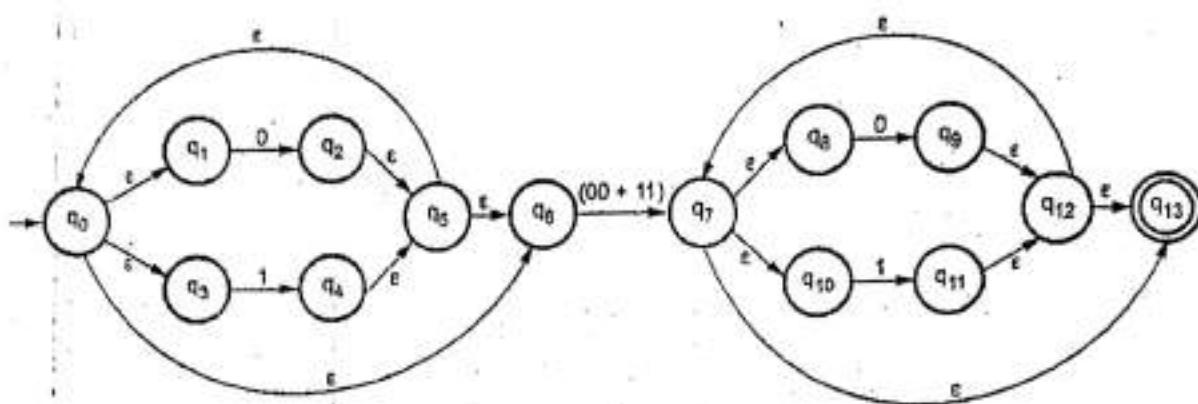


Step 2 :

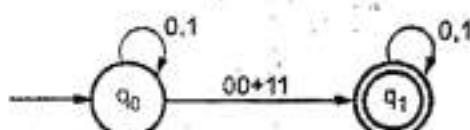
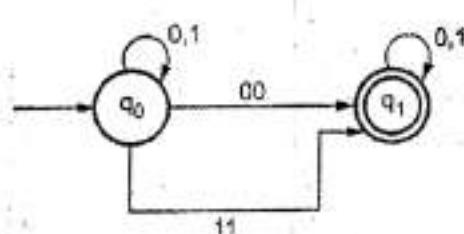
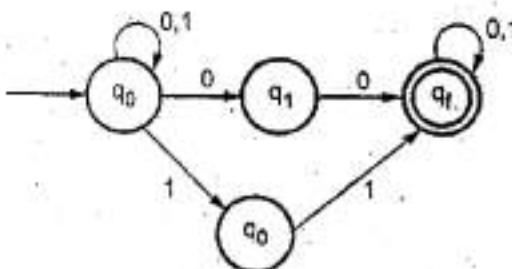


Step 3 :



Step 4 :**Step 5 :**

We will remove ϵ -transitions.

**Step 6 :****Step 7 :**

is the required FA.

→ **Example 2.11.5 :** For the following regular expression draw an NFA & recognizing the corresponding languages.

- i) $(00 + 1)^*$ ii) 001^*0^*11

GTU : Summer-11, Marks 6

Solution : i) $L = \{ \text{Every occurrence of } 10 \text{ is preceded by even number of } 0's \text{ and any number of } 1's \}$.

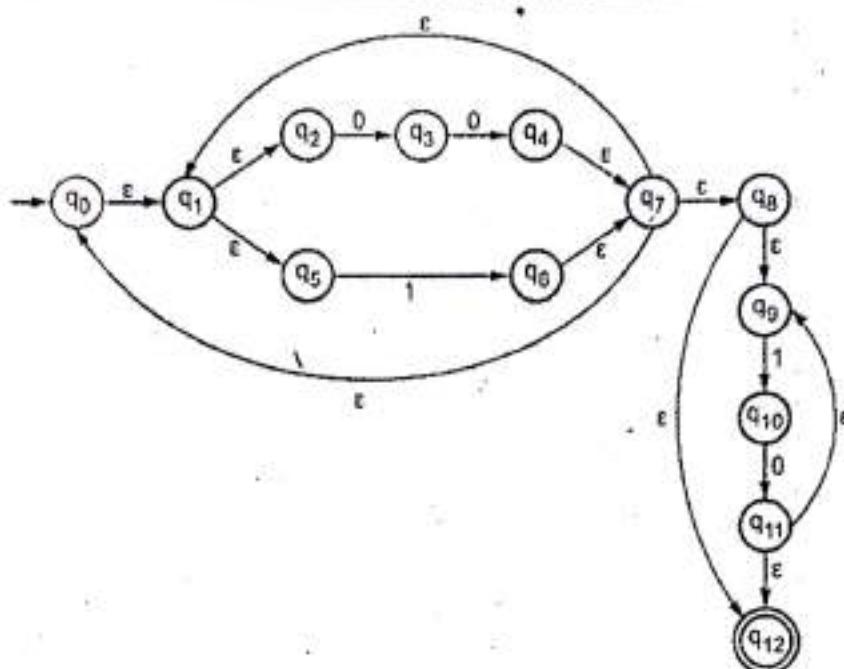


Fig. 2.11.13

ii) $L = \{ \text{The strings begin with } 00 \text{ and end with } 11, \text{ in between any number of } 1\text{'s are followed by any number of } 0\text{'s}\}$.

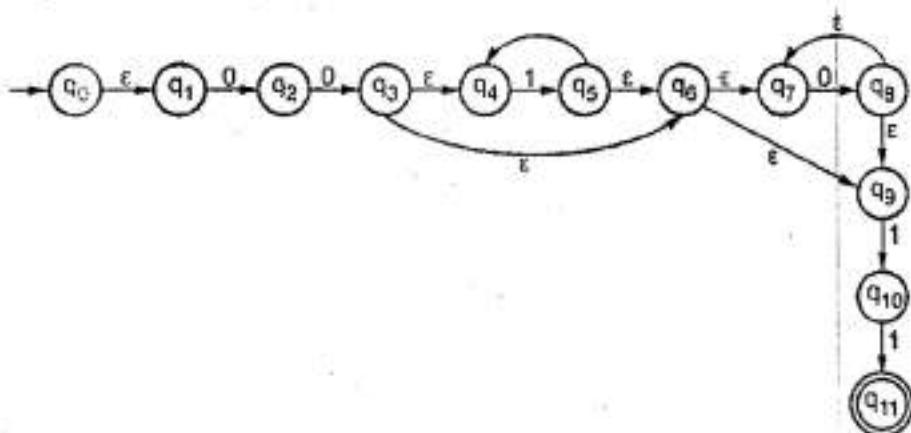


Fig. 2.11.14

→ **Example 2.11.6 :** Compare FA, NFA and NFA - Λ .

For the following regular expression draw an NFA- Λ recognizing the corresponding language.

$$(0 + 1)^* (10 + 110)^* 1$$

GTU : Winter-13, Marks 7

Solution : FA stands for finite automata. The DFA is a deterministic finite automata in which for every input there is unique next state.

NFA stands for non-deterministic finite automata in which there can be more than one states for an input.

NFA with ϵ represents the non deterministic finite automata with ϵ transitions. The NFA with ϵ can be converted to NFA without ϵ and then to DFA.

r.e. $(0 + 1)^* (10 + 110)^* 1$

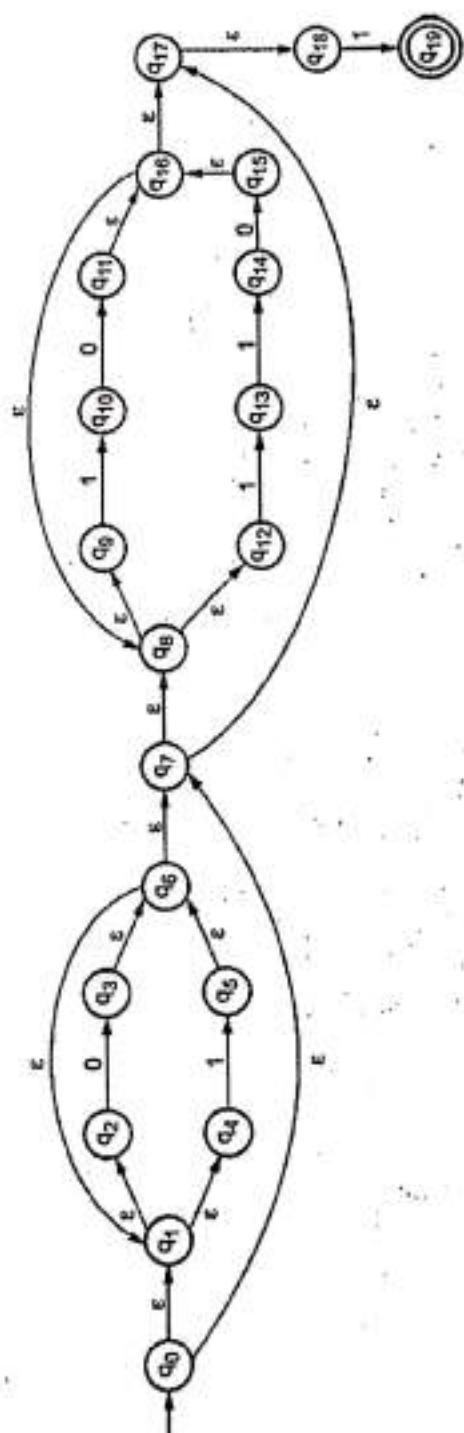


Fig. 2.11.15

→ Example 2.11.7 : Check the validity of the following equality with proper reason.
 $(00 * 1) * 1 = 1 + 0(0 + 10)^* 11$

GTU : Summer-16, Marks 7

Solution : The FA for L.H.S. and R.H.S. is

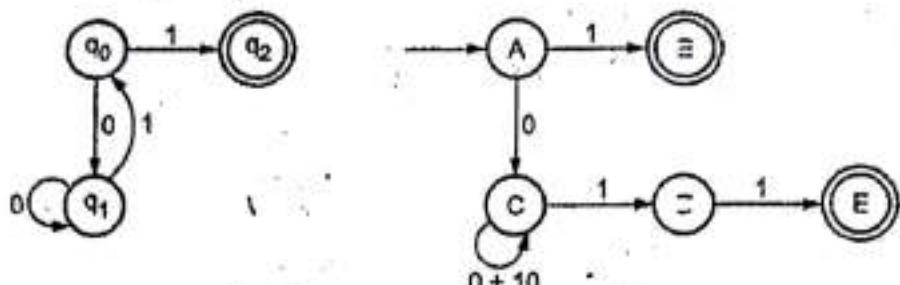


Fig. 2.11.16

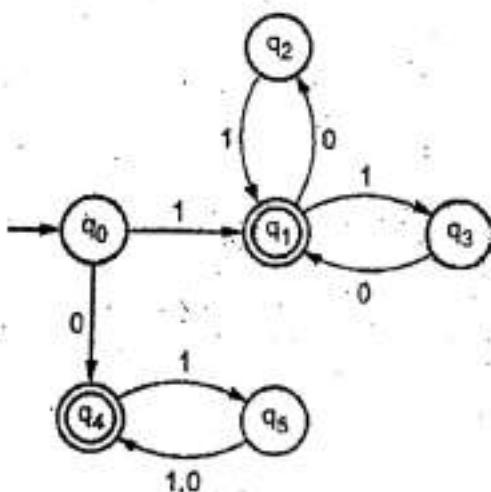
From these FA it is clear that the strings generated by both ends with either 1 or 11, the string consists of 01 as a substring. Hence L.H.S. = R.H.S. Thus both the r.e. are equivalent.

→ Example 2.11.8 : For the each of the following regular expressions, draw FA recognizing the corresponding language.

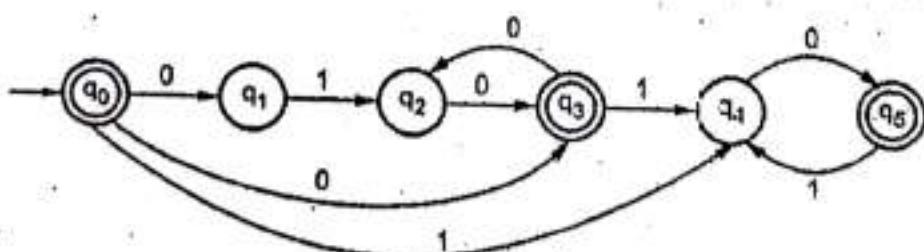
i) $1(01 + 10)^* + 0(11 + 10)^*$ ii) $(010 + 00)^*(10)^*$

GTU : Winter-14, Marks 7

Solution : i) $1(01 + 10)^* + 0(11 + 10)^*$



ii) $(010 + 00)^*(10)^*$

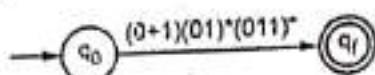


Example 2.11.9 : Design a FA for the regular expression $(0+1)(01)^*(011)^*$.

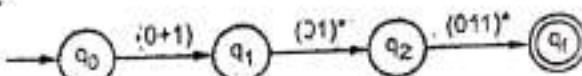
GTU : Winter-16, Marks 7

Solution : We draw FA for given regular expression using direct method

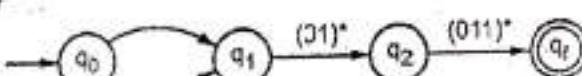
Step 1 :



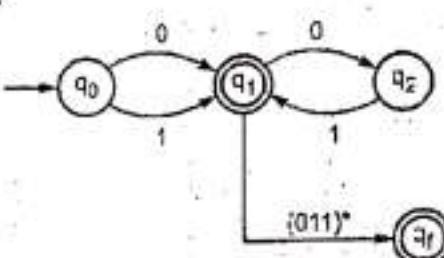
Step 2 :



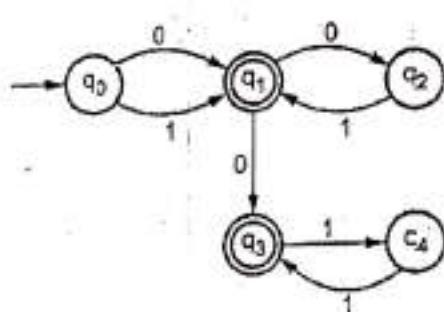
Step 3 :



Step 4 :



Step 5 :



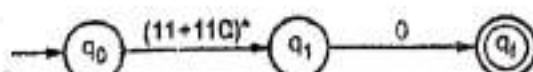
Example 2.11.10 : Draw a FA for following regular language.

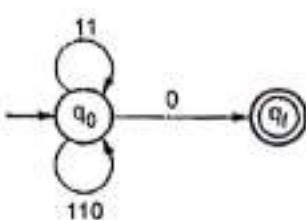
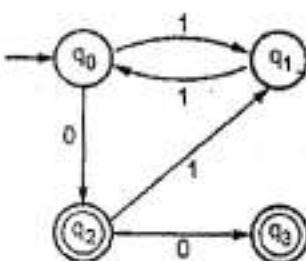
- i) $(11 + 110)^* 0$ ii) $(0 + 1)^*(10 + 11)$

GTU : Winter-17, Marks 4

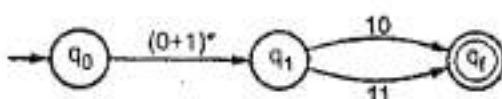
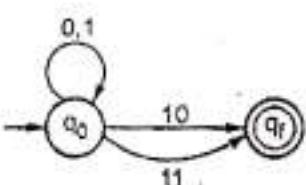
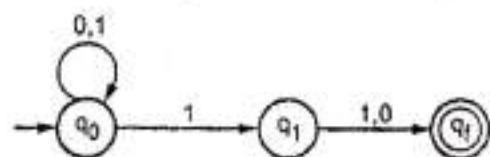
Solution : i) $(11 + 110)^* 0$

Step 1 :



Step 2 :**Step 3 :**

ii) $(0 + 1)^*(10 + 11)$

Step 1 :**Step 2 :****Step 3 :****Review Question**

1. Prove Kleene's Theorem Part 1 with illustration.

GTU : Summer-16, Marks 7

2.12 Minimization of Finite Automata

This method is also called as table filling method. This method is based on distinguishable and indistinguishable states.

Distinguishable states - If for some input string $w - \delta(p, w)$ gives accepting state and $\delta(q, w)$ gives non accepting states (or vice versa) then states p and q are called distinguishable states, or non-equivalent states.

Indistinguishable states - If for some input string $w - \delta(p, w)$ and $\delta(q, w)$ both produce either accepting states or non-accepting states then states p and q are called indistinguishable or equivalent states.

Let us understand this method with some illustrative examples.

→ Example 2.12.1 : Define distinguishable and indistinguishable states. Minimize the following DFA.

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

Solution : We will build the transitions table for given DFA as follows.

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

We will now construct a table for each pair of states.

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

We will mark X for all the final and non final states. Hence

B						
C	X	X				
D			X			
E				X		
F				X		
G				X		
H			X			
A	B	C	D	E	F	G

Thus we have marked X for (A, C), (B, C), (C, D), (C, E), (C, F), (C, G), (C, H).

Now we will consider every pair form above table. Consider pair (G, H).

$$\delta(G, 0) = G, \quad \delta(G, 1) = E$$

$$\delta(H, 0) = G, \quad \delta(H, 1) = C$$

As for $\delta(G, 1) = E$ and $\delta(H, 1) = C$. We can see X in (C, E) pair. Hence pair (G, H) is not equivalent.

Hence we will mark X in pair (G, H). Thus we will find the equivalent pairs.

Consider pair (B, H)

$$\delta(B, 0) = G, \quad \delta(B, 1) = C$$

$$\delta(H, 0) = G, \quad \delta(H, 1) = C$$

Thus the pair (B, H) is equivalent. Thus we obtain,

B	X						
C	X	X					
D	X	X	X				
E		X	X	X			
F	X	X	X		X		
G	X	X	X	X	X	X	
H	X		X	X	X	X	X
A	B	C	D	E	F	G	

Thus we get equivalent pairs as (A, E), (B, H), (D, F). Hence the minimized DFA will be,

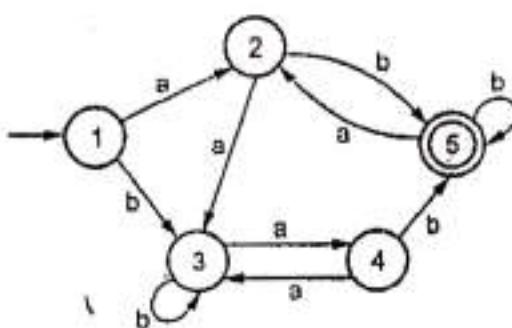
```

graph LR
    start(( )) --> A[A]
    A -- 0 --> B[B]
    A -- 1 --> D[D]
    B -- 0 --> G[G]
    B -- 1 --> C((C))
    C -- 0 --> A
    C -- 1 --> C
    D -- 0 --> C
    D -- 1 --> G
    G -- 0 --> G
    G -- 1 --> A
  
```

	0	1
A	B	D
B	G	C
C	A	C
D	C	G
G	G	A

→ Example 2.12.2 : Minimize the following DFA (if possible).

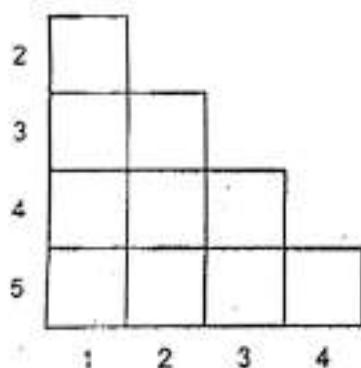
GTU : Summer-12, Marks 7



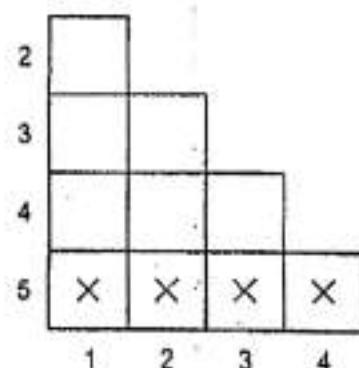
Solution : We will obtain transition table for given DFA..

Input State \	a	b
1	2	3
2	3	5
3	4	3
4	3	5
5	2	5

Step 1 : We will use table filling method for minimization.



Step 2 : From the constructed table, mark X for all final and nonfinal states.



Step 3 : Consider pair (1, 2)

$$\delta(1, a) = 2 \text{ and } \delta(1, b) = 3$$

$$\delta(2, a) = 3 \text{ and } \delta(2, b) = 5$$

As there is X in pair (3, 5). We will put X in pair (1, 2).

Step 4 : Consider pair (1, 3)

$$\delta(1, a) = 2 \text{ and } \delta(1, b) = 3$$

$$\delta(3, a) = 4 \text{ and } \delta(3, b) = 3$$

For input b both the pairs yield the same transition i.e. 3. There is no X mark for pair (2, 4) [The transitions for input a].

Hence we must check the pair (2, 4) first

$$\delta(2, a) = 3 \text{ and } \delta(2, b) = 5$$

$$\delta(4, a) = 3 \text{ and } \delta(4, b) = 5$$

Both the states i.e. (2, 4) gives out same target state. Hence pair (2, 4) is equivalent. Hence pair (1, 3) also becomes equivalent.

Step 5 : Consider pair (1, 4)

$$\delta(1, a) = 2 \text{ and } \delta(1, b) = 3$$

$$\delta(4, a) = 3 \text{ and } \delta(4, b) = 5$$

There is X in pair (3, 5). Hence X in pair (1, 4). After all these computations

Step 6 : Now the only remaining pair is (3, 4)

$$\delta(3, a) = 4 \text{ and } \delta(3, b) = 3$$

$$\delta(4, a) = 3 \text{ and } \delta(4, b) = 5$$

There is X in pair (3, 5), [The output states of b transitions].

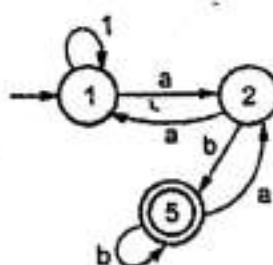
Thus pair (3, 5) is not equivalent.

The transition table becomes

	a	b
→	2	1
1	2	1
2	1	5
(5)	2	5

2	X		
3	✓		
4	X	✓	
5	X	X	X

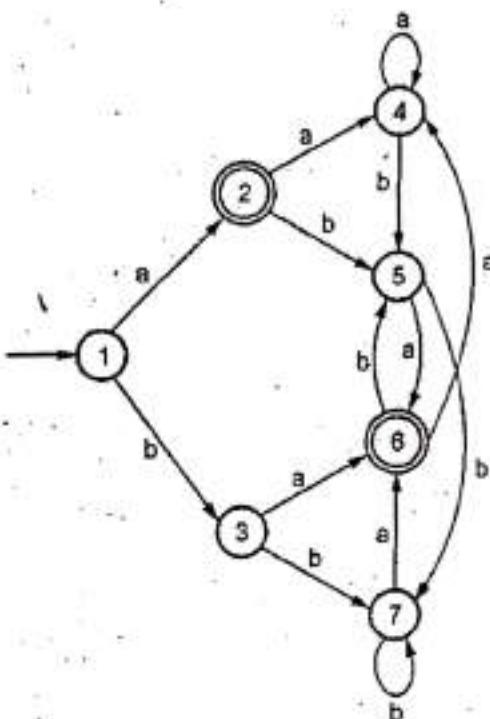
The transition graph becomes -



Thus we get $1 = 3$ and $2 = 4$.

⇒ **Example 2.12.3 :** For following NFA find minimum FA accepting same language

GTU : Summer-13, Marks : 7



Solution : We will design transition table for given FA.

State \ Up	a	b
1	2	3
2	4	5
3	6	7
4	4	5
5	6	7
6	4	5
7	6	7

From above transition table, state 2 and 6 are both final and both these states have, following transitions-

$$\delta(2, a) = 4 \quad \delta(6, a) = 4$$

$$\delta(2, b) = 5 \quad \delta(6, b) = 5$$

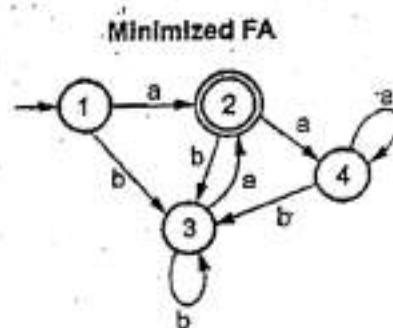
Thus we can minimize these two states to one state. Hence state 2 = state 6.

That means replace every occurrence of state 6 by state 2 and eliminate state 6. The transition table will be,

State	l/p	a	b
	1	2	3
2		4	5
3		2	7
4		4	5
5		2	7
7		2	7

From above table again state 3 = state 5 = state 7. Hence keep state 3 and eliminate state 5 and state 7. The minimized transition table will be -

Minimized Transition Table			
State	l/p	a	b
1		2	3
2		4	3
3		2	3
4		4	3

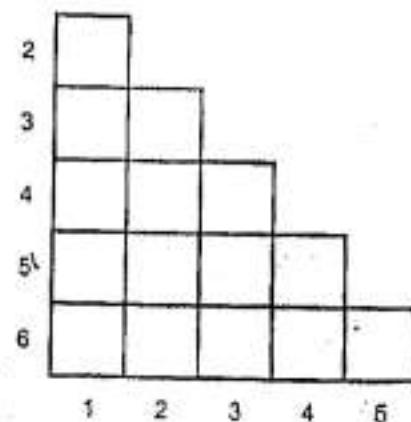


→ Example 2.12.4 : Minimize the following DFA (If possible GTU.: Summer-14, Marks 7)

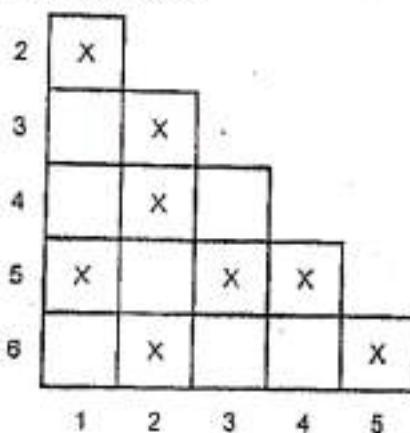
Solution : We will design transition table for given DFA.

Input State	a	b
1	2	1
2	4	1
3	2	5
4	6	3
5	4	5
6	5	2

We will use table filling method for minimization.



Mark X for all final and non final states.



Consider pair (1, 3)

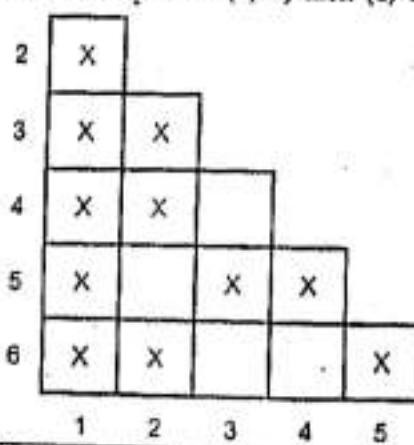
$$\delta(1, a) = 2$$

$$\delta(1, b) = 1$$

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

$$\delta(3, b) = 5$$

As $\delta(1, b) = 1$ and $\delta(3, b) = 5$ and there is X in (1, 5), the pair (1, 3) is not equivalent and put X in (1, 3). Similarly X can be put in (1, 4) and (1, 6).



Consider pair (3, 6)

$$\begin{array}{ll} \delta(3, a) = 2 & \delta(3, b) = 5 \\ \delta(6, a) = 5 & \delta(6, b) = 2 \end{array}$$

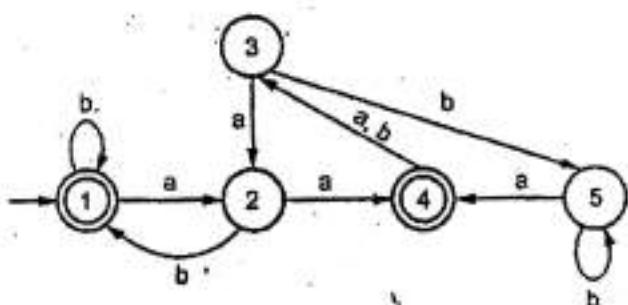
Both the states yield non-final states. $\therefore (3, 6)$ are equivalent and can be minimized. Pair (3, 4) are not equivalent as they emit final and non-final states. Similarly pair (4, 6) are not equivalent.

The table will be

	1	2	3	4	5
2	X				
3	X	X			
4	X	X	X		
5	X	✓	X	X	
6	X	X	✓	X	X

The minimize table by placing state 6 = 3

Input \ State	a	b
$\rightarrow 1$	2	1
2	4	1
3	2	5
4	3	3
5	4	5



Minimized DFA

Example 2.12.5 : Minimize the DFA shown in Fig. 2.12.1.

GTU : Summer-17, Marks 7

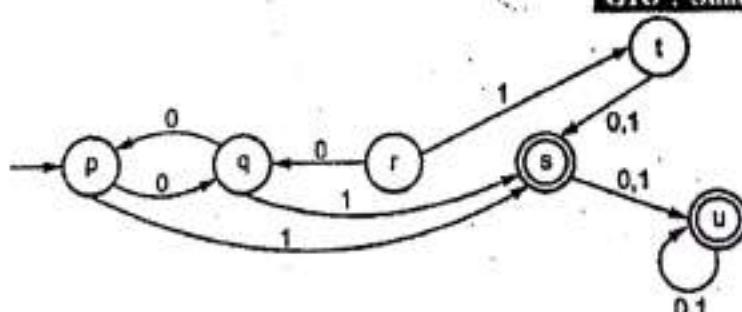


Fig. 2.12.1

Solution : We will design transition table for given FA

As S and u are both final states

l/p State \/ State	0	1
→ p	q	s
q	p	s
r	q	t
(S)	u	u
t	s	s
(u)	u	u

$$\text{and } \delta(S, 0) = u \quad \delta(u, 0) = u$$

$$\delta(S, 1) = u \quad \delta(u, 1) = u$$

Hence put S for every occurrence of u and eliminate u.

l/p State \/ State	0	1
p	q	s
q	p	s
r	q	t
(S)	s	s
t	s	s

The δ transition of state S and t are same but they can not be equivalent as S is a final state and t is a non-final state.

The minimized DFA will be

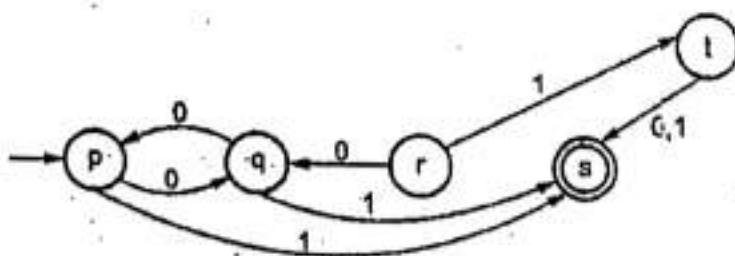


Fig. 2.12.2

2.13 Finite Automata with Output (Moore and Mealy Machine)

The finite automata is a collection of $(Q, \Sigma, \delta, q_0, F)$ where Q is a set of states including q_0 as a start state. In FA after reading the input string if we get final state then the string is said to be "acceptable". If we do not get final state then it is said that string is "rejected". That means there is no need of output for the finite Automata. The 'accept' or 'reject' acts like 'yes' or 'no' output for the machine. But if there is a need for specifying the output other than yes or no, then in such a case we require finite automata along with output. There are two types of FA with output and those are :

- 1) Moore machine 2) Mealy machine

1) Moore machine

Moore machine is a finite state machine in which the next state is decided by current state and current input symbol. The output symbol at a given time depends only on the present state of the machine. The formal definition of Moore machine is,

"Moore machine is a six tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where

Q is finite set of states.

Σ is finite set of input symbols.

Δ is an output alphabet.

δ is an transition function such that $Q \times \Sigma \rightarrow Q$. This is also known as state function.

λ is output function $Q \rightarrow \Delta$. This function is also known as machine function.

q_0 is the initial state of machine.

For example :

Consider the Moore machine given below -

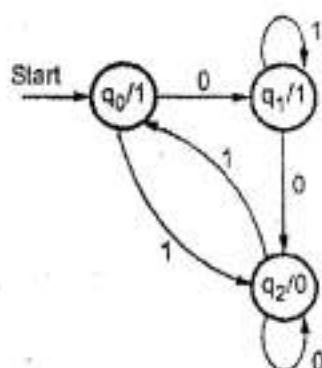


Fig. 2.13.1

The transition table will be -

Current state	Next state (δ)		Output (λ)
	0	1	
q_0	q_1	q_2	1
q_1	q_2	q_1	1
q_2	q_2	q_0	0

In Moore machine output is associated with every state. In the above given Moore machine when machine is in q_0 state the output will be 1. For the Moore machine if the length of input string is n then output string has length $n+1$.

For the string 0110 then the output will be 11110.

2) Mealy machine

Mealy machine is a machine in which output symbol depends upon the present input symbol and present state of the machine. The Mealy machine can be defined as -

Mealy machine is a six tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where

Q is finite set of states.

Σ is finite set of input symbols.

Δ is an output alphabet.

δ is state transition function such that $Q \times \Sigma \rightarrow Q$.

λ is machine function such that $Q \times \Sigma \rightarrow \Delta$.

q_0 is initial state of machine.

For example :

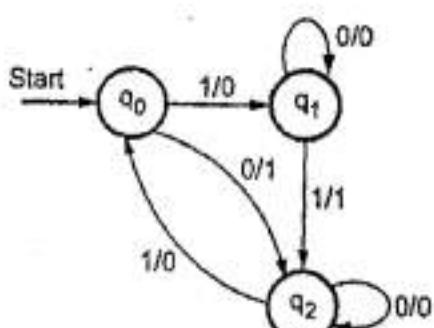


Fig. 2.13.2

For the input string 1001 the output will be 0001. In mealy machine the length of input string is equal to length of output string.

→ Example 2.13.1 : Design a Moore machine to generate 1's complement of given binary number.

GTU : Summer-19, Marks 3

Solution : To generate 1's complement of given binary number the simple logic which we will apply is that if input is 0 then output will be 1 and if input is 1 then output will be 0. That means there are three state one-start state, second state is for taking 0's as input and produces output as 1. Then third state is for taking 1's as input and producing output as 0.

Hence the Moore machine will shown in Fig. 2.13.3.

For instance, take one binary number 1011 then

Input :		1	0	1	1
State :	q_0	q_2	q_1	q_2	q_2
Output :	0	0	1	0	0

Thus we get 00100 as 1's complement of 1011, we can neglect the initial 0 and the output which we get is 0100 which is 1's complement of 1011.

The transition table can be drawn as below -

Current state	Next state		Output
	0	1	
q_0	q_1	q_2	0
q_1	q_1	q_2	1
q_2	q_1	q_2	0

Thus Moore machine $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$; where $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0, 1\}$, $\Delta = \{0, 1\}$.

The transition table shows the δ and λ functions.

→ **Example 2.13.2 :** Design a Moore and Mealy machine for a binary input sequence such that if it has a substring 101 the machine outputs A if input has substring 110 it outputs B otherwise it outputs C.

Solution : For designing such a machine we need to take care of two conditions and those are checking 101 and checking 110. If we get 101 the output will be A. If we recognize 110 the output will be B. For other strings the output will be C. We can make a partial design of it as shown in Fig. 2.13.4.

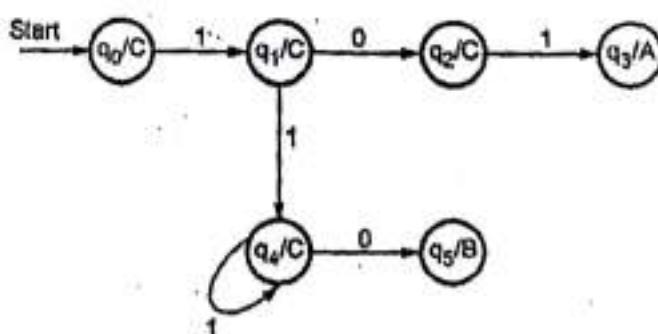
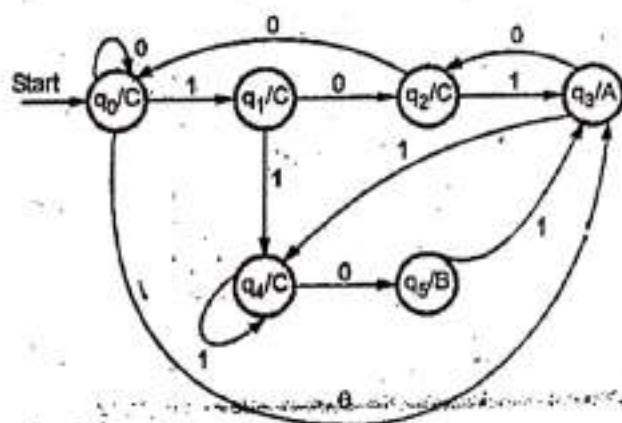
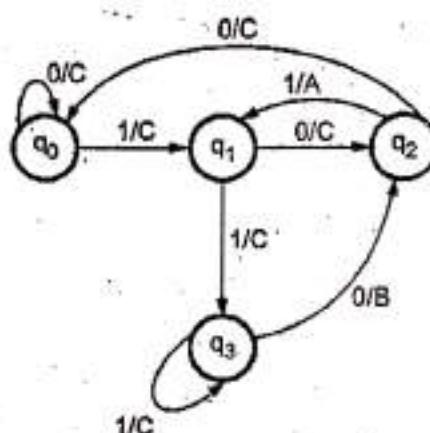


Fig. 2.13.4

Now we will insert the possibilities of 1's and 0's for each state. Then the Moore machine becomes



Now the Mealy machine can be



Example 2.13.3 : Design a Mealy machine to find 2's complement of a given binary number.

Solution : For designing 2's complement of a binary number we assume that input is read from LSB to MSB. We will keep the binary number as it is until we read first 1. Keep that 1 as it is then change remaining 1's by 0's and 0's by 1's.

For example :

Let the binary number be

1011

←

read from LSB

Keep the first 1 from LSB as it is and toggle the remaining bits we will get
0101

Thus 2's complement of 1011 is 0101. The required Mealy machine will be -

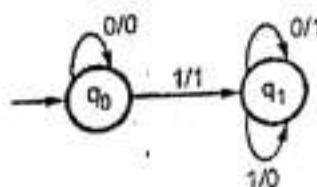


Fig. 2.13.5

→ Example 2.13.4 : Design a Moore machine which will increment the given binary number by 1.

Solution : We will read the binary number from LSB one bit at a time. We will replace each 1 by 0 until we get first 0. Once we get first 0 we will replace it by 1 and then keep remaining bits as it is.

For example :

1	0	1	1	← Read from LSB bit by bit
1	0	1	0	
			↑	
1	0	0	0	
		↑		
1	1	0	0	
	↑			
1	1	0	0	
	↑			

Using this logic we can build a Mealy machine as follows -

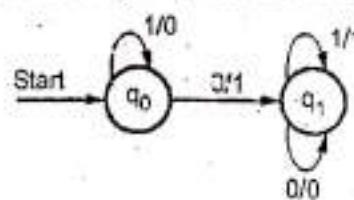


Fig. 2.13.6

→ Example 2.13.5 : Design a Mealy machine that uses its state to remember the last symbol read and emits output 'y' whenever current input matches to previous one, and emits n otherwise.

Solution : If 0 is read and immediately 0 is read then output will be 'y' because the current and next symbols will be same. But after reading 0 if we read 1 then output will be 'n' because the previous and next symbol will be different. Same is true for symbol 1. The Mealy machine will be shown in Fig. 2.13.7.

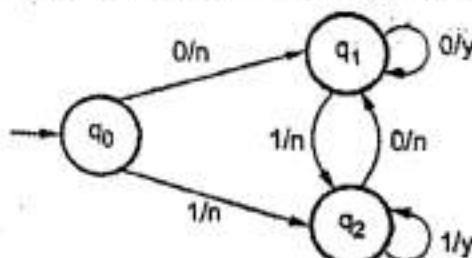


Fig. 2.13.7

→ **Example 2.13.6 :** Construct a Mealy machine which can output EVEN, ODD according as the total number of 1's encountered is even or odd. The input symbols are 0 and 1.

Solution : The number of 1's can be even or odd. Accordingly output will be EVEN or ODD. There is no restriction on number of 0's. The Mealy machine will be shown in Fig. 2.13.8.

Here A represents EVEN and B represents ODD.

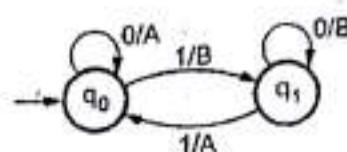


Fig. 2.13.8

2.13.1 Difference between Moore and Mealy Machine

Moore machine	Mealy machine
Only current state determines the output.	Input and current state determine the output.
Length of Moore machine is one longer than Mealy machine for given input.	Length of Mealy machine is smaller than the Moore machine for the given input.
Refer example 2.72.	Refer example 2.74.

2.13.2 Method for Conversion of Moore Machine to Mealy Machine

Let $M = (Q, \Sigma, \delta, \lambda, q_0)$ be a Moore machine. The equivalent Mealy machine can be represented by $M' = (Q, \Sigma, \delta, \lambda', q_0)$. The output function λ' can be obtained as -

$$\lambda'(q, a) = \lambda(\delta(q, a))$$

Let us understand this procedure with the help of some examples.

→ **Example 2.13.7 :** The Moore machine to determine residue mod 3 for binary number is given below. Convert it to Mealy equivalent machine. GTU : Winter-17, Marks 7

$Q \setminus \Sigma$	0	1	Output (λ)
q_0	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_1	q_2	2

Solution : The transition diagram for the given problem can be drawn as -

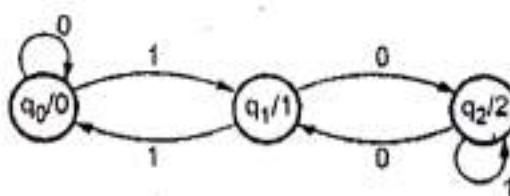


Fig. 2.13.9

The output function λ' can be obtained using following rule,

$$\lambda'(q, a) = \lambda(\delta(q, a))$$

Hence we will obtain output for every transition corresponding to input symbol.

$$\lambda'(q_0, 0) = \lambda(\delta(q_0, 0))$$

$$= \lambda(q_0) \quad \text{i.e. output of } q_0$$

$$\boxed{\lambda'(q_0, 0) = 0}$$

$$\lambda'(q_0, 1) = \lambda(\delta(q_0, 1))$$

$$= \lambda(q_1) \quad \text{i.e. output of } q_1$$

$$\boxed{\lambda'(q_0, 1) = 1}$$

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

$$= \lambda(q_2)$$

$$\boxed{\lambda'(q_1, 0) = 2}$$

$$\lambda'(q_1, 1) = \lambda(\delta(q_1, 1))$$

$$= \lambda(q_0)$$

$$\boxed{\lambda'(q_1, 1) = 0}$$

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

$$= \lambda(q_1)$$

$$\boxed{\lambda'(q_2, 0) = 1}$$

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

$$= \lambda(q_2)$$

$$\boxed{\lambda'(q_2, 1) = 2}$$

Hence the transition table can be drawn as follows -

Q	Σ	Input 0		Input 1	
		State	O/P	State	O/P
q_0	q_0	0		q_1	1
q_1	q_2	2		q_0	0
q_2	q_1	1		q_2	2

The transition diagram of Mealy machine is,

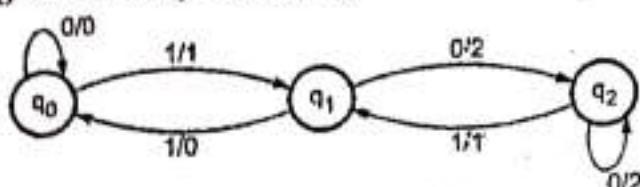


Fig. 2.13.10

The input string 10011 then the output for Mealy machine will be

Next state $q_1 \rightarrow q_2 \rightarrow q_2 \rightarrow q_1 \rightarrow q_0$

Output 1 2 2 1 0

The output sequence for Moore machine will be

Input 1 0 0 1 1

Next state $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_2 \rightarrow q_1 \rightarrow q_0$

Output 0 1 2 2 1 0

The length of output sequence is $n+1$ in Moore machine and is n in Mealy machine which is desired.

Example 2.13.8 : Convert the following Moore machine into equivalent Mealy machine
 $M = (\{q_0, q_1\}, \{a, b\}, \{0, 1\} \delta, \lambda, q_0)$

Solution :

δ	a	b	Output (λ)
q_0	q_0	q_1	0
q_1	q_0	q_1	1

The equivalent Mealy machine can be obtained as follows -

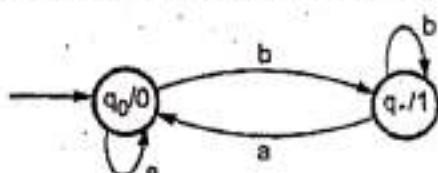


Fig. 2.13.11

$$\lambda'(q_0, a) = \lambda(\delta(q_0, a))$$

$$= \lambda(q_0)$$

$$= 0$$

$$\lambda'(q_0, b) = \lambda(\delta(q_0, b))$$

$$= \lambda(q_1)$$

$$= 1$$

$$\lambda'(q_1, a) = \lambda(\delta(q_1, a))$$

$$\begin{aligned}
 &= \lambda(q_0) \\
 &= 0 \\
 \lambda'(q_1, b) &= \lambda(\delta(q_1, b)) \\
 &= \lambda(q_1) \\
 &= 1
 \end{aligned}$$

Hence the transition table can be drawn as follows -

Q	Σ	Input a		Input b	
		State	O/P	State	O/P
q_0		q_0	0	q_1	1
q_1		q_0	0	q_1	1

The equivalent Mealy machine will be,

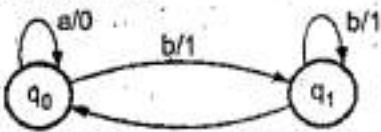


Fig. 2.13.12

Consider the output sequence for Moore machine for input sequence,
a b b a

Then

$$\begin{array}{ccccccc}
 q_0 & \xrightarrow{a} & q_0 & \xrightarrow{b} & q_1 & \xrightarrow{b} & q_1 & \xrightarrow{a} & q_0 \\
 0 & & 0 & & 1 & & 1 & & 0
 \end{array}$$

Similarly output sequence for Mealy machine will be

$$\begin{array}{ccccccc}
 q_0 & \xrightarrow{a} & q_1 & \xrightarrow{b} & q_1 & \xrightarrow{b} & q_1 & \longrightarrow & q_0 \\
 0 & & 1 & & 1 & & 1 & & 0
 \end{array}$$

We can note that length of Moore machine is $n+1$ and that of Mealy machine is n .

→ **Example 2.13.9 :** Convert the Moore machine shown in Fig. 2.13.13 into an equivalent Mealy machine.

GTU : Summer-17, Marks 4

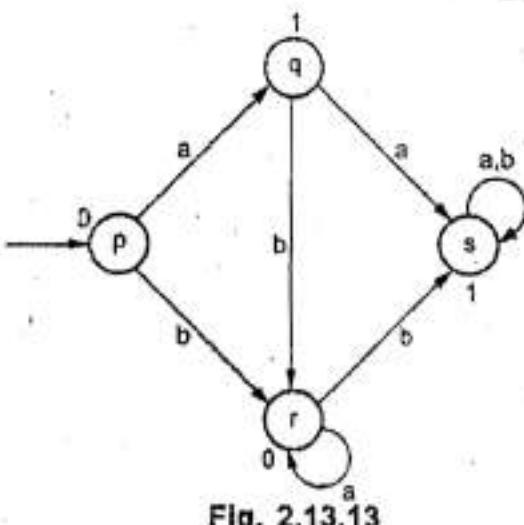


Fig. 2.13.13

Solution : The transition graph is

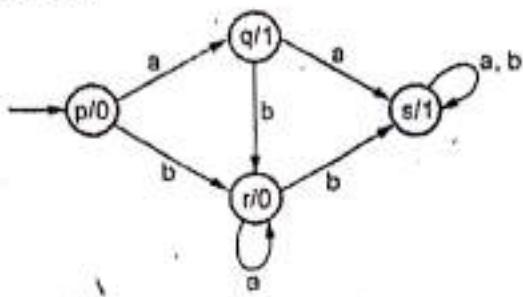


Fig. 2.13.13 (a)

The output function λ' can be obtained using following rule

$$\lambda'(q, a) = \lambda(\delta(q, a))$$

Hence we will obtain output for every transition corresponding to input symbol

$$\lambda'(p, a) = \lambda(\delta(p, a)) = \lambda(q) \text{ i.e. output of } q$$

$$\therefore \lambda'(p, a) = 1$$

$$\lambda'(p, b) = \lambda(\delta(p, b)) = \lambda(r)$$

$$\lambda'(p, b) = 0$$

$$\lambda'(q, a) = \lambda(\delta(q, a)) = \lambda(s)$$

$$\lambda'(q, a) = 1$$

$$\lambda'(q, b) = \lambda(\delta(q, b)) = \lambda(r)$$

$$\lambda'(q, b) = 0$$

$$\lambda'(r, a) = \lambda(\delta(r, a)) = \lambda(r)$$

$$\lambda'(r, a) = 0$$

$$\lambda'(r, b) = \lambda(\delta(r, b)) = \lambda(s)$$

$$\lambda'(r, b) = 1$$

$$\lambda'(s, a) = \lambda(\delta(s, a)) = \lambda(s)$$

$$\lambda'(s, a) = 1$$

$$\lambda'(s, b) = \lambda(\delta(s, b)) = \lambda(s)$$

$$\lambda'(s, b) = 1$$

Hence transition table using above computations will be :

Σ		Input a		Input b	
		state	o/p	state	o/p
Q	q	1	r	0	
	s	1	r	0	
r	r	0	s	1	
	s	1	s	1	

The mealy machine can be shown by following transition graph:

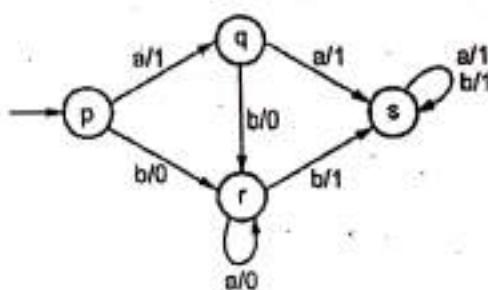


Fig. 2.13.14

→ Example 2.13.10 : Convert the given Moore machine into Mealy machine. Draw state transition diagram of Mealy machine.

GTU : Summer-18, Marks 4

Present State	Next State		Output
	0	1	
$\rightarrow p_0$	r	q_0	e
p_1	r	q_0	1
q_0	p_1	s_0	0
q_1	p_1	s_0	1
r	q_1	p_1	0
s_0	s_1	r	0
s_1	s_1	r	1

Solution : We will draw the transition graph as follows :

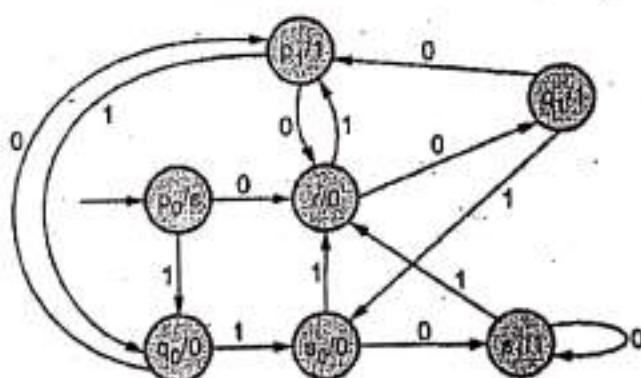


Fig. 2.13.15

The output function λ' can be obtained using following rule

$$\lambda'(q, a) = \lambda(\delta(q, a))$$

Hence we will obtain output for every transition corresponding to inputs symbol.

$$\lambda'(p_0, 0) = \lambda(\delta(p_0, 0)) = \lambda(r) = 0$$

$$\lambda'(p_0, 1) = \lambda(\delta(p_0, 1)) = \lambda(q_0) = 0$$

$$\lambda'(p_1, 0) = \lambda(\delta(p_1, 0)) = \lambda(r) = 0$$

$$\lambda'(p_1, 1) = \lambda(\delta(p_1, 1)) = \lambda(q_0) = 0$$

$$\lambda'(q_0, 0) = \lambda(\delta(q_0, 0)) = \lambda(p_1) = 1$$

$$\lambda'(q_0, 1) = \lambda(\delta(q_0, 1)) = \lambda(s_0) = 0$$

$$\lambda'(q_1, 0) = \lambda(\delta(q_1, 0)) = \lambda(p_1) = 1$$

$$\lambda'(q_1, 1) = \lambda(\delta(q_1, 1)) = \lambda(s_0) = 0$$

$$\lambda'(r, 0) = \lambda(\delta(r, 0)) = \lambda(q_1) = 1$$

$$\lambda'(r, 1) = \lambda(\delta(r, 1)) = \lambda(p_1) = 1$$

$$\lambda'(s_0, 0) = \lambda(\delta(s_0, 0)) = \lambda(s_1) = 1$$

$$\lambda'(s_0, 1) = \lambda(\delta(s_0, 1)) = \lambda(r) = 0$$

$$\lambda'(s_1, 0) = \lambda(\delta(s_1, 0)) = \lambda(s_1) = 1$$

$$\lambda'(s_1, 1) = \lambda(\delta(s_1, 1)) = \lambda(r) = 0$$

Hence transition table using above computations will be -

Σ	Input 0		Input 1	
Q	State	Output	State	Output
p ₀	p ₀	0	p ₁	0
p ₁	p ₁	0	p ₀	0
q ₀	q ₁	1	q ₀	0
q ₁	q ₀	0	q ₁	1
r	p ₁	1	p ₁	1
s ₀	s ₁	1	s ₀	0
s ₁	s ₁	1	r	0

The Mealy machine can be shown by following graph.

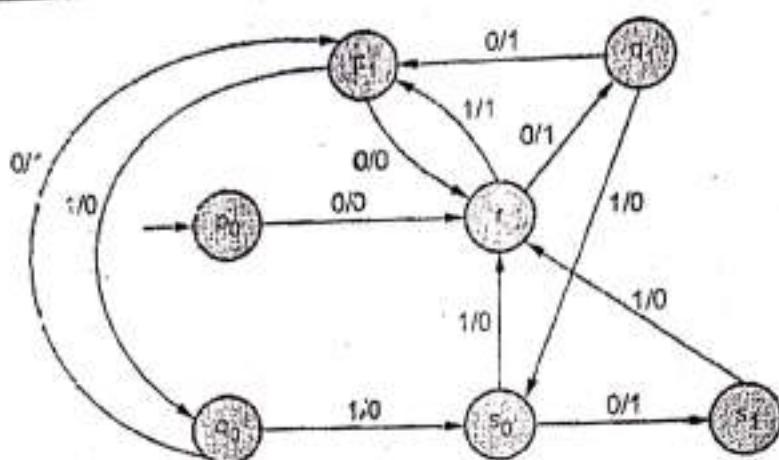


Fig. 2.13.16

2.13.3 Method for Conversion of Mealy Machine to Moore Machine

In Moore machine the output is associated with every state and in Mealy machine the output is given along the edge with input symbol. To convert Moore machine to Mealy machine state output symbols are distributed to input symbol paths. But while converting Mealy machine to Moore machine we will create a separate state for every new output symbol and according incoming and outgoing edges are distributed.

Let $M = (Q, \Sigma, \Delta, \delta, q_0)$ be a Mealy machine then there exists a Moore machine M' equivalent to M .

The M' can be given as $M' = (Q \times \Delta, \Sigma, \Delta, \delta', \lambda', [q_0, b_0])$ where b_0 is an output symbol selected from Δ . We will calculate δ' and λ' as follows -

$$\delta'([q, b], a) = [\delta(q, a), \lambda(q, a)]$$

$$\lambda([q, b]) = b$$

δ defines the move made by M' on input a .

M' on input a .

λ' defines the output made for the corresponding state q .

Thus we have to calculate δ' and λ' for $q_0, q_1, q_2, \dots, q_n$ on input a_0, a_1, \dots, a_n and should emit the outputs $b_1, b_2, b_3, \dots, b_n$.

Let us understand this method of conversion with the help of some examples.

→ Example 2.13.11 : Convert the following Mealy machine into equivalent Moore machine.

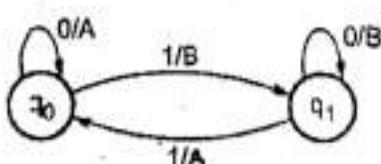


Fig. 2.13.17

Solution : The states for Moore machine will be $[q_0, A]$, $[q_0, B]$, $[q_1, A]$, $[q_1, B]$. Then we will calculate δ' and λ' as follows -

$$\begin{aligned}\delta'([q_0, A], 0) &= [\delta(q_0, 0), \lambda(q_0, 0)] \\ &= [q_0, A]\end{aligned}$$

$$\lambda'([q_0, A]) = A$$

$$\begin{aligned}\delta'([q_1, A], 0) &= [\delta(q_1, 0), \lambda(q_1, 0)] \\ &= [q_1, B]\end{aligned}$$

$$\lambda'([q_1, A]) = A$$

$$\begin{aligned}\delta'([q_1, A], 1) &= [\delta(q_1, 1), \lambda(q_1, 1)] \\ &= [q_0, A]\end{aligned}$$

$$\lambda'([q_1, A]) = A$$

Hence,

	0	1	Output
$[q_0, A]$	$[q_0, A]$	$[q_1, B]$	A
$[q_0, B]$	$[q_0, A]$	$[q_1, B]$	B
$[q_1, A]$	$[q_1, B]$	$[q_0, A]$	A

$$\begin{aligned}\delta'([q_1, B], 0) &= [\delta(q_1, 0), \lambda(q_1, 0)] \\ &= [q_1, B]\end{aligned}$$

$$\lambda'([q_1, B]) = B$$

$$\begin{aligned}\delta'([q_1, B], 1) &= [\delta(q_1, 1), \lambda(q_1, 1)] \\ &= [q_0, B]\end{aligned}$$

$$\lambda'([q_1, B]) = B$$

Finally the transition table will be ,

$$\delta'([q_0, A], 1) = [\delta(q_0, 1), \lambda(q_0, 1)] = [q_1, B]$$

$$\lambda'([q_0, A]) = A$$

Hence we can show a partial transition table as,

	0	1	Output
$[q_0, A]$	$[q_0, A]$	$[q_1, B]$	A

Continuing in this fashion we can calculate δ' and λ' as follows -

$$\begin{aligned}\delta'([q_0, B], 0) &= [\delta(q_0, 0), \lambda(q_0, 0)] \\ &= [q_0, A] \\ \lambda([q_0, B]) &= B \\ \delta'((q_1, B), 1) &= [\delta(q_0, 1), \lambda(q_0, 1)] \\ &= [q_1, B] \\ \lambda'([q_0, B]) &= B\end{aligned}$$

Hence

	0	1	Output
[q ₀ , A]	[q ₀ , A]	[q ₁ , B]	A
[q ₀ , B]	[q ₀ , A]	[q ₁ , B]	B

State \ I/P	0	1	Output
[q ₀ , A]	[q ₀ , A]	[q ₁ , B]	A
[q ₀ , B]	[q ₀ , A]	[q ₁ , B]	B
[q ₁ , A]	[q ₁ , B]	[q ₀ , A]	A
[q ₁ , B]	[q ₁ , B]	[q ₀ , A]	B

The transition diagram will be,

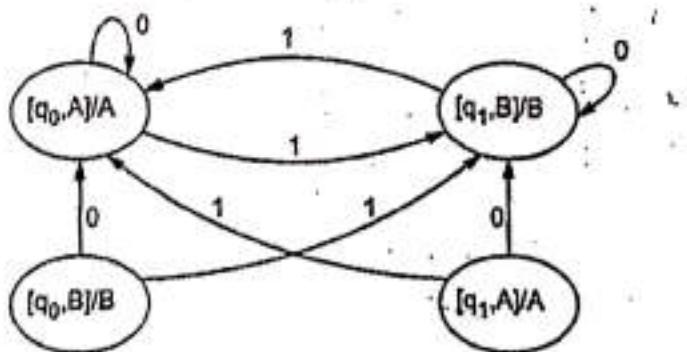


Fig. 2.13.18

→ Example 2.13.12 : Convert the following Mealy machine into equivalent Moore machine.

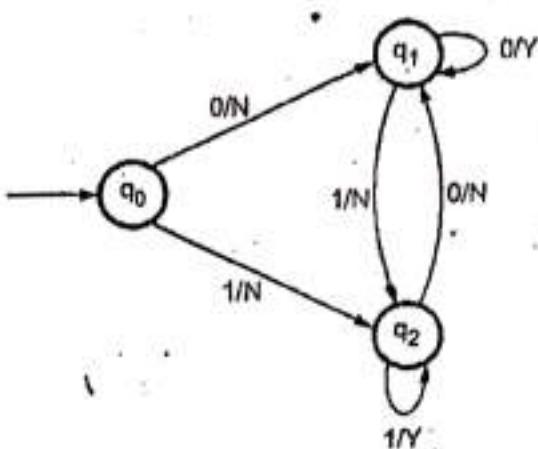


Fig. 2.13.19

Solution : We will write the transition table for given transition graph -

	Input 0	Output	Input 1	Output
q_0	q_1	N	q_2	N
q_1	q_1	Y	q_2	N
q_2	q_1	N	q_2	Y

Now we will find out the states and corresponding outputs for Moore machine. The states for Moore machine will be -

$[q_0, N]$, $[q_0, Y]$, $[q_1, N]$, $[q_1, Y]$, $[q_2, N]$, $[q_2, Y]$

Now let us calculate δ' and λ' for all the above given states -

$$\begin{aligned}\delta'([q_0, N], 0) &= [\delta(q_0, 0), \lambda(q_0, 0)] \\ &= [q_1, N]\end{aligned}$$

$$\lambda'([q_0, N]) = N$$

Similarly,

$$\begin{aligned}\delta'([q_0, N], 1) &= [\delta(q_0, 1), \lambda(q_0, 1)] \\ &= [q_2, N]\end{aligned}$$

$$\lambda'([q_0, N]) = N$$

The partial transition table is -

State \ I/P	0	1	Output
$[q_0, N]$	$[q_1, N]$	$[q_2, N]$	N

Now, for remaining states the corresponding transitions and outputs can be obtained as follows -

$$\delta'([q_0, Y], 0) = [\delta(q_0, 0), \lambda(q_0, 0)] \\ = [q_1, N]$$

$$\lambda'([q_0, Y]) = Y$$

$$\delta'([q_0, Y], 1) = [\delta(q_0, 1), \lambda(q_0, 1)] \\ = [q_2, N]$$

$$\lambda'([q_0, Y]) = Y$$

$$\delta'([q_1, N], 0) = [\delta(q_1, 0), \lambda(q_1, 0)] \\ = [q_1, Y]$$

$$\lambda'([q_1, N]) = N$$

$$\delta'([q_1, N], 1) = [\delta(q_1, 1), \lambda(q_1, 1)] \\ = [q_2, N]$$

$$\lambda'([q_1, N]) = N$$

$$\delta'([q_1, Y], 0) = [\delta(q_1, 0), \lambda(q_1, 0)] \\ = [q_1, Y]$$

$$\lambda'([q_1, Y]) = Y$$

$$\delta'([q_2, N], 0) = [\delta(q_2, 0), \lambda(q_2, 0)] = [q_1, N] \\ \lambda'([q_2, N]) = N$$

$$\delta'([q_2, N], 1) = [\delta(q_2, 1), \lambda(q_2, 1)] = [q_2, Y] \\ \lambda'([q_2, N]) = N$$

$$\delta'([q_1, Y], 1) = [\delta(q_1, 1), \lambda(q_1, 1)] \\ = [q_2, N]$$

$$\lambda'([q_1, Y]) = Y$$

$$\delta'([q_2, Y], 0) = [\delta(q_2, 0), \lambda(q_2, 0)] \\ = [q_1, N]$$

$$\lambda'([q_2, Y]) = Y$$

$$\delta'([q_2, Y], 1) = [\delta(q_2, 1), \lambda(q_2, 1)] \\ = [q_2, Y]$$

$$\lambda'([q_2, Y]) = Y$$

The transition table can be drawn as -

State \ Σ	0	1	Output
[q ₀ , N]	[q ₁ , N]	[q ₂ , N]	N
[q ₀ , Y]	[q ₁ , N]	[q ₂ , N]	Y
[q ₁ , N]	[q ₁ , Y]	[q ₂ , N]	N
[q ₁ , Y]	[q ₁ , Y]	[q ₂ , N]	Y
[q ₂ , N]	[q ₁ , N]	[q ₂ , Y]	N
[q ₂ , Y]	[q ₁ , N]	[q ₂ , Y]	Y

- Example 2.13.13 : Convert the Mealy machine shown in given figure into Moore machine.

GTU : Summer-16, Marks 7

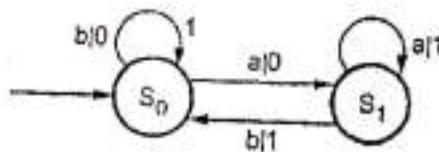


Fig. 2.13.20

Solution : We will write transition table for given transition graph

Input a	Output	Input b	Output
[S ₀ , 0]	S ₁	0	[S ₁ , 0]
[S ₁ , 0]	S ₁	1	[S ₀ , 1]

The states for Moore machine will be

$$[S_0, 0], [S_0, 1], [S_1, 0], [S_1, 1]$$

Now let us calculate δ' and λ' for these states

$$\delta'([S_0, 0], a) = [\delta(S_0, a), \lambda(S_0, a)] = [S_1, 0]$$

$$\lambda'([S_0, 0]) = 0$$

Similarly,

$$\delta'([S_0, 0], b) = [\delta(S_0, b), \lambda(S_0, b)] = [S_0, 0]$$

$$\lambda'([S_0, 0]) = 0$$

$$\delta'([S_0, 1], a) = [\delta(S_0, a), \lambda(S_0, a)] = [S_1, 0]$$

$$\lambda'([S_0, 1]) = 1$$

$$\delta'([S_0, 1], b) = [\delta(S_0, b), \lambda(S_0, b)] = [S_0, 0]$$

$$\lambda'([S_0, 1]) = 1$$

$$\delta'([S_1, 0], a) = [\delta(S_1, a), \lambda(S_1, a)] = [S_1, 1]$$

$$\lambda'([S_1, 0]) = 0$$

$$\delta'([S_1, 0], b) = [\delta(S_1, b), \lambda(S_1, b)] = [S_0, 1]$$

$$\delta'([S_1, 0]) = 0$$

$$\delta'([S_1, 1], a) = [\delta(S_1, a), \lambda(S_1, a)] = [S_1, 1]$$

$$\delta'([S_1, 1]) = 1$$

$$\delta'([S_1, 1], b) = [\delta(S_1, b), \lambda(S_1, b)] = [S_0, 1]$$

$$\lambda'([S_1, 1]) = 1$$

The transition table for above δ' computations is as given below.

Σ	a	b	Output
State			
[$S_0, 0$]	[$S_1, 0$]	[$S_0, 0$]	0
[$S_0, 1$]	[$S_1, 0$]	[$S_0, 0$]	0
[$S_1, 0$]	[$S_1, 1$]	[$S_0, 1$]	0
[$S_1, 1$]	[$S_2, 1$]	[$S_1, 1$]	1

The transition diagram for Moore machine is

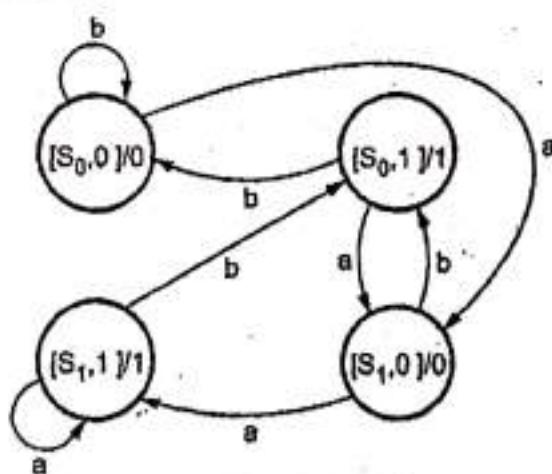


Fig. 2.13.21

Review Question

1. Explain Moore machine and Mealy machine.

GTU : Winter-18, Marks 7

2.14 Applications of Regular Expressions and Limitations of FA

The regular expressions can be modeled by finite automata. We have solved many examples and experienced that regular expressions are effective representation of languages. The smaller unit of regular expression can express the given language over certain input set.

There are following applications of regular expressions and finite automata.

1. Regular expressions in UNIX

There are various UNIX notations used for regular expressions. These notations have many additional capabilities and features. These notations have ability to name and refer previous strings that have a matched pattern. This ability helps in recognizing nonregular languages. UNIX regular expressions allow to write character classes. There are some rules for these character classes which are as given below -

- 1) The symbol · stands for any single character.
- 2) The sequence [1, 2, 3, ... 10] means $1 + 2 + 3 + \dots + 10$.

3) The range of characters can be specified using square brackets.

For example : [a-z] denotes set of lower case letters.

- 4) For matching with some special character a backslash is used.

For example : \- means match with character -.

- 5) Special notations can be used to represent some common class of characters. For example : [:digit:] represents the class [0-9]. The [:alpha:] is same as [A-Za-Z].

- 6) The operator | is used to denote union.

- 7) ? is used to denote preceding zero or single character.

For example : - ? [0-9] means the number can be positive or negative number.

- 8) + is used to denote one or more occurrences.

- 9) * is used to denote zero or more occurrences.

- 10) {n} means n copies of.

For example : a {3} means aaa.

2. Lexical analyzers :

Compiler uses this program of lexical analyzer in the process of compilation. The task of lexical analyzer is to scan the input program and separate out the "tokens".

For example : Identifier is a category of token in the source language and it can be identified by regular expression as

(letter) (letter + digit)*

If anything in the source language matches with this regular expression then it is recognized as identifier. The letter is nothing but a set {A, B, ... Z, a, b, ... z} * and digit is {0, 1, ... 9} *. The regular expression is effective way for identifying token from a language.

We can represent the above regular expression using a finite automata as follows-

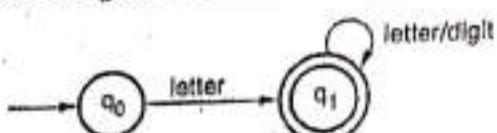


Fig. 2.14.1

3. Finding patterns in text

Regular expressions are useful notations used for finding the patterns from given text. A large class of pattern can be identified by regular expression.

For example : The strings ending with digits is a pattern in the given text which can be recognized by following regular expression -

$[a-zA-Z] + [0-9] +$

Thus use of regular expressions for identifying patterns from given text makes the job of compiler more simplified.

In web applications also the pattern matching is done using regular expressions.

4. GREP utility in unix

In Unix there is a GREP utility for searching the desired pattern in the file. For searching the pattern from the Grep makes use of regular expressions. When particular string is present in the file, then using grep we get the line containing that string from the file.

For example : If we type the grep command on Unix prompt then we will get the output as follows \$ grep hello test.txt.

hello friends

\$ grep b.g test.txt.

big brother ← these are the lines

bug ← present in test.txt.

bag of books ← file containing

bigger ← b[any single character] g

Here dot stands for matching with one character.

Limitations of FA

The main limitation of FA is that it can not remember arbitrarily long input sequence because it does not contain any memory. Hence it can not solve following types of problems.

1. Checking well-formedness of parenthesis.
2. Checking the palindrome condition of given language.

Thus FA cannot accept non-regular or context free languages.

2.15 Union, Intersection and Complement of Regular Languages

Union Operation :

The union of two regular languages is regular. If L_1 and L_2 are two languages then $L_1 \cup L_2 = L(R_1 + R_2)$. That means the L denotes the language which is accepted by both L_1 and L_2 .

Complement Operation :

The complement of regular language is regular. Consider L_1 be regular language which is accepted by a DFA $M = (Q, \Sigma, \delta, q_0, F)$. The complement of regular language is \bar{L}_1 which is accepted by $M' = (Q, \Sigma, \delta, q_0, Q - F)$. That means M' is a DFA with final states $\in F$ and M' is a DFA in which all the non-final states of M become final. In other words, we can say that the strings that are accepted by M are rejected by M' similarly, the strings rejected by M are accepted by M' .

Intersection Operation :

If L_1 and L_2 are two regular languages then $L_1 \cap L_2$ is regular. Consider that languages L_1 is regular. That means there exists some DFA M_1 that accepts L_1 . We can write $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$. Similarly being L_2 regular there is another DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.

Let, L be the language obtained from $L_1 \cap L_2$. We can simulate $M = (Q, \Sigma, \delta, q, F)$.

where $Q = Q_1 \cap Q_2$

$\delta = \delta_1 \cap \delta_2$ a mapping function derived from both the DFAS.

$q \in Q$ which is initial state of machine M .

$F = F_1 \cap F_2$, the set of final states, which is common for M_1 and M_2 both.

Difference Operation :

If L_1 and L_2 are two regular languages then $L_1 - L_2$ is regular. The $L_1 - L_2$ can also be denoted as $L_1 \cup \bar{L}_2$.

Consider L_1 be regular language which is accepted by DFA $M = (Q, \Sigma, \delta, q_0, F)$. The complement of regular language L_1 is \bar{L}_1 which is accepted by $M' = (Q, \Sigma, \delta, q_0, Q - F)$. That means M' is a DFA with final state's set F and M' is a DFA in which all the non final states of M become final states and all the final states of M become non-final states. Thus L_1 and \bar{L}_2 are two regular languages. That also means : these languages are accepted by regular expressions. If $L_1 = L(R_1)$ and $\bar{L}_2 = L(R'_2)$. Then $L_1 \cup \bar{L}_2 = L(R_1 + R'_2)$.

Example 2.15.1 : Draw Finite Automata (FA) for following languages :

$$L_1 = \{x / 00 \text{ is not a substring of } x\}$$

$$L_2 = \{x / x \text{ ends with } 01\}$$

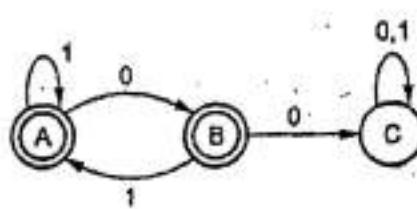
Find FA accepting languages i) $L_1 \cap L_2$ and ii) $L_2 - L_1$.

GTU : Summers-11, Marks 9, Summer-15, Marks 5, Summer-19, Marks 7

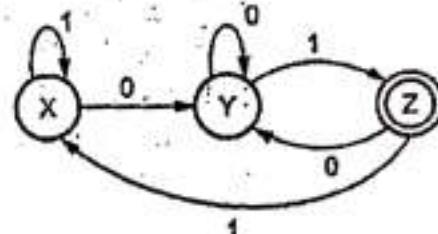
Solution : We will construct a combined transition graph considering both the FA's for the input 0 or 1 transitions. The process of creating combined transition graph is as follows -

Step 1 :

FA for L_1



FA for L_2



Step 2 :

Now we will compute δ transitions

$$\delta(A, 0) = B$$

$$\therefore \delta((A, X), 0) = (B, Y) \text{ New state}$$

$$\delta(X, 0) = Y$$

$$\delta(A, 1) = A$$

$$\delta(X, 1) = X$$

$$\therefore \delta((A, X), 1) = (A, X)$$

As one new state is getting generated i.e. (B, Y) we will contain δ

transition on it

$$\delta(B, 0) = C$$

$$\therefore \delta((B, Y), 0) = (C, Y) \text{ New state}$$

$$\delta(Y, 0) = Y$$

$$\delta(B, 1) = A$$

$$\therefore \delta((B, Y), 1) = (A, Z) \text{ New state}$$

$$\delta(Y, 1) = Z$$

Now consider $\delta(C, Y)$

$$\delta(C, 0) = C$$

$$\therefore \delta((C, Y), 0) = (C, Y)$$

$$\delta(Y, 0) = Y$$

$$\delta(C, 1) = C$$

$\therefore \delta((C, Y), 1) = (C, Z)$ New state

$$\delta(Y, 1) = Z$$

Consider $\delta(A, Z)$

$$\delta(A, 0) = B$$

$\therefore \delta((A, Z), 0) = (B, Y)$

$$\delta(Z, 0) = Y$$

$$\delta(A, 1) = A$$

$\therefore \delta((A, Z), 1) = (A, X)$

$$\delta(Z, 1) = X$$

Consider $\delta(C, Z)$

$$\delta(C, 0) = C$$

$\therefore \delta((C, Z), 0) = (C, Y)$

$$\delta(Z, 0) = Y$$

$$\delta(C, 1) = C$$

$\therefore \delta((C, Z), 1) = (C, X)$ New state

$$\delta(Z, 1) = X$$

Consider $\delta(C, X)$

$$\delta(C, 0) = C$$

$\therefore \delta((C, X), 0) = (C, Y)$

$$\delta(X, 0) = Y$$

$$\delta(C, 1) = C$$

$\therefore \delta((C, X), 1) = (C, X)$

$$\delta(X, 1) = X$$

As no new state is getting generated we will stop applying new transitions and construct transition graph.

Step 3 :

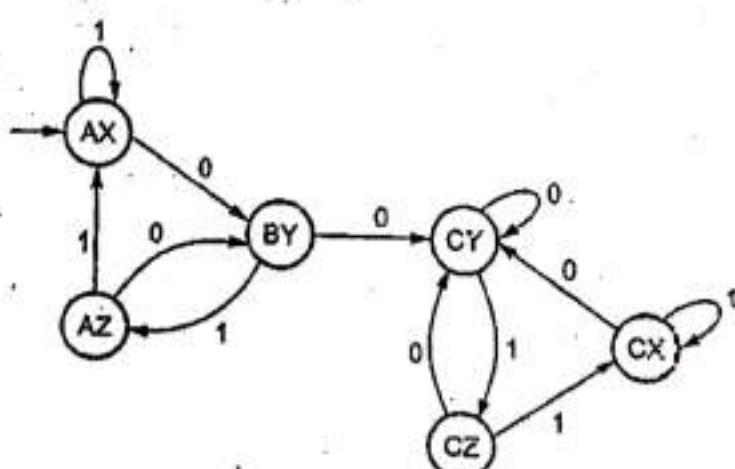
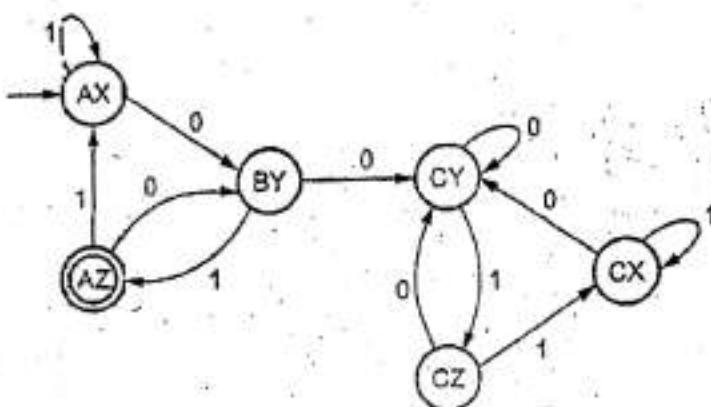


Fig. 2.15.1 Combined Transition Graph

Step 4 :

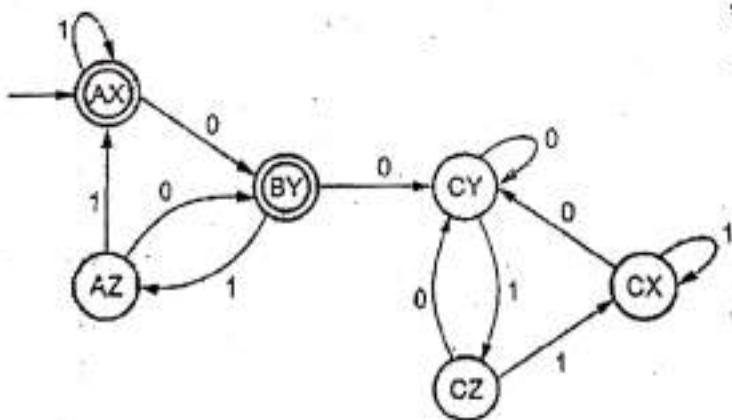
Now we will generate FAs for $L_1 \cap L_2$ from the transition graph in step 3. The $L_1 \cap L_2$ means the strings that are accepted by both L_1 and L_2 . Hence we will mark state AZ as the only final state because state A and state Z both are final states of L_1 and L_2 .

Also note that the state AZ accepts the strings not containing 00 as substring and ending with 01.

Fig. 2.15.2 $L_1 \cap L_2$ **Step 5 :**

We will generate FA $L_1 - L_2$. From the transition graph in step 3. The $L_1 - L_2$ means the strings that are accepted by L_1 but (not accepted) rejected by L_2 .

Hence the final states in following transition diagrams will be the accept states of L_1 that are combined with non-accept states of L_2 . Such states are AX and BY :

Fig. 2.15.3 $L_1 - L_2$

→ Example 2.15.2 : Draw Finite Automata (FA) for following languages :

$$L_1 = \{x / 11 \text{ is not a substring of } x, x \in \{0,1\}^*\}, \quad L_2 = \{x / x \text{ ends with } 10, x \in \{0,1\}^*\}$$

Find FA accepting languages i) $L_1 \cap L_2$ and ii) $L_1 - L_2$ GTU : Winter-2012, Marks 8

Solution : Refer example 2.13.12 and replace 0 by 1 and 1 by 0. Hence

i) $L_1 \cap L_2$ will be -

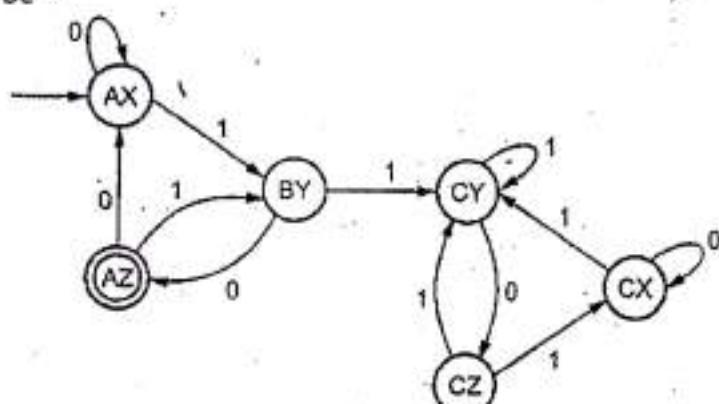


Fig. 2.15.4

ii) $L_1 - L_2$

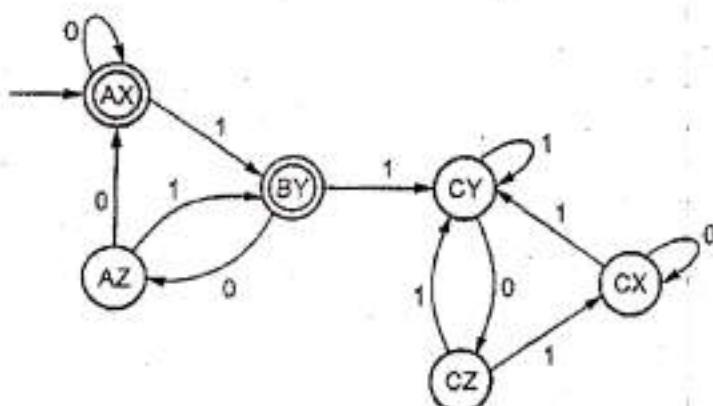


Fig. 2.15.5

→ Example 2.15.3 : Show that for any language $L, L^* = (L^*)^* = (L^+)^* = (L^s)^*$

GTU : Summer-13, Marks 7

Solution : The Kleen Closure L^* of L is the infinite union.

$$L^* = L^0 \cup L^1 \cup L^2 \cup \dots \cup L^n \cup \dots$$

$$\therefore (L^*)^* = (L^0 \cup L^1 \cup L^2 \cup \dots \cup L^n \cup \dots) \cup (L^0 \cup L^1 \cup L^2 \cup \dots \cup L^n \cup \dots)$$

$$(L^*)^* = (L^0 \cup L^1 \cup L^2 \cup \dots \cup L^n \cup \dots) \quad \dots (1)$$

$$\text{The} \quad L^+ = L^1 \cup L^2 \cup L^3 \cup \dots \cup L^n \cup \dots$$

$$(L^+)^* = \{L^0\} \text{ i.e. } \in U \{L^1 U L^2 U L^3 U \dots L^n U \dots\}$$

$$(L^+)^* = (L^0 U L^1 U L^2 U L^3 U \dots L^n U \dots) \quad \dots(2)$$

The $(L^+)^* = (L^0 U L^1 U L^2 U \dots L^n U \dots) U (L^1 U L^2 U \dots L^n U \dots)$

$$(L^+)^* = (L^0 U L^1 U L^2 U L^3 U \dots L^n U \dots) \quad \dots(3)$$

From equation (1), (2) and (3) we conclude that

$$L^* = (L^*)^* = (L^+)^* = (L^+)^*$$

Example 2.15.4 : Let M_1 and M_2 be the FA in Fig. 2.15.6 below for the language L_1 and L_2 , find $L_1 \cup L_2$ and $L_1 \cap L_2$. GTU : Summer-13, Marks 7

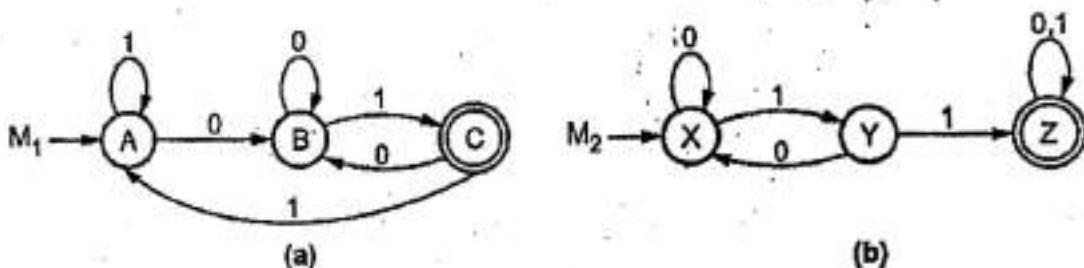


Fig. 2.15.6

Solution : The language L_1 represents that all the strings end with 01. The language L_2 contains at least one pair of 11.

$\therefore L_1 \cup L_2 = \{\text{The language that ends either 01 and contains at least one pair of 11}\}$

$\{L_1 \cap L_2\} = \{\text{The language that contains (10) as a substring}\}$

Example 2.15.5 : Let M_1 and M_2 be the FAs pictured below, recognizing languages L_1 and L_2 respectively. GTU : Summer-14, Marks 7

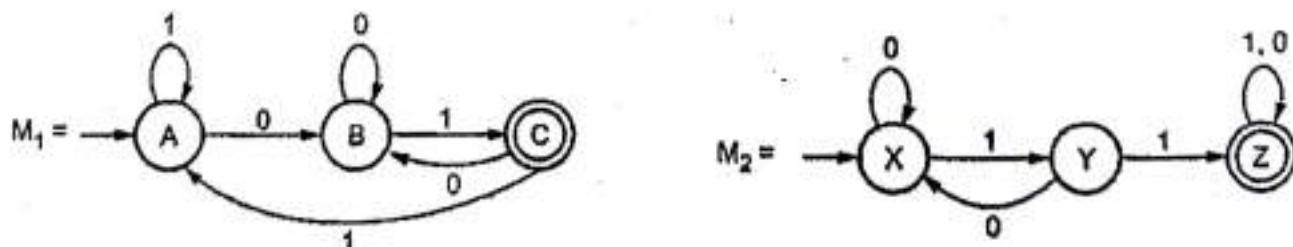


Fig. 2.15.7

Draw the FAs recognizing the following languages.

$$L_1 \cap L_2$$

$$L_2 - L_1$$

Solution :

Step 1 : We will apply δ transitions to design combined transition diagram.

$$\delta(A, 0) = B \quad \therefore \quad \delta((A, X), 0) = (B, X) \text{ New state}$$

$$\delta(X, 0) = X$$

$$\delta(A, 1) = A$$

$$\delta(X, 1) = Y \quad \therefore \quad \delta((A, X), 1) = (A, Y) \text{ New state}$$

\therefore Apply δ transitions on (B, X) and (A, Y)

$$\delta(B, 0) = B \quad \therefore \quad \delta((B, X), 0) = (B, X)$$

$$\delta(X, 0) = X$$

$$\delta(B, 1) = C \quad \therefore \quad \delta((B, X), 1) = (C, Y) \text{ New state}$$

$$\delta(X, 1) = Y$$

Consider $\delta(A, Y)$

$$\delta(A, 0) = B \quad \therefore \quad \delta((A, Y), 0) = (B, X)$$

$$\delta(Y, 0) = X$$

$$\delta(A, 1) = A \quad \therefore \quad \delta((A, Y), 1) = (A, Z) \text{ New state}$$

$$\delta(Y, 1) = Z$$

Consider $\delta(C, Y)$

$$\delta(C, 0) = B \quad \therefore \quad \delta((C, Y), 0) = (B, X)$$

$$\delta(Y, 0) = X$$

$$\delta(C, 1) = A \quad \therefore \quad \delta((C, Y), 1) = (A, Z)$$

$$\delta(Y, 1) = Z$$

Consider $\delta(A, Z)$

$$\delta(A, 0) = B \quad \therefore \quad \delta((A, Z), 0) = (B, Z) \text{ New state}$$

$$\delta(Z, 0) = Z$$

$$\delta(A, 1) = A \quad \therefore \quad \delta((A, Z), 1) = (A, Z)$$

$$\delta(Z, 1) = Z$$

Consider $\delta(B, Z)$

$$\delta(B, 0) = B$$

$$\therefore \delta((B, Z), 0) = (B, Z)$$

$$\delta(Z, 0) = Z$$

$$\delta(B, 1) = C$$

$$\delta(Z, 1) = Z$$

$\delta((B, Z), 1) = (C, Z)$ New state

Consider $\delta(C, Z)$

$$\delta(C, 0) = B$$

$$\delta((C, Z), 0) = (B, Z)$$

$$\delta(Z, 0) = Z$$

$$\delta((C, Z), 1) = (A, Z)$$

$$\delta(Z, 1) = Z$$

As no new state is getting generated the combined transition diagram from above computations will be -

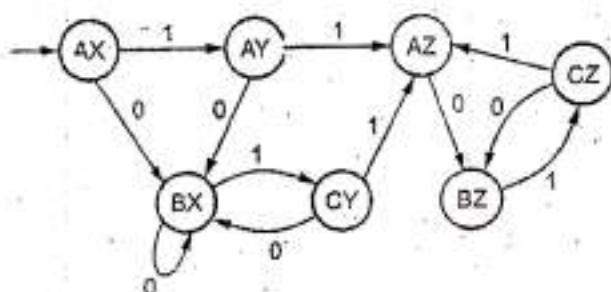


Fig. 2.15.8 Combined transition diagram

i) $L_1 \cap L_2$ means the states that are final in both M_1 and M_2 . The CZ is only such state. Hence FA for $L_1 \cap L_2$ is

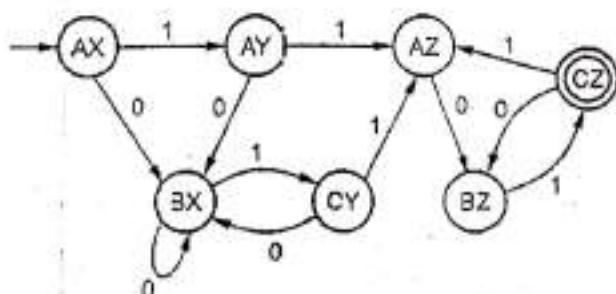
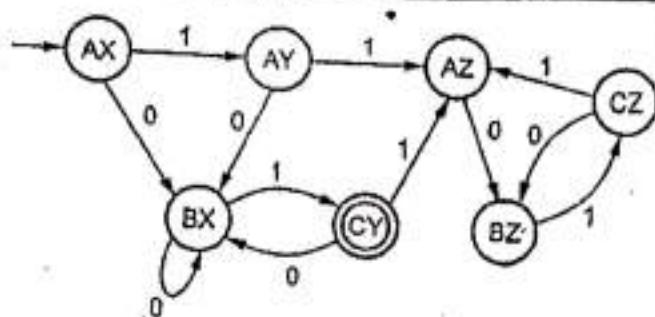


Fig. 2.15.9 $L_1 \cap L_2$

ii) $L_1 - L_2$ means the final state of M_1 and nonfinal state of M_2 . The only such state is CY.

Fig. 2.15.10 $L_1 - L_2$

Example 2.15.6 : Let M_1 , M_2 and M_3 be the FAs pictured in Fig. 2.15.11, recognizing languages L_1 , L_2 and L_3 respectively.

GTU : Summer-15, Marks 15, Summer-17, 18, Marks 7

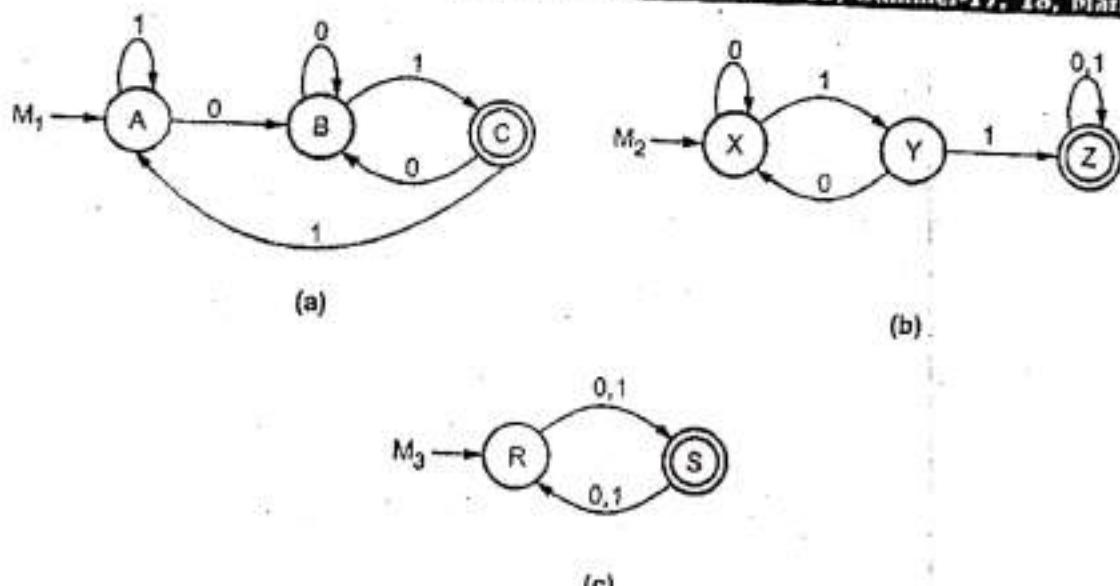


Fig. 2.15.11

Draw FAs recognizing the following languages.

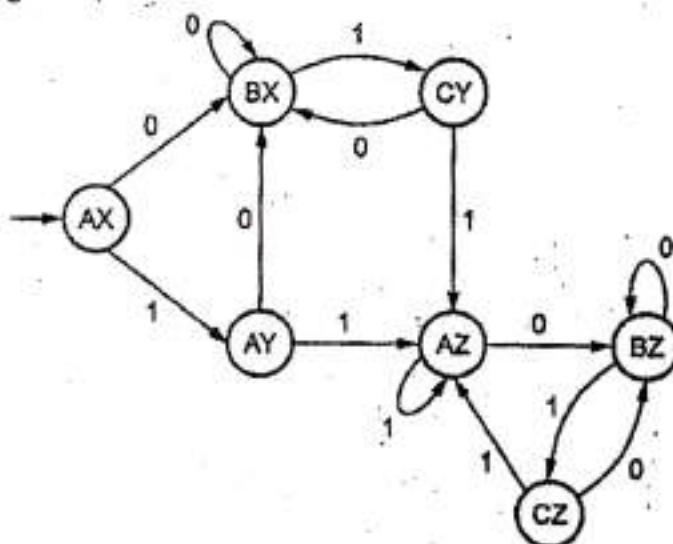
- $L_1 \cup L_2$
- $L_1 \cap L_2$
- $L_1 - L_2$
- $L_1 \cap L_3$
- $L_3 - L_2$

Solution : Step 1 : We will calculate δ transitions for input 0 and 1 both for each states of M_1 and M_2

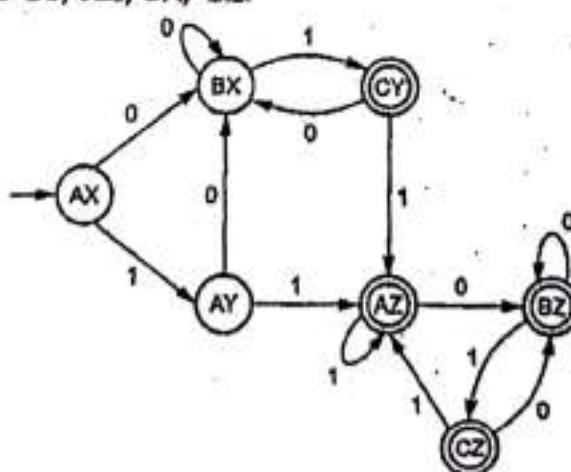
$$\begin{array}{ll} \delta((A, X), 0) = (B, X) & \text{New state} \\ \delta((A, X), 1) = (A, Y) & \text{New state} \\ \delta((B, X), 0) = (B, X) & \\ \delta((B, X), 1) = (C, Y) & \text{New state} \end{array}$$

- $\delta((A, Y), 0) = (B, X)$
 $\delta((A, Y), 1) = (A, Z)$ New state
 $\delta((C, Y), 0) = (B, X)$
 $\delta((C, Y), 1) = (A, Z)$
 $\delta((A, Z), 0) = (B, Z)$ New state
 $\delta((A, Z), 1) = (A, Z)$
 $\delta((B, Z), 0) = (B, Z)$
 $\delta((B, Z), 1) = (C, Z)$ New state
 $\delta((C, Z), 0) = (B, Z)$
 $\delta((C, Z), 1) = (A, Z)$

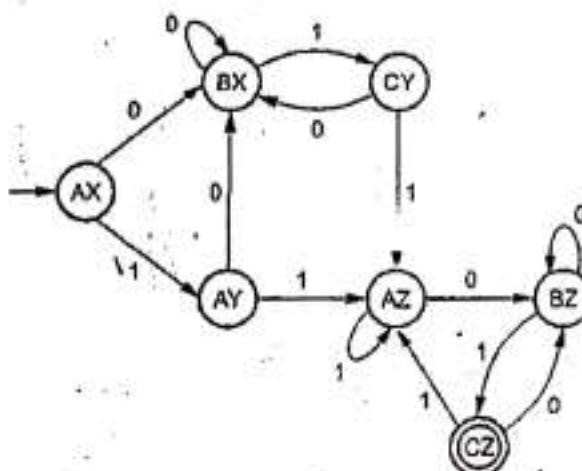
The transition diagram will be



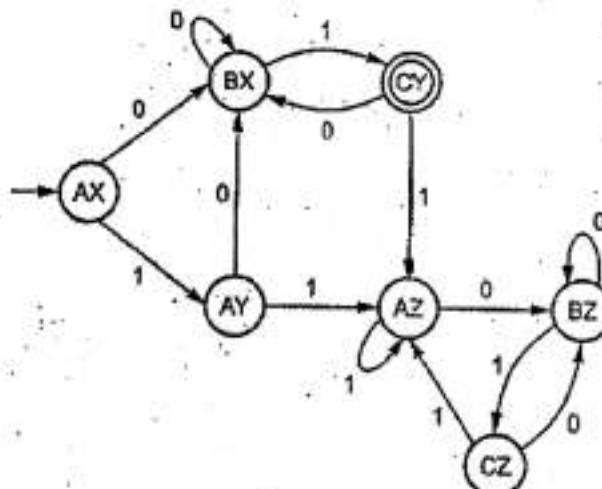
- i) $L_1 \cup L_2$ indicates the states C and Z combined with any state are final state
 \therefore Final states are CY, AZ, BX, CZ.



- ii) $L_1 \cap L_2$ indicates that the states which are final in both M_1 and M_2 . The only such state is CZ.



- iii) $L_1 - L_2$ indicates the final state in M_1 but non final state of M_2 . The only such state is CY.



Step 2 : We will draw the combined transition diagram for M_1 and M_2

$\delta((A, R), 0) = (B, S)$	New state
$\delta((A, R), 1) = (A, S)$	New state
$\delta((B, S), 0) = (B, R)$	New state
$\delta((B, S), 1) = (C, R)$	New state
$\delta((A, S), 0) = (B, R)$	
$\delta((A, S), 1) = (A, R)$	
$\delta((B, R), 0) = (B, S)$	
$\delta((B, R), 1) = (C, S)$	New state

$$\delta((C, R), 0) = (B, S)$$

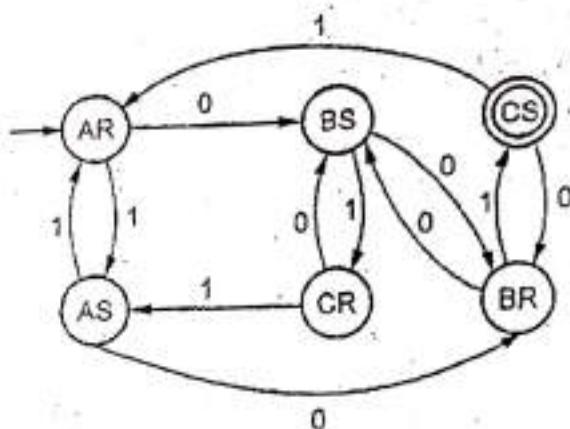
$$\delta((C, R), 1) = (A, S)$$

$$\delta((C, S), 0) = (B, R)$$

$$\delta((C, S), 1) = (A, R)$$

As no new state is getting generated the combined transition graph can be drawn using above transitions.

$L_1 \cap L_3$ means the final states of both the M_1 and M_3 . The CS is the state in which C is the final $\in M_1$ S is final state $\in M_2$

Fig. 2.15.12 (a) $L_1 \cap L_3$

Step 3 : We will draw the combined transition diagram for M_3 and M_2

$$\delta((R, X), 0) = (S, X) \quad \text{New state}$$

$$\delta((R, X), 1) = (S, Y) \quad \text{New state}$$

$$\delta((S, X), 0) = (R, X)$$

$$\delta((S, X), 1) = (R, Y) \quad \text{New state}$$

$$\delta((S, Y), 0) = (R, X)$$

$$\delta((S, Y), 1) = (R, Z) \quad \text{New state}$$

$$\delta((R, Y), 0) = (S, X)$$

$$\delta((R, Y), 1) = (S, Z) \quad \text{New state}$$

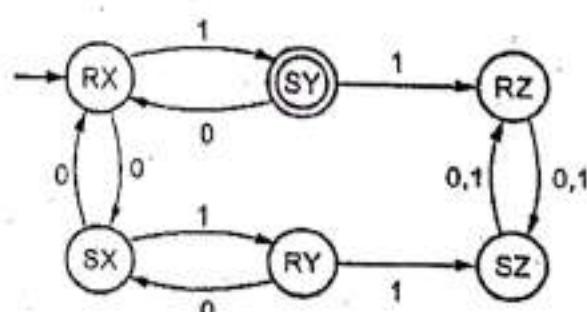
$$\delta((R, Z), 0) = (S, Z)$$

$$\delta((R, Z), 1) = (S, Z)$$

$$\delta((S, Z), 0) = (R, Z)$$

$$\delta((S, Z), 1) = (R, Z)$$

As no new state is getting generated the transition diagram can be

Fig. 2.15.12 (b) $L_3 - L_2$

$L_3 - L_2$ means accept state of M_3 and non accept state of M_2 . The only such state is SY. Hence SY is final state.

Example 2.15.7 : There are 2 languages over $\Sigma = \{a, b\}$

$L_1 = \text{all strings with a double "a"} \quad L_2 = \text{all strings with an even number of "a"}$

Find a regular expression and an FA that define $L_1 \cap L_2$. GTU : Winter-16, Marks 7

Solution : We will construct a combined transition graph considering both the FA's for input a and b transitions. The process of creating a combined transition graph is as follows :

Step 1 :

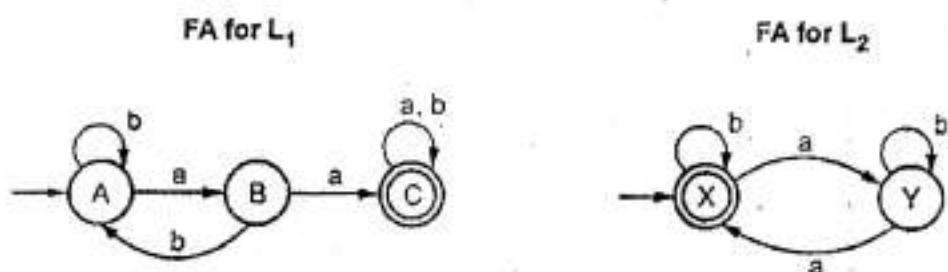


Fig. 2.15.13

Step 2 : We will compute δ transitions

$$\delta(A, a) = B \quad \therefore \delta((A, X), a) = (B, Y) \text{ New state}$$

$$\delta(X, a) = Y$$

$$\delta(A, b) = A$$

$$\delta(X, b) = X \quad \therefore \delta((A, X), b) = (A, X)$$

Now we will obtain δ transition on (B, Y) . The δ transition will be

$$\delta(B, a) = C \quad \therefore \delta((B, Y), a) = (C, X)$$

$$\delta(Y, a) = X$$

$$\delta(B, b) = A$$

$$\delta(Y, b) = Y \quad \therefore \delta((B, Y), b) = (A, Y)$$

Consider $\delta(A, X)$

$$\delta(A, a) = B \quad \therefore \delta((A, X), a) = (B, Y)$$

$$\delta(X, a) = Y$$

$$\delta(A, b) = A$$

$$\delta(X, b) = X \quad \therefore \delta((A, X), b) = (A, X)$$

Consider $\delta(C, X)$

$$\delta(C, a) = C$$

$$\delta(X, a) = Y$$

$$\delta(C, b) = C$$

$$\delta(X, b) = X$$

Consider $\delta(A, Y)$

$$\delta(A, a) = B \therefore \delta((A, Y), a) = (B, X)$$

$$\delta(Y, a) = X$$

$$\delta(A, b) = A \therefore \delta((A, Y), b) = (A, Y)$$

$$\delta(Y, b) = Y$$

Consider $\delta(B, X)$

$$\delta(B, a) = C \therefore \delta((B, X), a) = (C, Y)$$

$$\delta(X, a) = Y$$

$$\delta(B, b) = A \therefore \delta((B, X), b) = (A, X)$$

$$\delta(X, b) = X$$

Consider $\delta(C, Y)$

$$\delta(C, a) = C \therefore \delta((C, Y), a) = (C, X)$$

$$\delta(Y, a) = X$$

$$\delta(C, b) = C \therefore \delta((C, Y), b) = (C, Y)$$

$$\delta(Y, b) = Y$$

As no new state is getting generated, we will stop applying new transitions and construct transition graph.

As both C and X are final states the state CX is final state.

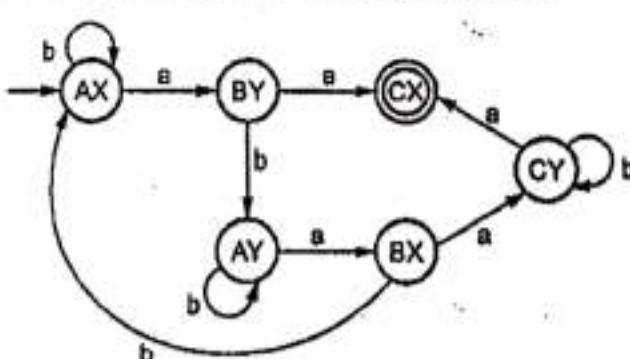


Fig. 2.15.13 (a)

Example 2.15.8 : Let FA_1 and FA_2 be the FAs as shown in Fig. 2.15.14 recognizing the languages L_1 and L_2 respectively. Draw an FA recognizing the language, $L_1 \cup L_2$.

GTU : Winter-19, Marks 3

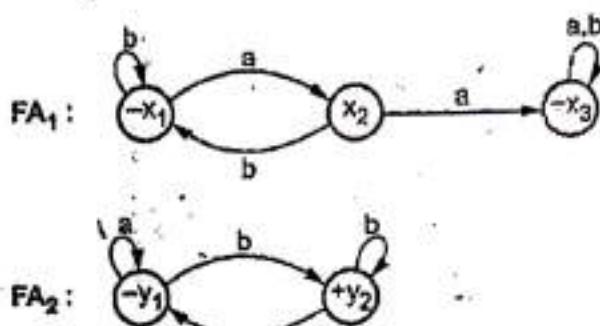


Fig. 2.15.14

Solution :

Step 1 : We will apply δ transitions to design combined transition diagram.

$$\delta(x_1, a) = x_2$$

$$\delta(y_1, a) = y_1$$

$$\delta(x_1, b) = x_1$$

$$\delta(y_1, b) = y_2$$

$$\therefore \delta((x, y_1), a) = (x_2, y_1)$$

$$\therefore \delta((x, y_1), b) = (x_1, y_2)$$

Now apply δ transitions on (x_2, y_1) and (x_1, y_2) .

$$\delta(x_1, a) = x_2$$

$$\delta(y_2, a) = y_1$$

$$\delta(x_1, b) = x_1$$

$$\delta(y_2, b) = y_2$$

$$\delta(x_2, a) = x_3$$

$$\therefore \delta((x_1, y_2), a) = (x_2, y_1)$$

$$\therefore \delta((x_1, y_2), b) = (x_1, y_2)$$

$$\delta(y_1, a) = y_1$$

$$\therefore \delta((x_2, y_1), a) = (x_3, y_1)$$

New state

$$\delta(x_2, b) = x_1$$

$$\delta(y_1, b) = y_2$$

$$\therefore \delta((x_2, y_1), b) = (x_1, y_2)$$

Now consider $\delta(x_3, y_1)$

$$\delta(x_3, a) = x_3$$

$$\delta(y_1, a) = y_1$$

$$\therefore \delta((x_3, y_1), a) = (x_3, y_1)$$

$$\delta(x_3, b) = x_3$$

$$\delta(y_1, b) = y_2$$

$$\therefore \delta(x_3, y_1) = (x_3, y_2)$$

New state

Consider $\delta(x_3, y_2)$

$$\delta(x_3, a) = x_3$$

$$\therefore \delta(x_3, y_2) = (x_3, y_1)$$

$$\delta(y_2, a) = y_1$$

$$\delta(x_3, b) = x_3$$

$$\therefore \delta(x_3, y_2) = (x_3, y_2)$$

$$\delta(y_2, b) = y_2$$

As no new state getting generated the combined transition diagram is

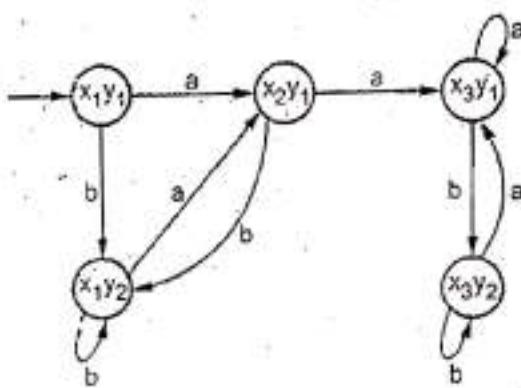


Fig. 2.15.15

$L_1 \cup L_2$ means states x_3 and y_2 combined with any state are final states. Hence $\{(x_3, y_2), (x_3, y_1), (x_1, y_2)\}$ are final states.

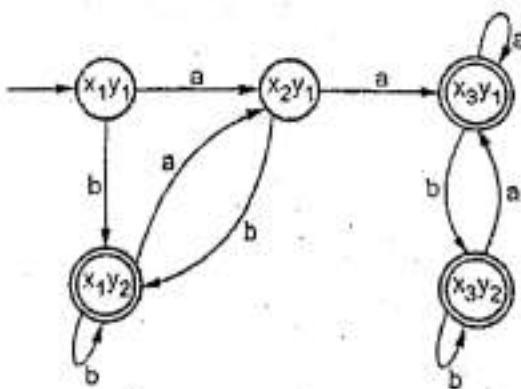


Fig. 2.15.16

Review Question

- What are the closure properties of regular languages.

GTU : Winter-18, Marks 3

2.16 Pumping Lemma

This is a basic and important theorem used for checking whether given string is accepted by regular expression or not. In short, this lemma tells us whether given language is regular or not.

One key theme is that any language for which it is possible to design the finite automata is definitely the regular language.

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

Proof : If the language L is regular it is accepted by a DFA, $M = (Q, \Sigma, \delta, q_0, F)$. With some particular number of states say, n . Consider the input can be $a_1, a_2, a_3, \dots, a_m$, $m \geq n$. The mapping function δ could be written as $\delta(q_0, q_1, q_2, q_3, \dots, q_i) = q_i$.

The transition diagram is as shown in Fig. 2.16.1.

If q_m is in F i.e. $q_1, q_2, q_3, \dots, q_m$ is in $L(M)$ then $a_1, a_2, \dots, a_j, a_{k+1}, a_{k+2}, \dots, a_m$ is also in $L(M)$. Since there is path from q_0 to q_m that goes through q_j but not around the loop labelled $a_{j+1} \dots a_k$. Thus

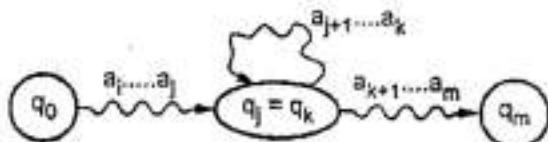


Fig. 2.16.1 Pumping lemma

$$\delta(q_0, a_1, a_2, \dots, a_j, a_{k+1}, \dots, a_m) = \delta(\delta(q_0, q_1, \dots, q_j), a_{k+1}, \dots, a_m)$$

$$= \delta(q_j, q_{k+1}, \dots, q_m)$$

$$= \delta(q_k, q_{k+1}, \dots, q_m)$$

$$= q_m$$

That is what we have proved i.e. given any long string can be accepted by FA, we should be able to find a substring near the beginning of the string that may be pumped i.e. repeated as many times as we like and resulting string may be accepted by FA.

The pumping lemma is used to check whether given language is regular or not.

The pumping lemma is used to check whether given language is regular or not. We will discuss it with the help of some examples.

► **Example 2.16.1 :** Show that the set $L = \{b^{i^2} \mid i > 1\}$ is not regular.

Solution : We have to prove that the language $L = b^{i^2}$ is not regular. This language is such that number of b 's is always a perfect square.

For example, if we take $i = 1$

$$L = b^{1^2} = b \quad \text{the length} = 1^2$$

$$= b^{2^2} = bbbb \quad \text{length} = 2^2$$

and so on.

Now let us consider

$$L = b^{n^2} \text{ where } \text{length} = n^2$$

it is denoted by z .

$$|z| = n^2$$

By pumping lemma $z = u v w$

where $1 \leq |v| \leq n$

As $z = u v^i w$ where $i = 1$

Now we will pump v i.e. make $i = z$.

As we made $i = 2$ we have added one n^2 .

$1 \leq |v| \leq n$.

$$n^2 + 1 \leq |uvw| \leq n + n^2$$

$$\text{i.e. } n^2 + 1 \leq |uvw| \leq n^2 + n + n + 1$$

$$\text{i.e. } n^2 + 1 \leq |uvw| \leq (n+1)^2$$

$$= n^2 \leq |uvw| \leq (n+1)^2$$

Thus the string lies between two consecutive perfect squares. But the string is not a perfect square. Hence we can say the given language is not regular.

For example,

$$L = b^{i^2}$$

$$\text{Let } i = 2$$

$$L = bbbb$$

$$L = uvw$$

$$\text{Assume } uvw = bbbb$$

Take

$$u = b$$

$$v = bb$$

$$w = b$$

By pumping lemma, even if we pump v i.e. increase v then language should show the length as perfect square.

$$\begin{aligned} & uvw \\ &= uv.vw \\ &= bbbbbb \\ &= \text{length of } b \text{ is not a perfect square} \end{aligned}$$

Thus the behaviour of the language is not regular, as after pumping something onto it does not show the same property (being square for this example.)

Example 2.16.2 : Define pumping lemma for regular languages.

Prove that the language $L = \{a^n : n \text{ is a prime number}\}$ is not regular.

GTU : Winter-11, Marks 5

Solution : Let us assume L is a regular and n is a prime number.

$$L = a^n$$

$$|z| = uvw \quad i = 1$$

Now consider $L = uv^i w$ where $i = 2$

$$= uv.vw$$

Adding 1 to n we get,

$$n < |uvvw|$$

$$n < n+1$$

But $n + 1$ is not a prime number. Hence what we have assumed becomes contradictory. Thus L behaves as it is not a regular language.

Example 2.16.3 : Show that $L = \{0^n 1^{n+1} \mid n > 0\}$ is not regular.

Solution : Let us assume that L is a regular language.

$$\begin{aligned} |z| &= |uvw| \\ &= 0^n 1^{n+1} \end{aligned}$$

Length of string $|z| = n + n + 1 = 2n + 1$.

That means length is always odd.

By pumping lemma

$$= |uv.vw|$$

That is if we add $2n + 1$

$$2n + 1 < (2n + 1) + 2n + 1$$

$$2n + 1 < 4n + 2$$

But if $n = 1$ then we obtain $4n + 2 = 6$ which is no way odd. Hence the language becomes irregular.

Even if we add 1 to the length of $|z|$, then

$$|z| = 2n + 1 + 1 = 2n + 2$$

= even length of the string.

So this is not a regular language.

The simple way to know whether given language is regular or not is that try to draw finite automata for it, if you can easily draw the FA for the given L then that language is surely the regular otherwise not.

Example 2.16.4 : Define pumping lemma. Use the pumping lemma to show that the following languages are not regular.

i) $L = 0^n 1 0^{2n} / n \geq 0$

ii) $L = 0^i 1^j 0^k / k > i+j$

GTU : Summer-12, Marks 7

Solution. : Pumping Lemma : Refer section 2.16.

i) Let us assume that L is a regular language.

$$\begin{aligned}|z| &= |uvw| \\&= 0^n 1 0^{2n} | n \geq 0\end{aligned}$$

Length of string $|z| = n+1+2n = 3n+1$.

Thus we get the alternate strings that are of odd length.

By Pumping Lemma,

$$|z| = |uv.vw|$$

That is we add $3n+1$ then

$$3n+1 < (3n+1)+3n+1$$

$$3n+1 < 6n+2$$

By this pumping of strings we get all the strings that are of even length. Hence the language becomes irregular.

Thus the language is not regular.

ii) Let us assume that $L = 0^i 1^j 0^k | k > i+j$ be a regular language.

Then by Pumping Lemma,

$$|z| = |uvw| \in L$$

$$z = 0^i 1^j 0^k$$

Then after applying Pumping Lemma, following cases are possible.

Case 1 :

$$\text{Let, } z = uv^t w$$

$$z = 01000 \quad \text{where } i = 1, j = 1 \text{ and } k = 3.$$

If we put $t = 2$ then if we assume

$$z = \underset{u}{0} \underset{v}{1} \underset{w}{\underbrace{000}} \quad \text{then}$$

$$z = uv^t w \quad \text{becomes}$$

$$= uvww$$

$$z = 011000 \notin L \text{ because } i+j = k.$$

Case 2 :

$$\text{Let } z = \underset{u}{0} \underset{v}{01} \underset{w}{\underbrace{11000000}}$$

If $t = 2$ then,

$$z = uvww$$

$$z = 0010111000000$$

$$z = 0^n 1 0^{n+1} 0^m \notin L \text{ clearly.}$$

From above two cases we can clearly note that the behaviour of language L is not regular. This ultimately proves that L is not regular.

→ **Example 2.16.5 :** Use the pumping lemma to show that following language is not regular
 $L = \{xy | x, y \in \{0,1\}^* \text{ and } y \text{ is either } x \text{ or } x^r\}$ GTU : Summer-13, 17, Marks 7

Solution : Let, $L = \{xy | x, y \in \{0,1\}^* \text{ and } y \text{ is either } x \text{ or } x^r\}$

Assume $y = x^r$

$$\{L = xy \mid y \text{ is } x^r\}$$

We assume $S = 0101.$

$$S^r = 1010.$$

Hence $L = 01011010.$

For splitting into x, y, z parts such that

$$w = xyz \mid y \mid > 1 \text{ and } |xy| < n.$$

Let

$$\begin{array}{ccccccc} S & = & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ & & \downarrow & \downarrow & & & \boxed{1} & & \\ & & = & x & y & & z & & \end{array}$$

Assume

$$|x| = n - 1$$

$$|y| = 1$$

$$|xy| = |x| + |y|$$

$$= (n-1) + 1$$

$$|xy| = n$$

According pumping Lemma,

When $w = x y^i z \in L$ then language L is said to be regular. We assume $L = xx^r$ is regular. We will find $w = xy^i z$.

$$w = 01011010$$

from equation 1.

If

$$i = 0 \text{ then}$$

$$w = xz$$

$$w = 0011010$$

$$\neq xx^r$$

Hence $w \notin L$.

If

$$i = 2 \text{ then}$$

$$w = xyyz$$

$$= 011011010.$$

$$\neq xx^r$$

$w \notin L$.

This shows that the given language is not regular.

Example 2.16.6 : Are the following sets regular ? (Using Pumping Lemma) Prove the same.

- $\{a^n b^{2n} / n > 0\}$
- $\{a^n b^m / 0 < n < m\}$
- $\{a^n b^{2m} / m, n > 0\}$.

Solution : i) Let, $L = \{a^n b^{2n} / n > 0\}$ be a regular language.

Then by pumping lemma,

$$|z| = |uvw| \in L$$

$$\therefore \text{If } z = a^n b^{2n}$$

$$|z| = n + 2n = 3n$$

That means length is always multiple of 3.

By pumping lemma,

$$|u.v.v.w| = 3n + 1$$

That is we do not get the length of resultant string in multiple of 3. This shows that $z \notin L$. This is not a regular language. Hence our assumption of L being regular is wrong so the given language.

$\{L = a^n b^{2n}\}$ is not regular.

ii) Let us assume that

$L = \{a^n b^{2m} / 0 < n < m\}$ is a regular language. That means

$$|z| = |uvw| \in L$$

If we assume string z as

$z = a^n b^{2m}$ then after applying pumping lemma following cases are possible.

Case 1 :

$$z = uv^iw = a^n b^{2m} = aabbbaaa when n = 2 and m = 3 as n < m.$$

Let $i = 2$ then if we assume

$$z = \underbrace{aa}_u \underbrace{bbb}_v \underbrace{bbb}_w$$

If $z = uvvw$ then

$$\begin{aligned} z &= aa bbb bbb bbb \\ &= a^2 b^9 \end{aligned}$$

Here $9 \neq 2m$. That means this language $\notin L$.

Case 2 : Let

$z = aa \underline{bbb} bbb$ with $n = 2, m = 3$ as $n < m$

If by pumping lemma

$z = \underline{aa} \underline{bbb} bbb$

If $z = uv^iw^v$ with $i = 0$ then

$z = uw$

i.e. $z = aa \underline{bbb}$

$$= a^2 b^3 \neq a^n b^{2m}$$

That means given language is not regular.

From above two cases it is proved that

$L = a^n b^{2m}$ is not regular.

iii) $L = \{a^n b^{2m} \mid n, m > 0\}$ be a regular language is our assumption. Then by pumping lemma,

$|z| = |uvw| \in L$ and if the string

$z = uv^iw$ then it should belong to L . Hence it is said to be regular. If we assume

$z = a^2 b^6$ assuming $n = 2$ and $m = 3$

Then consider various cases as follows.

Case 1 : Let $n = 2, m = 3$ then

$z = \underline{aa} \underline{bbb} bbb$

If $i = 2$ then $z = uv^2w$ becomes

$z = uv^2w$ then

$z = aa \underline{bbb} bbb bbb$

$$z = a^2 b^9 \neq a^n b^{2m} \notin L$$

Hence given language is not regular.

Case 2 : Let $n = 2, m = 1$ and $i = 0$

then $z = uv^iw$ becomes

$z = uw$ and the string

$z = a^2 b^2$ i.e.

$= \underline{aa} \underline{b} \underline{b}$ becomes

$z = aab$

$$z = a^2 b^1 \neq a^n b^{2m} \notin L$$

That means given language is not regular. From these cases it is proved that given language L is not regular.

⇒ **Example 2.16.7 :** Use pumping lemma to show whether or not the following languages are regular :

$$i) A_1 = \{www \mid w \in (a, b)^*\} \quad ii) A_2 = \{a^{2n} \mid n \geq 0\}.$$

Solution : i) Let,

$A_1 = \{www \mid w \in (a, b)^*\}$ is assumed to be a regular language. Then by pumping lemma the string $w = xyz \in A_1$ such that

$$w = xyz$$

$$\text{Here } |w| = , |www| = 3n \text{ where } n > 0$$

By pumping lemma if

$$w = xyz$$

$$= www$$

$$|w| = > 3n$$

That means $w \notin L$. This shows that given language is not regular. Hence our assumption is wrong.

ii) $A_2 = \{a^{2n} \mid n \geq 0\}$ is a language in which length of string is always even.

i.e. If $n = 1$ then $A_2 = aa$

If $n = 2$ then $A_2 = aaaa$ and so on

If $w = xyz$ then

$$|w| = a^{2n} = xy^i z$$

Then $|w| > 2n$. That means we may get a string of odd length. Hence given language is not regular.

⇒ **Example 2.16.8 :** Define Pumping Lemma for Regular Languages. Use Pumping Lemma to show that following languages are not regular.

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

$$L = \{WW^R \mid w \in \{0, 1\}^*\}$$

GTU : Summer-15, Marks 7

Solution : i) Refer example 2.16.6 (i).

ii) Refer example 2.16.3.

⇒ **Example 2.16.9 :** If $L = \{0^i 1^i \mid i \geq 0\}$ Prove that L is regular. **GTU : Winter-16, Marks 7**

Solution : Let, $L = \{0^i 1^i \mid i \geq 0\}$

Then by pumping lemma

$$|z| = |uvw| \in L$$

If the string is $z = uv^i w$ then $\in L$

Then it is said to be regular

If we assume $i = 2$, then consider various cases as follows

Case 1 :

$$z = \begin{array}{c} 0 \ 0 \ 1 \ 1 \\ \downarrow \quad \downarrow \\ u \quad v \end{array}$$

with $i = 2$ $z = uv^i w$ becomes

$$\begin{aligned} z &= uvvw \\ &= 001011 \\ &= 0^2 1 0 1^2 \notin L \end{aligned}$$

Hence given language is not regular

Case 2 :

$$z = \begin{array}{c} 0 \ 0 \ 1 \ 1 \\ \downarrow \quad \downarrow \quad \downarrow \\ u \quad v \quad w \end{array}$$

with $i = 2$ $z = uv^i w$ becomes

$$\begin{aligned} z &= uvvw \\ &= 00111 \\ &= 0^2 1^3 \notin L \end{aligned}$$

Hence given language is not regular.

Thus from above cases, clearly given language is not regular.

⇒ **Example 2.16.10 :** Using pumping lemma for CFL's, show that the language $L = \{a^m b^n c^n | m \leq n \leq 2m\}$ is not context free.

GTU : Summer-17, Marks 4

Solution : Let $L = \{a^m b^n c^n\}$ where $m \leq n \leq 2m$

Various strings that may belong to this language L can be
 $\{abcc, aabbccc, aabbcccc, \dots\}$

Let us apply pumping lemma by picking up a valid string say $aabbcccc$.

There are three conditions to be satisfied by L to be CFL using pumping lemma

If $w = pqrst$ with $w \in L$ and $|w| > n$

1) $qs \neq c$

2) $|qrs| \leq n$

3) For all $i \geq 0$, $pq^i rs^i t \in L$.

Let, $w = aabbccccc$

Here $|w| = 8$

Assume $w = \begin{array}{c} aa \\ | \\ pq \\ | \\ q \end{array} \begin{array}{c} b \\ | \\ qr \\ | \\ r \end{array} \begin{array}{c} b \\ | \\ s \\ | \\ t \end{array} cc \begin{array}{c} cc \\ | \\ t \end{array}$

i) $qs \neq \epsilon$, i.e. $bcc \neq \epsilon$ is satisfied.

ii) $|qrs|$ should be $\leq n$

$$|qrs| = 4 \leq 8 \text{ is satisfied}$$

iii) Assume $i = 2$ then

$$w = \begin{array}{c} aa \\ | \\ pq \\ | \\ q \end{array} \begin{array}{c} b \\ | \\ qr \\ | \\ r \end{array} \begin{array}{c} b \\ | \\ s \\ | \\ t \end{array} cc \begin{array}{c} cc \\ | \\ t \end{array} \text{ becomes}$$

$$\begin{aligned} w &= pq^i rs^i t = pq^2 rs^2 t \\ &= pqqrssst \\ \text{i.e. } &= aabbccccc \\ &= a^2 b^3 c^6 \notin L = a^m b^m c^n \end{aligned}$$

Thus the condition $w = pq^i rs^i t \in L$ is not satisfied. Hence given language is not CFL.

Example 2.16.11 : Prove that the language $L = \{a^n b^n c b^{n+1} | n = 1, 2, 3, \dots\}$ is non regular using pumping lemma.

GTU : Winter-19, Marks 4

Solution : Let us assume that L is regular language

$$|z| = |uvw| = a^n b^n a b^{n+1} | n = 1, 2, 3, \dots$$

$$|z| = n + n + 1 + n + 1 = 3n + 2$$

By pumping lemma,

$$|z| = |u \cdot v \cdot v \cdot w|$$

Suppose

$$w = a^n b^n \overbrace{a^{n+1}}^{\downarrow} \\ u \quad v \quad w$$

$$\text{Then, } |u \cdot v \cdot v \cdot w| = |a^n b^n b^n a b^{n+1}| \\ = n + n + n + 1 + n + 1 \\ = 4n + 2$$

This shows that $|u \cdot v \cdot v \cdot w| \notin L$

Hence our assumption is wrong.

The given language L is not regular.

Review Questions

1. Prove : Any regular language can be accepted by a finite automaton (Kleene's theorem, Part-I)

GTU : Summer-12, Marks 7

2. Explain Pumping Lemma and its applications.

GTU : Summer-16, Marks 7, Winter-18, Marks 3

2.17 Short Questions and Answers

- Q.1 _____ model is normally used to model the regular expression.

Ans. : DFA.

- Q.2 DFA and Simulate NFA(True/False)

Ans. : True

- Q.3 Regular expressions are used to represent which language

- | | | | |
|----------------------------|--------------------|----------------------------|-----------------------|
| <input type="checkbox"/> a | Recursive language | <input type="checkbox"/> b | Context free language |
| <input type="checkbox"/> c | Regular language | <input type="checkbox"/> d | All of these |

[Ans. : a]

- Q.4 The set of all strings over $\Sigma = \{a,b\}$ in which all strings of a's and b's ending in bb is

- | | | | |
|---------------------------------------|--------------|----------------------------|--------------|
| <input type="checkbox"/> a | ab | <input type="checkbox"/> b | a^*bbb |
| <input checked="" type="checkbox"/> c | $(a+b)^* bb$ | <input type="checkbox"/> d | All of these |

[Ans. : c]

- Q.5 Which of the following is not a regular expression ?

- | | | | |
|----------------------------|--------------|----------------------------|----------------------|
| <input type="checkbox"/> a | $(a+b)^*abb$ | <input type="checkbox"/> b | $abba^*$ |
| <input type="checkbox"/> c | $0 + 1$ | <input type="checkbox"/> d | $(a+b)^* \cdot (aa)$ |

[Ans. : d]

Q.6 Regular expressions are

- | | |
|--------------------------------------------|--------------------------------------------|
| <input type="checkbox"/> a Type 0 language | <input type="checkbox"/> b Type 1 language |
| <input type="checkbox"/> c Type 2 language | <input type="checkbox"/> d Type 3 language |

[Ans. : a]

Q.7 Regular expressions are closed under.

- | | |
|---------------------------------------|-----------------------------------------|
| <input type="checkbox"/> a union | <input type="checkbox"/> b intersection |
| <input type="checkbox"/> c kleen star | <input type="checkbox"/> d all of these |

[Ans. : d]

Q.8 Moore machine is an application of _____.

Ans. : Finite automata with output.

Q.9 During the representation of mealy machine where do we specify the output ?

Ans. : In Mealy machine representation the output is specified on the transition edge along with input.

Q.10 The DFA can be represented by _____.

- | | |
|---------------------------------------------|-----------------------------------------------|
| <input type="checkbox"/> a transition graph | <input type="checkbox"/> b transition diagram |
| <input type="checkbox"/> c both a and b | <input type="checkbox"/> d Programming code |

[Ans. : c]

Q.11 An automation is a _____ device and a grammar is a _____ device.

- | | |
|--------------------------------------------------|--------------------------------------------------|
| <input type="checkbox"/> a generative, cognitive | <input type="checkbox"/> b generative, acceptor |
| <input type="checkbox"/> c acceptor, cognitive | <input type="checkbox"/> d cognitive, generative |

[Ans. : d]

Q.12 What is the regular expression notation used for matching zero or more number of characters ?

- | | |
|-------------------------------|------------------------------|
| <input type="checkbox"/> a \$ | <input type="checkbox"/> b # |
| <input type="checkbox"/> c * | <input type="checkbox"/> d + |

[Ans. : c]

Q.13 You can use regular expressions in the Find what and Replace with strings to enhance your search (True/False)

Ans. : True

Q.14 What is the use of pumping lemma ?

Ans. : The pumping lemma is used to check whether the given language is regular or not.



Context Free Grammar

3.1 Definition of Context Free Grammar

The context free grammar can be formally defined as a set denoted by $G = (V, T, P, S)$ where V and T are set of non terminals and terminals respectively. P is set of production rules, where each production rule is in the form of

Non terminal \rightarrow Non terminals

or Non terminal \rightarrow Terminals

S is a start symbol.

For example,

$G = (V, T, P, S)$ where $V = \{S\}$, $T = \{+, *\}$, and S is start symbol

$P = \{ S \rightarrow S + S$

$S \rightarrow S * S$

$S \rightarrow (S)$

$S \rightarrow 4 \}$

If the language is $4 + 4 * 4$ then we can use the production rules given by P . The start symbol is S . The number of non terminals in the rules P is one and the only non terminal i.e. S . The terminals are $+$, $*$, $($, $)$ and 4 .

We are using following conventions.

1. The capital letters are used to denote the non terminals.
2. The lower case letters are used to denote the terminals.

Example : The formation of production rules for checking syntax of any English statement is

SENTENCE \rightarrow NOUN VERB

NOUN \rightarrow Rama | Sita | Gopal

VERB \rightarrow goes | writes | sings

Thus if want to derive a string "Rama sings" then we can follow the above rules.

The language generated by G is denoted by $L(G)$. The language L is called context free language CFL if $L(G)$ is for CFG.

For example

$$G = \{S, B\} \{a, b\}, (S \rightarrow a B b, S \rightarrow B \rightarrow bbb)$$

is a grammar. Then we can derive a string abbbb as -

i) We will first start from start symbol S

$$S \Rightarrow a B b$$

ii) Then we will replace B by bbb

$$S \Rightarrow a B b \Rightarrow a bbbb$$

3.2 Union, Concatenation and Kleen's CFG

Consider $G = (V, T, P, S)$ be a context free grammar then, we can derive a string w from it. This w can be obtained from $(VUT)^*$ where V denotes the set of non-terminal symbols and T denotes the set of terminal symbols. The derivation of w from start symbol S can be written as $S \xrightarrow{\text{lm}} w$ which is called as **sentential form**. If $S \xrightarrow{\text{rm}} w$ then w is a left-sentential form.

If $S \xrightarrow{\text{rm}} w$ then w is a right-sentential form. Here lm stands for leftmost derivation and rm stands for rightmost derivation.

For example :

Consider the grammar,

$$S \rightarrow S + S | S * S | 4$$

Then for deriving the string $w = 4 + 4 * 4$ we use above grammar as -

$$\begin{array}{ccccccc} S & \xrightarrow{\text{lm}} & S * S & \xrightarrow{\text{lm}} & S + S * S & \xrightarrow{\text{lm}} & 4 + S * S \\ & & & & & & \xrightarrow{\text{lm}} \\ & & & & & & 4 + 4 * S \\ & & & & & & \xrightarrow{\text{lm}} \\ & & & & & & 4 + 4 * 4 \end{array}$$

Here $S + S * S$ is called left-sentential form.

$$\begin{array}{ccccccc} S & \xrightarrow{\text{rm}} & S + S & \xrightarrow{\text{rm}} & S + S * S & \xrightarrow{\text{rm}} & S + S * 4 \\ & & & & & & \xrightarrow{\text{rm}} \\ & & & & & & 4 + 4 * 4 \\ & & & & & & \xrightarrow{\text{rm}} \end{array}$$

$S + S * 4$ is called right sentential form. Thus we can obtain the desired language L by certain rules. Let us now solve some examples for derivation of CFG.

Problems on context free grammar

→ Example 3.2.1 : Construct the CFG for the language having any number of a 's over the set $\Sigma = \{a\}$

Solution : As we know the regular expression for above mentioned language is

$$\text{r.e.} = a^*$$

Let us build the production rules for the same,

$$S \rightarrow aS \quad \text{rule 1}$$

$$S \rightarrow \epsilon \quad \text{rule 1}$$

Now if want "aaa" string to be derived we can start with start symbols.

S

aS

$a a S \quad :: \quad \text{rule 1}$

$a a a S \quad :: \quad \text{rule 1}$

$a a a a S \quad :: \quad \text{rule 1}$

$a a a a a S \quad :: \quad \text{rule 1}$

$a a a a a \epsilon \quad :: \quad \text{rule 2}$

= aaa

The r.e. $= a^*$ suggest a set of $\{\epsilon, a, aa, aaa, \dots\}$. We can have a null string simply because S is a start symbol, and rule 2 gives $S \rightarrow \epsilon$.

→ Example 3.2.2 : Try to recognize the language L for given CFG.

$$G = [\{S\}, \{a, b\}, P, \{S\}]$$

$$\text{where } P = \left\{ \begin{array}{l} S \rightarrow aSb \\ S \rightarrow ab \end{array} \right\}$$

Solution : Since $S \rightarrow aSb \mid ab$ is a rule. \mid indicates the 'or' operator.

$$S \rightarrow aSb$$

If this rule can be recursively applied then,

S

aSb

↓

$a a S b b$

↓

$a a a S b b b$

and if finally we can put $S \rightarrow ab$ then it becomes $aaaa\ bbbb$. Thus we can have any number of a's first then equal number of b's following it. Hence we can guess the language as $\{L = a^n b^n \text{ where } n \geq 1\}$. The only way to recognize the language is to try out various strings from the given production rules. Simply by observing the derived strings, one can find out the language getting generated from given CFG.

Example 3.2.3 : Construct the CFG for the regular expression $(0+1)^*$

Solution : The CFG can be given by,

$$\begin{aligned} P &= \{S \rightarrow 0S \mid 1S \\ &\quad S \rightarrow \epsilon\} \end{aligned}$$

The rules are in combination of 0's and 1's with the start symbol. Since $(0+1)^*$ indicates $\{\epsilon, 0, 1, 01, 10, 00, 11, \dots\}$ in this set ϵ is a string. So in the rules we can set the rule $S \rightarrow \epsilon$.

Example 3.2.4 : Construct a grammar for the language containing strings of at least two a's.

Solution : Let $G = (V, T, P, S)$

where

$$\begin{aligned} V &= \{S, A\} \\ T &= \{a, b\} \\ P &= \{S \rightarrow AaAaA \\ &\quad A \rightarrow aA \mid bA \mid \epsilon\} \end{aligned}$$

The rule $S \rightarrow AaAaA$ is something in which the two a's are maintained since at least two a's should be there in the strings. And $A \rightarrow aA \mid bA \mid \epsilon$ gives any combination of a's and b's i.e. this rule gives the strings of $(a+b)^*$.

Thus the logic for this example will be

(any thing) a (any thing) a (any thing)

So before a or after a there could be any combination of a's and b's.

Example 3.2.5 : Construct a grammar generating

$$L = w c w^T \text{ where } w \in \{a, b\}^*$$

Solution : The strings which can be generated for given L is {mcm, bcb, abcba, bacab ...}

The grammar could be

$$S \rightarrow aSa$$

$$S \rightarrow bSb$$

$$S \rightarrow c$$

Since the language $L = w c w^T$ where $w \in (a + b)^*$

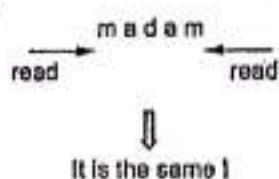
Hence $S \rightarrow a S a$ or $S \rightarrow b S b$. The string $abcbba$ can be generated from given production rules as

S	<table border="1"><tr><td>a</td><td>b</td><td>c</td><td>b</td><td>a</td></tr></table>	a	b	c	b	a
a	b	c	b	a		
$a S a$	<table border="1"><tr><td>a</td><td>b</td><td>c</td><td>b</td><td>a</td></tr></table>	a	b	c	b	a
a	b	c	b	a		
$a b S b$	<table border="1"><tr><td>a</td><td>b</td><td>c</td><td>b</td><td>a</td></tr></table>	a	b	c	b	a
a	b	c	b	a		
$a b c b a$	<table border="1"><tr><td>a</td><td>b</td><td>c</td><td>b</td><td>a</td></tr></table>	a	b	c	b	a
a	b	c	b	a		

Thus any of this kind of string could be derived from the given production rules.

→ Example 3.2.6 : Construct CFG for the language L which has all the strings which are all palindrome over $\Sigma = \{a, b\}$.

Solution : As we know the strings are palindrome if they possess same alphabets from forward as well as from backward.



For example, the string "madam" is a palindrome because

Since the language L is over $\Sigma = \{a, b\}$. We want the production rules to be build a's and b's. As ϵ can be the palindrome, a can be palindrome even b can be palindrome. So we can write the production rules as

$$G = ([S], \{a, b\}, P, S)$$

P can be

$$S \rightarrow a S a$$

$$S \rightarrow b S b$$

$$S \rightarrow a$$

$$S \rightarrow b$$

$$S \rightarrow \epsilon$$

The string abaaba can be derived as

S

$a S a$	$\epsilon b a a b \epsilon$
$a b S b a$	$\epsilon b a a b a$
$a b a S a b$	$\epsilon b a a b a$
$a b a a b$	$\epsilon b a a b a$
$a b a a b a$	$\epsilon b a a b a$

which is a palindrome.

Example 3.2.7 : Construct CFG which consists of all the strings having at least one occurrence of 000.

Solution : The CFG for this language can be equivalent to r.e. (any thing) (000) (anything)

$$\text{Thus r.e.} = (0+1)^* 000 (0+1)^*$$

Let us build the production rules as

$$S \rightarrow ATA$$

$$A \rightarrow 0A \mid 1A \mid \epsilon$$

$$T \rightarrow 000$$

Example 3.2.8 : Construct CFG for the language in which there are no consecutive b's, the strings may or may not have consecutive a's.

$$\text{Solution : } S \rightarrow aS \mid bA \mid a \mid b \mid \epsilon$$

$$A \rightarrow aS \mid a \mid \epsilon$$

In the above rules, there is no condition on occurrence of a's. But no consecutive b's are allowed. Note that in the rule $S \rightarrow bA$ and A gives all the strings which are starting with letter a.

$$\text{Thus } G = (S, A), \{a, b\}, P, S$$

Let us derive the string

Derivation	Input	Productions
S		
aS	a a b a b	$S \rightarrow aS$
aaS	a a b a b	$S \rightarrow aS$
aa b A	a a b a b	$S \rightarrow bA$
aa b a S	a a b a b	$A \rightarrow aS$
aa b a b	a a b a b	$S \rightarrow b$

Example 3.2.9 : Recognize the context free language for the given CFG.

$$S \rightarrow aB \mid bA$$

$$A \rightarrow a \mid aS \mid bAA$$

$$B \rightarrow b \mid bS \mid aBB$$

Solution : To find the language denoted by given CFG we try to derive the rules and get various strings. Then after observing those strings we come to know, which language it is denoting. Let us rewrite all those rules and number them.

$$S \rightarrow aB \quad ; \quad \text{rule 1}$$

$S \rightarrow b A$	rule 2
$A \rightarrow a$	rule 3
$A \rightarrow a S$	rule 4
$A \rightarrow b A A$	rule 5
$B \rightarrow b$	rule 6
$B \rightarrow b S$	rule 7
$B \rightarrow a B B$	rule 8

Now let us apply the rules randomly starting with start symbol.

S	
$a B$	rule 1
$a a B B$	rule 8
$a a b S B$	rule 7
$a a b b A B$	rule 2
$a a b b a B$	rule 3
$a a b b a b S$	rule 7
$a a b b a b b A$	rule 2
$a a b b a b b a$	rule 3

We have got some string as "aabbabba". Let us try out something else.

S	
$b A$	rule 2
$b b A A$	rule 5
$b b a S A$	rule 4
$b b a a B A$	rule 1
$b b a a b A$	rule 6
$b b a a b a$	rule 3

Now we have got 'bababa' such a string !

Thus by obtaining more and more strings by applying more and more rules randomly will help us to find out what language it indicates. Observe the strings generated carefully, we can draw a conclusion as these strings contain equal number of a's and equal number of b's. Even you can try out various rules to obtain some more strings. And see that it is a language L containing all the strings having equal number of a's and b's.

→ Example 3.2.10 : Construct CFG for the language containing at least one occurrence of double a.

Solution : The CFG can be built with a logic as : One rule we will built for double a i.e. $A \rightarrow aa$.

And the other rule we will built for any number of a 's and b 's in any combination.
i.e. $B \rightarrow aB \mid bB \mid \epsilon$ which is always equivalent to $(a + b)^*$ (you can note it as golden rule !)

If we combine both of these rules we can get the desired context free grammar.

$$S \rightarrow BAB \Rightarrow (\text{anything}) \left(\begin{array}{c} \text{one occurrence} \\ \text{of double } a \end{array} \right) (\text{anything})$$

$$A \rightarrow aa$$

$$B \rightarrow aB \mid bB \mid \epsilon$$

Thus the CFG contains $V = \{A, B, S\}$ $T = \{a, b\}$. The start symbol is S .

Let us derive "ababb"

S		
BAB		
aBAB	ababb	$B \rightarrow aB$
abBAB	ababb	$B \rightarrow bB$
abAB	ababb	$B \rightarrow \epsilon$
abaaB	ababb	$A \rightarrow aa$
abbaB	ababb	$B \rightarrow bB$
abbae	ababb	$B \rightarrow \epsilon$
= ababb	ababb	

→ **Example 3.2.11 :** Construct CFG for the language containing all the strings of different first and last symbols over $\Sigma = \{0, 1\}$.

Solution : Since the problem statement says as if the string starts with 0 it should end with 1 or if the string starts with 1 it should end with 0.

$$S \rightarrow 0 A 1 \mid 1 A 0$$

$$A \rightarrow 0 A \mid 1 A \mid \epsilon$$

Thus clearly, in the above CFG different start and end symbols are maintained. The non terminal $A \rightarrow 0A \mid 1A \mid \epsilon$ indicates $(0 + 1)^*$. Thus the given CFG is equivalent to the regular expression $[0(0+1)^*1 + 1(0+1)^*0]$.

→ **Example 3.2.12 :** Construct the CFG for the language $L = a^n b^{2n}$ where $n \geq 1$.

Solution : As in some previous example we have seen the case of $a^n b^n$. Now the number of b 's are doubled. So we can write

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

It is as simple as this !

- Example 3.2.13 : Construct the production rules for defining a language
 $L = \{a^x b^y \mid x \neq y\}$.

Solution : This is the language in which all the a's must appear before all the b's. But total number of a's must not be equal to total number of b's. Either number of a's must be equal to number of b's or number of b's must be equal to number of a's. The rule

$$S \rightarrow aSb$$

gives us equal number of a's must be followed by equal number of b's. But according to problem statement we need more number of a's either or more number of b's. Hence the required production rules can be -

$$S \rightarrow aSb \mid R1 \mid R2$$

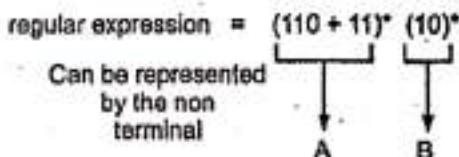
$$R1 \rightarrow aR1 \mid \epsilon$$

$$R2 \rightarrow bR2 \mid b$$

where S is a start symbol.

- Example 3.2.14 : Find the CFG for the regular expression $(110 + 11)^*(10)^*$.

Solution : Let



If we assume S as start symbol then,

$$S \rightarrow AB$$

Now for each non terminal A and B we can define production rules as

$$A \rightarrow 110A \mid 11A \mid \epsilon$$

$$B \rightarrow 10B \mid \epsilon$$

Hence, finally the CFG for given r.e. can be

$$S \rightarrow AB$$

$$A \rightarrow 110A \mid 11A \mid \epsilon$$

$$B \rightarrow 10B \mid \epsilon$$

- Example 3.2.15 : Build a CFG for generating the integers.

Solution : The integer is any numerical value without decimal place which can be either signed or unsigned. Hence the CFG can be -

$$S \rightarrow GI$$

$$G \rightarrow + \mid -$$

$$I \rightarrow DI \mid D$$

$$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \dots \mid 9$$

Here G is for denoting sign, I denotes integer and D denotes digits from 0 to 9. To understand the rules lets derive these rules for some integer.

Let, -21 can be derived as

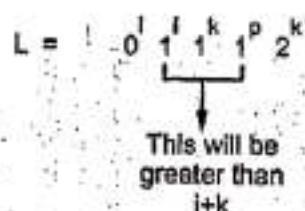
$$\begin{array}{ll} S \rightarrow GI \\ - I & G \rightarrow - \\ - DI & I \rightarrow DI \\ - 2I & D \rightarrow 2 \\ - 2D & I \rightarrow D \\ - 21 & D \rightarrow 1 \end{array}$$

→ Example 3.2.16 : Design a CFG for the following language.

$$L = \{ 0^i 1^j 0^k / j > i + k \}$$

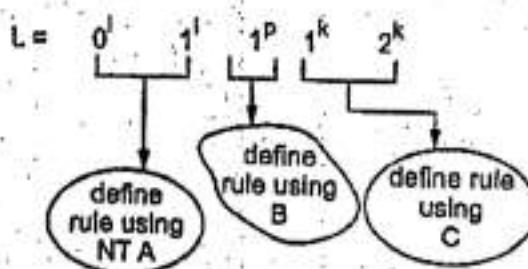
GTU : Summer-11, Marks 4, Summer-12, Marks 7

Solution : As the language $L = 0^i 1^j 2^k$ such that $j > i + k$.



Hence we can rewrite L as

We can again rewrite L for simplification as -



$$A \rightarrow 0 A 1 \mid \epsilon$$

$$B \rightarrow 1 B \mid 1$$

$$C \rightarrow 1 C 2 \mid \epsilon$$

Hence the CFG can be

$$\begin{aligned} S &\rightarrow A B C \\ A &\rightarrow 0 A 1 \mid \epsilon \\ B &\rightarrow 1 B \mid 1 \\ C &\rightarrow 1 C 2 \mid \epsilon \end{aligned}$$

→ Example 3.2.17 : Build a CFG for the language $L = \{0^i 1^j 2^k \mid i=j\}$

Solution : As $i = j$ and there is no condition on k . We can say that k is an arbitrary value. Then we rewrite L as

$$\begin{aligned} S &\rightarrow A B \\ A &\rightarrow 0 A 1 \mid \epsilon \\ B &\rightarrow 2 B \mid \epsilon \end{aligned}$$

→ Example 3.2.18 : Obtain CFG for the language $L = \{0^i 1^j 2^k \mid j \leq k\}$

Solution : Here $j \leq k$. That means the language L has two variations.

$$\begin{aligned} L_1 &= 0^i 1^j 2^j \\ L_2 &= 0^i 1^j 2^k \text{ where } k > j. \text{ Hence } L_2 = 0^i 1^j 2^j \cdot 2^k \end{aligned}$$

Hence the required CPG can be

$$\begin{aligned} S &\rightarrow A B \\ A &\rightarrow 0 A \mid \epsilon \\ B &\rightarrow 1 B 2 C \mid 12 \\ C &\rightarrow 2 C \mid \epsilon \end{aligned}$$

→ Example 3.2.19 : Find CFG for the following languages.

1. $L = \{a^i b^j c^k \mid j > i + k\}$
2. $L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$

GTU : Winter-11, Marks 5; Winter-19, Marks 2

Solution : 1) Refer example 3.2.16 by assuming 0 as a and 1 as b

2) There are two conditions i) $i = j$ or ii) $j = k$. Hence CFG can be

$$\begin{aligned} S &\rightarrow AB \mid XY \\ A &\rightarrow aAb \mid \epsilon \\ B &\rightarrow cB \mid \epsilon \\ X &\rightarrow aX \mid \epsilon \\ Y &\rightarrow bYc \mid \epsilon \end{aligned}$$

Example 3.2.20 : Let L be the language corresponding to the regular expression $(011 + 1)^* (01)^*$. Find the CFG generating L . GTU : Summer-14, 16, Marks 7

Solution : r.e. = $(011+1)^* (01)^*$

We will breakup the given regular expression into sub-regular expressions.

$$\text{r.e.} = \boxed{(011+1)^*} \quad \boxed{(01)^*}$$

$\downarrow \quad \downarrow$
r1 r2

For each r1 and r2 the CFG can be written as follows :

Production rule for r1	Production rule for r2
$A \rightarrow 011A \mid 1A \mid 011 \mid 1 \mid \epsilon$	$B \rightarrow 01B \mid \epsilon$

By combining above production rules we can create CFG $G = (V, T, P, S)$

where $V = (S, A, B)$ with S as a start symbol.

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

The set of production rules P is

$$\begin{aligned} S &\rightarrow AB|\epsilon \\ A &\rightarrow 011A \mid 1A \mid 011 \mid 1 \mid \epsilon \\ B &\rightarrow 01B \mid \epsilon \end{aligned}$$

Example 3.2.21 : Define CFG and design a CFG for the following language.
 $L = \{x \in \{0,1\}^* \mid n_0(x) \neq n_1(x)\}$ GTU : Summer-14, Marks 7

Solution : CFG - Refer section 3.2

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

$V = (S, A, B)$ with S as a start symbol.

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

The set of production rules P is

$$\begin{array}{ll} S \rightarrow A & /* \text{Defines more 0's than 1's */} \\ S \rightarrow B & /* \text{Defines more 1's than 0's */} \\ A \rightarrow aA \mid a \\ A \rightarrow aAb \\ B \rightarrow bB \mid b \\ B \rightarrow aBb \end{array}$$

→ Example 3.2.22 : Generate the context free grammar that give the following languages.

- i) {W | W contains at least three 1's}
- ii) {W | W starts and ends with same symbol}

GTU : Winter-14, Marks 7

Solution : We will assume $\Sigma = \{0, 1\}$

- i) The regular expression for given W is

$$\text{r.e.} = (0 + 1)^* 1 (0 + 1)^* 1 (0 + 1)^* 1 (0 + 1)^*$$

Hence P the set of production rules is

$$S \rightarrow A \ 1 \ A \ 1 \ A \ 1 \ A$$

$$A \rightarrow 0A \mid 1A \mid \epsilon$$

- ii) The r.e. will be

$$\text{r.e. } [0 (0 + 1)^* 0] + [1 (0 + 1)^* 1]$$

Hence CFG will be

$$S \rightarrow 0A0 \mid 1A1$$

$$A \rightarrow 0A \mid 1A \mid \epsilon$$

→ Example 3.2.23 : Define Context Free Grammar (CFG). Design CFG for generating following languages :

1) For balanced parenthesis

2) Set of even length strings in {a, b, c, d} with two middle symbol equal.

GTU : Summer-15, Marks 7

Solution : 1) The CFG can be

$$S \rightarrow (S)$$

$$S \rightarrow SS$$

$$S \rightarrow \epsilon$$

2) The CFG will be

$$S \rightarrow (a \mid b \mid c \mid d) S (a \mid b \mid c \mid d) \mid a.a \mid b.b \mid c.c \mid d.d$$

→ Example 3.2.24 : Define context free grammar. Find context-free grammar for the language : $L = \{a^i b^j \mid i < 2j\}$.

GTU : Summer-18, Marks 7

Solution : Context free grammar :

$$\text{Let, } L = \{a^i b^j \mid i < 2j\}$$

The production rules can be,

$$S \rightarrow \epsilon \mid Sb \mid aSb$$

⇒ Example 3.2.25 : Find context free grammar for the following language.

$$L_1 = \{a^i b^j c^k \mid i = j + k\}, L_2 = (011+1)^* (01)^*, L_3 = (0+1)1^* (1+(01)^*)$$

GTU : Summer-19, Marks 7

Solution :

$$L_1 = S \rightarrow aSc \mid A$$

$$A \rightarrow aAb \mid \epsilon$$

$$L_2 = \text{Refer example 3.2.20}$$

$$L_3 = \text{Let, regular expression r.e. } = (0+1)1^* (1+(01)^*)$$

$$\therefore S \rightarrow ABC$$

$$A \rightarrow 0 \mid 1$$

$$B \rightarrow 1B \mid \epsilon$$

$$C \rightarrow D \mid E$$

$$D \rightarrow 1$$

$$E \rightarrow 01E \mid \epsilon$$

⇒ Example 3.2.26 : Give CFG for following languages :

$$1) L = a^* b^* \quad 2) L = \{a^{n+2} b^n \mid n \geq 0\}$$

GTU : Summer-19, Marks 4

Solution :

$$1) \quad L = a^* b^*$$

$$S \rightarrow AB$$

$$A \rightarrow aA \mid \epsilon$$

$$B \rightarrow bB \mid \epsilon$$

$$2) \quad L = a^{n+2} b^n$$

$$S \rightarrow aSb \mid aa$$

⇒ Example 3.2.27 : Describe the language generated by the following grammars :

$$i) S \rightarrow aA \mid bC \mid b \quad ii) S \rightarrow aT \mid bT \mid \Lambda$$

$$A \rightarrow aS \mid bB$$

$$T \rightarrow aS \mid bS$$

$$B \rightarrow aC \mid bA \mid a$$

$$C \rightarrow aB \mid bS$$

GTU : Winter-19, Marks 3

Solution :

i) $L = \{ \text{The strings contain } a's \text{ and } b's \text{ in such a way that } b \text{ always appear odd number of times and } a \text{ always appear even number of times.} \}$

ii) $L = \{ \text{The language contains any number of } a's \text{ or } b's \text{ but the length of the string should be even.} \}$

3.3 Regular Grammar

A regular grammar is defined as

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

V is set of symbols called nonterminals which are used to define the rules.

T is a set of symbols called terminals.

P is a set of production rules.

S is a start symbol which $\in V$.

The production rules P are of the form.

$$A \rightarrow aB$$

$$A \rightarrow a$$

where A and B are non terminal symbols, and a is terminal symbol.

For example

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

$$V = \{S, A\}$$

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

S is a start symbol and production rules are as given below -

$$S \rightarrow 0S$$

$$S \rightarrow 1B$$

$$B \rightarrow \epsilon$$

This is called a regular language and it can be represented by some DFA. Thus, when a grammar can be represented by some finite automata then it, is a regular grammar.

3.3.1 Construction of Regular Grammar from Regular Expression

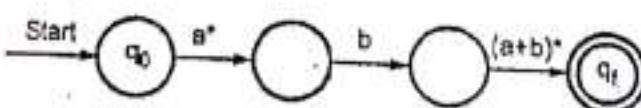
We can convert the regular expression into its equivalent regular grammar by using following method -

1. Construct a NFA with ϵ from given regular expression.
2. Eliminate ϵ transitions and convert it to equivalent DFA.
3. From constructed DFA, the corresponding states become nonterminal symbols and transitions made are equivalent to production rules.

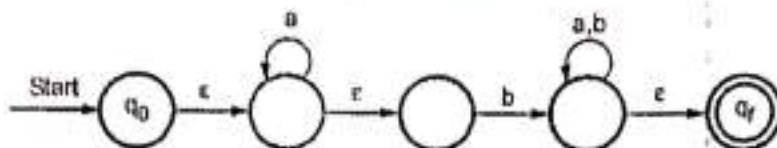
We will understand this method with the help of some examples -

→ Example 3.3.1 : Construct a regular grammar for the regular expression $a^*b(a + b)^*$.

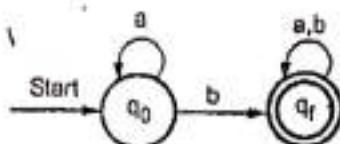
Solution : We will first construct a DFA in a straightforward manner for given regular expression.



Now we will convert it to NFA with ϵ transitions.



Now eliminate ϵ moves,



We can write the regular grammar as,

$$G = (V, T, P, A_0) \text{ where}$$

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

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

$$P = \{A_0 \rightarrow aA_0$$

$$A_0 \rightarrow bA_1$$

$$A_0 \rightarrow b$$

$$A_1 \rightarrow aA_1$$

$$A_1 \rightarrow bA_1$$

$$A_1 \rightarrow a$$

$$A_1 \rightarrow b$$

}

and A_0 is a start symbol.

Thus G is required regular grammar.

3.3.2 Right-linear and Left-linear Grammar

If the non terminal symbol appears as a rightmost symbol in each production of regular grammar then it is called right linear grammar. The right linear grammar is of following form

$$A \rightarrow aB$$

$$A \rightarrow a$$

$$A \rightarrow \epsilon$$

Where A and B are nonterminal symbols and 'a' is a terminal symbol.

If the nonterminal symbol appears as a leftmost symbol in each production of regular grammar then it is called left linear grammar.

The left linear grammar is of following form

$$A \rightarrow Ba$$

$$A \rightarrow a$$

$$A \rightarrow \epsilon$$

Where A and B are non terminal symbols and 'a' is a terminal symbol.

The language is called regular if it is accepted by either left linear or right linear grammar.

→ Example 3.3.2 : Construct right linear grammar for the following DFA

	0	1
→ A	B	C
B	B	C
C	A	C

Solution : The transition graph will be shown in Fig. 3.3.1.

$$A \rightarrow 0B|1C|0$$

$$B \rightarrow 0B \mid 0$$

$$C \rightarrow 0A \mid 1A$$

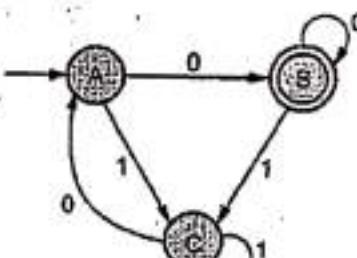


Fig. 3.3.1

→ Example 3.3.3 : Give the left linear grammar for $RE(10)^*1$.

GTU : Summer-19, Marks 3

Solution :

$$S \rightarrow A1$$

$$A \rightarrow 10A \mid \epsilon$$

→ Example 3.3.4 : Construct finite automata for following left linear grammar :

$$S \rightarrow X0|Y1$$

$$X \rightarrow Y1$$

$$Y \rightarrow Y0|1$$

GTU : Summer-19, Marks 7

Solution :

Step 1 : We will first convert the given left linear grammar to right linear grammar.

$$S \rightarrow 0X|1Y$$

$$X \rightarrow 1Y$$

$$Y \rightarrow 0Y|1$$

Step 2 : Now create Finite automata

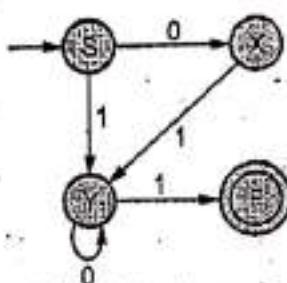


Fig. 3.3.2

Step 3 : Now change start state to final state and final state to start state. Also reverse the directions in FA. Then the required FA becomes

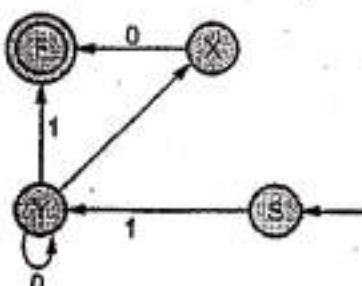


Fig. 3.3.3

3.4 Derivation and Languages

The production rules are used to derive certain strings. If $G = (V, T, P, S)$ be some context free grammar then generation of some language using specific rules is called derivation.

As we know, derivation for any string means replacement of non terminal by its appropriate definition. There may be a situation, in which there are many non terminals. Then which non terminal should be replaced by its definition is sometimes confusing to decide.

Hence we normally apply two methods of deriving. The leftmost derivation is a derivation in which the leftmost non terminal is replaced first from the sentential form.

The rightmost derivation is a derivation in which rightmost non terminal is replaced first from the sentential form.

Let us see how it works.

For example

$S \rightarrow XYX$	$S \rightarrow XYX$
$S \rightarrow aYX$	$S \rightarrow XYa$
$S \rightarrow abX$	$S \rightarrow Xba$
$S \rightarrow aba$	$S \rightarrow dba$

Leftmost derivation Rightmost derivation

Note that we have replaced first X from left to right in leftmost derivation and XYX the last X i.e. the rightmost symbol.

Actually, we may use leftmost derivation or rightmost derivation we get the same string. The type of derivation does not affect on getting of a string.

Let us solve few problems based on this topic.

→ Example 3.4.1 : Derive the string "abbabba" for leftmost derivation and rightmost derivation using a CFG given by,

$$\begin{aligned} S &\rightarrow aB|bA \\ A &\rightarrow a|aS|bAA \\ B &\rightarrow b|bS|aBB \end{aligned}$$

Solution : Let us see the leftmost derivation first,

S	
aB	$S \rightarrow aB$
aaBB	$B \rightarrow aBB$
abbB	$B \rightarrow b$
abbS	$B \rightarrow bS$
abbbaB	$S \rightarrow aB$
abbbabS	$B \rightarrow bS$
abbbabA	$S \rightarrow bA$
abbabba	$A \rightarrow a$

is derived. Now let us solve using rightmost derivation.

S	
aB	$S \rightarrow aB$
aaBB	$B \rightarrow aBB$

$aBbS$	$B \rightarrow bS$
$aBbbA$	$S \rightarrow bA$
$aBbb\alpha$	$A \rightarrow \alpha$
$abSbb\alpha$	$B \rightarrow bS$
$abbAbba$	$S \rightarrow bA$
$abbabb\alpha$	$A \rightarrow \alpha$

is a derivation.

→ Example 3.4.2 : Derive the string 1000111 for leftmost and rightmost derivation using CFG.

$$G = (V, T, P, S) \text{ where}$$

$$V = \{S, T\}$$

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

$$P = \{S \rightarrow T00T$$

$$T \rightarrow 0T \mid 1T \mid \epsilon\}$$

Solution : The leftmost derivation can be

S	
T00T	$S \rightarrow T00T$
1T00T	$T \rightarrow 1T$
10T00T	$T \rightarrow 0T$
10 ε 00T	$T \rightarrow \epsilon$
1000T	
10001T	$T \rightarrow 1T$
100011T	$T \rightarrow 1T$
1000111T	$T \rightarrow 1T$
1000111 ε	$T \rightarrow \epsilon$
1000111	

Rightmost derivation -

S	
T00T	
T001T	$T \rightarrow 1T$
T0011T	$T \rightarrow 1T$
T00111T	$T \rightarrow 1T$
T00111 ε	$T \rightarrow \epsilon$

$T00111$
 $1T00111 \quad T \rightarrow 1T$
 $10T00111 \quad T \rightarrow 0T$
 $10 \epsilon 00111 \quad T \rightarrow \epsilon$
 1000111

→ Example 3.4.3 : Let G be the grammar

$$\begin{aligned} S &\rightarrow aB \mid bA \\ A &\rightarrow a \mid aS \mid bAA \\ B &\rightarrow b \mid bS \mid aBB \end{aligned}$$

For string $aaabbbaaa$, find left most derivation and right most derivation.

GTU : Summer-18, Marks 4

Solution :

Leftmost Derivation	Rightmost
S	S
aB	aB
$aaBB$	$aaBB$
$aaaBBB$	$aaBbS$
$aaabBB$	$aaBbbA$
$aaabbSB$	$aaBbba$
$aaabbaBB$	$aaaBBbba$
$aaabbabB$	$aaaBbbba$
$aaabbabbS$	$aaabSbbba$
$aaabbabbbA$	$aaabbAbba$
$aaabbabbbba$	$aaabbabbbba$

→ Example 3.4.4 : Consider the grammar :

$$\begin{aligned} S &\rightarrow aAS \mid a \\ A &\rightarrow SbA \mid SS \mid ba \end{aligned}$$

Derive left most and right most derivation of string $aabbbaa$ using given grammar.

GTU : Summer-19, Marks 3

Solution :

Leftmost Derivation	Rightmost Derivation
S	S
aAS	aAS
$aSbAS$	aAa
$aabbAS$	$aSbAa$
$aabbba$	$aSbbba$
	$aabbbaa$

3.5 Derivation Trees

Derivation trees is a graphical representation for the derivation of the given production rules for a given CFG. It is the simple way to show how the derivation can be done to obtain some string from given set of production rules. The derivation tree is also called parse tree.

Following are properties of any derivation tree -

1. The root node is always a node indicating start symbol.
2. The derivation is read from left to right.
3. The leaf nodes are always terminal nodes.
4. The interior nodes are always the non terminal nodes.

For example,

$S \rightarrow bSb \mid a \mid b$ is a production rule. The S is a start symbol.

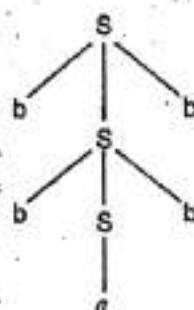
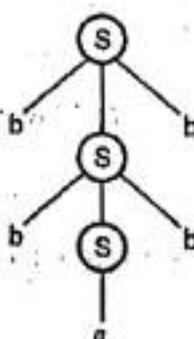


Fig. 3.5.1 Derivation tree

The above tree is a derivation tree drawn for deriving a string bbabb. By simply reading the leaf nodes we can obtain the desired string. The same tree can also be denoted by,

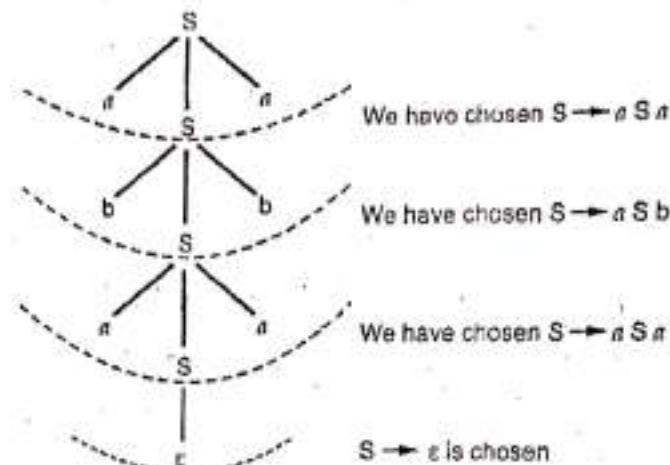


→ Example 3.5.1 : Draw a derivation tree for the string abaaba for the CFG given by,

G where $P = \{S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow a|b|\epsilon\}$



→ Example 3.5.2 : Construct the derivation tree for the string aabbabba from the CFG given by

$S \rightarrow aB|bA$

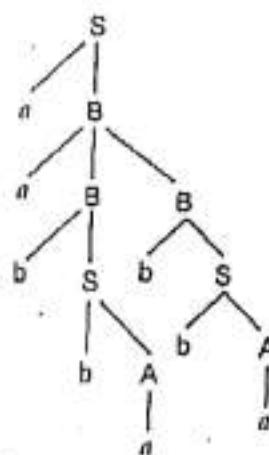
$A \rightarrow a|aS|bAA$

$B \rightarrow b|bS|aBB$

Solution : To draw a tree we will first try to obtain derivation for the string aabbabba

S	
aB	$S \rightarrow aB$
<u>a</u> <u>bB</u>	$S \rightarrow aBB$
<u>a</u> <u>b</u> <u>bS</u> B	$B \rightarrow bS$
<u>a</u> <u>b</u> <u>b</u> <u>A</u> B	$S \rightarrow bA$
<u>a</u> <u>b</u> <u>b</u> <u>a</u> <u>B</u>	$A \rightarrow a$
<u>a</u> <u>b</u> <u>b</u> <u>a</u> <u>bS</u>	$B \rightarrow bS$
<u>a</u> <u>b</u> <u>b</u> <u>a</u> <u>bA</u>	$S \rightarrow bA$
<u>a</u> <u>b</u> <u>b</u> <u>a</u> <u>b</u> <u>A</u>	$A \rightarrow a$

Now let us draw a tree.



Example 3.5.3 : Consider following grammar :

$$S \rightarrow A1B$$

$$A \rightarrow 0A \mid \lambda$$

$$B \rightarrow 0B \mid 1B \mid \lambda$$

Give leftmost and rightmost derivations of the string 00101. Also draw the parse tree corresponding to this string.

GTU : Summer-17. Marks 7

Solution :

Leftmost Derivation	Rightmost Derivation	Parse Tree
S	S	
A1B	A1B	
0A1B	A10B	
00A1B	A101B	
001B	A101	
0010	0A101	
0010B	00A101	
00101	00101	<pre> graph TD S -- A --> A1 S -- 1 --> B1 A1 -- 0 --> A2 A1 -- A --> A3 A2 -- 0 --> A4 A2 -- 0 --> A5 A3 -- 1 --> A6 A3 -- B --> B2 B2 -- B --> B3 B2 -- A --> B4 B4 -- A --> B5 B5 -- lambda --> B6 </pre>

3.5.1 Relationship between Derivation Trees and Derivation

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

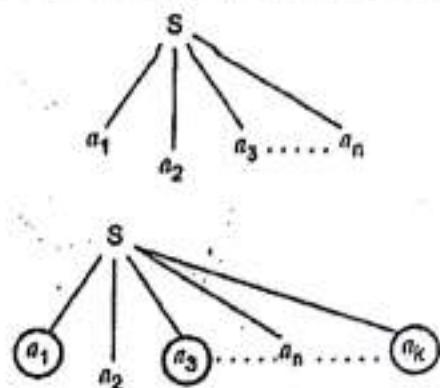
Proof : If there is a non terminal S in V then $S \Rightarrow a$ if and only if there is a S -tree which gives the string a . We can easily prove this using induction theorem. We can apply induction on number of interior nodes.

Basis of Induction :

Let us assume S is the only one interior which gives the string a which can be obtained by deriving $S \Rightarrow a_1, a_2, a_3, \dots, a_n$. Then there exists a derivation tree which derives $a_1, a_2, a_3, \dots, a_n$ and gives the string a .

Induction step : We assume, that the derivation will enable us to draw the tree with $K-1$ interior nodes. We have to prove, that it is possible to draw the derivation tree for K interior nodes which gives the string a . In the derivation tree not all the nodes are leaves or interior nodes. Some are leaves whereas others are interior nodes considering for them S is a parent node.

Thus this derivation tree will ultimately give the string a .



3.6 Ambiguous CFG

The grammar can be derived in either leftmost derivation or rightmost derivation. One can draw a derivation tree called as parse tree or syntax tree based on these derivations. The parse tree has to be unique even though the derivation is leftmost or rightmost.

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

For example - The CFG given by $G = (V, T, P, S)$

where

$$V = \{E\}$$

$$T = \{\text{id}\}$$

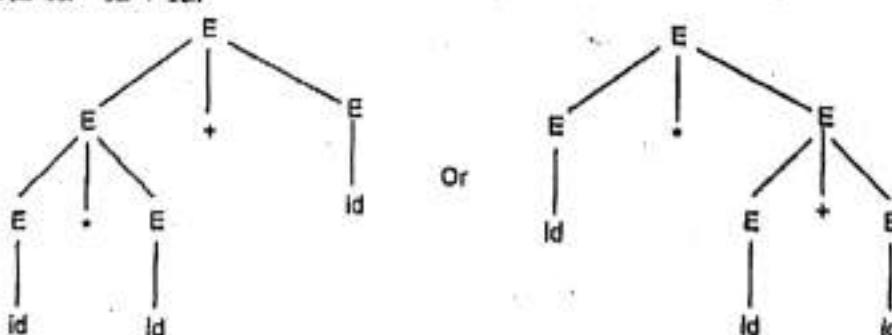
$$P = \{E \rightarrow E + E$$

$$E \rightarrow E * E$$

$$E \rightarrow \text{id} \}$$

$$S = \{E\}$$

Now if the string is $\text{id} * \text{id} + \text{id}$ then we can draw the two different parse trees indicating our $\text{id} * \text{id} + \text{id}$.



Thus the above grammar is an ambiguous grammar. Let us solve some exercise on it.

→ Example 3.6.1 : Check whether the given grammar is ambiguous or not.

$$S \rightarrow C \cup S$$

$$S \rightarrow iC \cup S \cup e$$

$$S \rightarrow \alpha$$

$$C \rightarrow b$$

Solution : To check whether given grammar is ambiguous or not we will have some string for derivation tree such as ibtibtibtbta.

Let us draw the derivation tree.

ibtibtibtbta

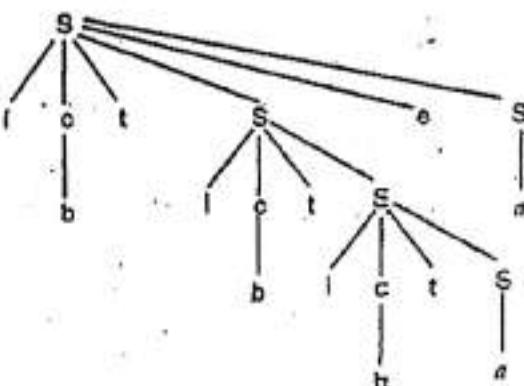


Fig. 3.6.1 (a) For example 3.6.1

or

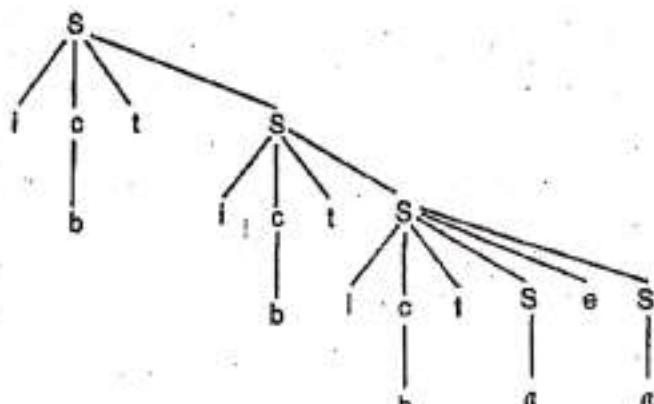


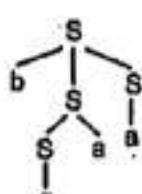
Fig. 3.6.1 (b) For example 3.6.1

Thus we have got more than two parse trees. Hence the given grammar is ambiguous.

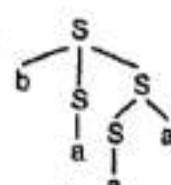
→ Example 3.6.2 : Show that CFG $S \rightarrow a \mid Sa \mid bSS \mid SSB \mid SbS$ is ambiguous.

GTU : Summer-13, Marks 7

Solution : Consider the string baab



Parse tree 1



Parse tree 2

As there are two different parse trees for given string for the given CFG. The corresponding CFG is said to be ambiguous.

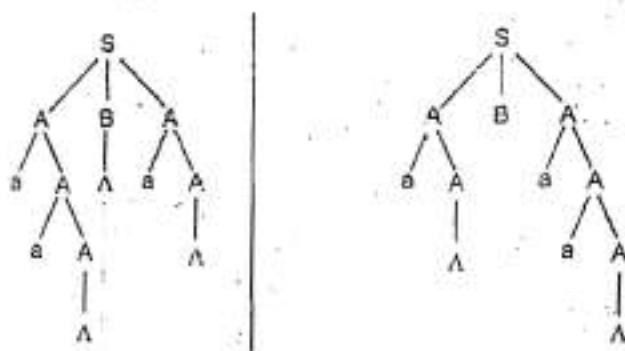
→ Example 3.6.3 : Define - ambiguous grammar, leftmost derivation. Check whether the following grammars are ambiguous or not. Justify your answer with proper reason.

- i) $S \rightarrow ABA$ ii) $S \rightarrow A|B$
 $A \rightarrow aA|\Lambda$ $A \rightarrow aAb|aab$
 $B \rightarrow bB|\Lambda$ $B \rightarrow abB|\Lambda$

GTU : Winter-19, Marks 7

Solution : Ambiguous grammar : Refer section 3.6.

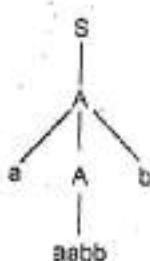
- i) Let, us derive string "aaa"



This shows that given grammar is ambiguous.

- ii) Let us derive the string aaabbb

This shows that given grammar is not ambiguous.



Review Question

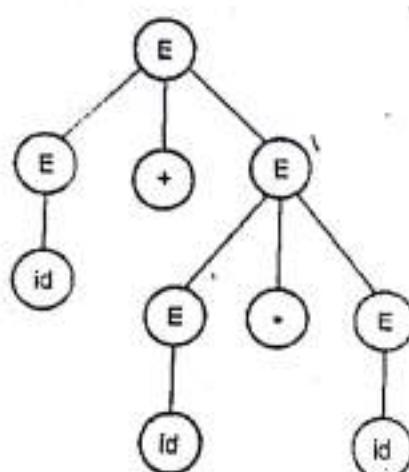
1. Define CFG. When is a CFG called an 'ambiguous CFG' ?

GTU : Summer-17, Marks 3

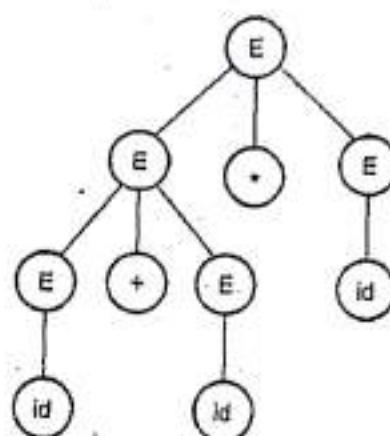
3.7 Unambiguous CFG and Algebraic Expressions

Consider the ambiguous grammar as -

$$E \rightarrow E + E \mid E * E \mid id$$



(a) Parse tree 1



(b) Parse tree 2

Fig. 3.7.1 Ambiguous grammar

is an ambiguous grammar. We will design the parse tree for $id + id * id$ as follows.

For removing the ambiguity we will apply one rule : If the grammar has left associative operator (such as $+, -, *, /$) then induce the left recursion and if the grammar has right associative operator (exponential operator) then induce the right recursion.

The unambiguous grammar is

$$E \rightarrow E + T$$

$$E \rightarrow T$$

$$T \rightarrow T * F$$

$$T \rightarrow F$$

$$F \rightarrow id$$

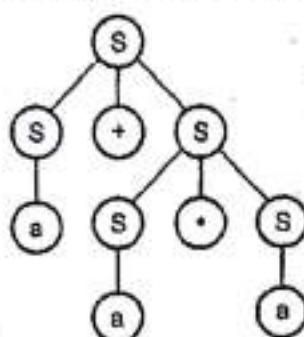
→ Example 3.7.1 : Prove that the following CFG is ambiguous.
 $S \rightarrow S + S \mid S^* S \mid (S) a$

Write the unambiguous CFG for the above grammar. Draw parse tree for the string $a + a^* a$

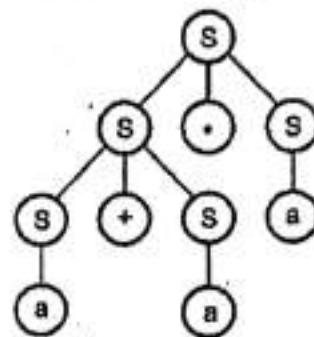
GTU : Winter-12. Marks 7

Solution : Let, $S \rightarrow S + S | S * S | (S)a$

To prove that above grammar is ambiguous, consider the string at $a+a^*a$



Parse tree 1



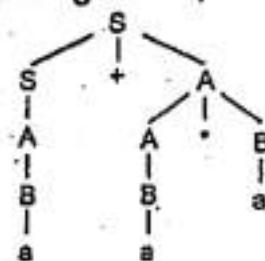
Parse tree 2

As we get two different parse trees for the same string, we call this grammar as ambiguous grammar.

The equivalent unambiguous grammar will be -

$$\begin{aligned} S &\rightarrow S + A \\ S &\rightarrow A \\ A &\rightarrow A^* B \\ A &\rightarrow B \\ B &\rightarrow (S) \\ B &\rightarrow a \end{aligned}$$

The parse tree for $a+a^*a$ with unambiguous grammar,



→ **Example 3.7.2 :** Find an equivalent unambiguous grammar for following :

$$S \rightarrow A | B \quad A \rightarrow aAb | ab \quad B \rightarrow abB | A$$

GTU : Summer-13, Marks 7

Solution : Let, the given ambiguous CFG is

$$\begin{aligned} S &\rightarrow A | B \\ A &\rightarrow aAb | ab \\ B &\rightarrow abB | \lambda \end{aligned}$$

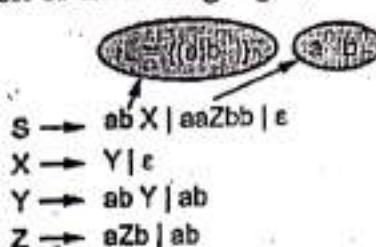
The A non-terminal derives the language

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

The B non-terminal derives the language

$$L = \{(ab)^n \text{ where } n \geq 0\}$$

By considering the generation of these languages, the unambiguous grammar will be



$$S \rightarrow abX \mid aaZbb \mid \epsilon$$

$$X \rightarrow Y \mid \epsilon$$

$$Y \rightarrow abY \mid ab$$

$$Z \rightarrow aZb \mid ab$$

Example 3.7.3 : Consider the grammar : $S \rightarrow ABA, A \rightarrow aA \mid \epsilon, B \rightarrow bB \mid \epsilon$

Is given grammar ambiguous ? If so then remove ambiguity. GTU : Summer-19, Marks 7

Solution : Consider string aa

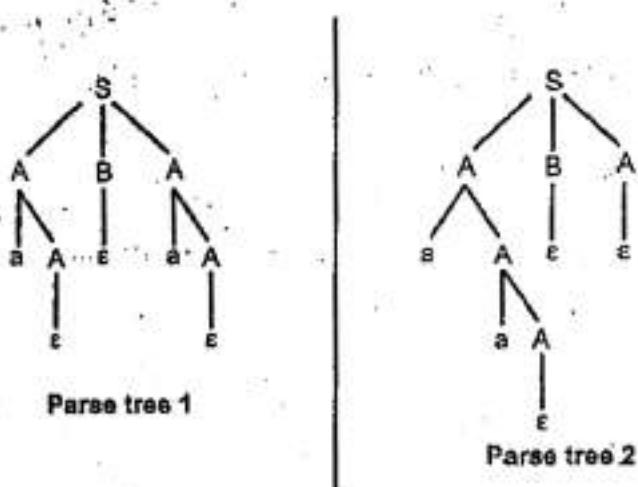


Fig. 3.7.2

As two different parse trees can be constructed for deriving string aa, the given grammar is ambiguous. Now we can remove the ambiguity of grammar, the grammar becomes

$$S \rightarrow aS \mid bS \mid \epsilon$$

3.8 Backus Naur Form (BNF)

- Backus Naur form is a representation of context free grammar in which particular notations are used.
- The non terminals in BNF are enclosed within special symbols < and >
- The empty string is written as <empty>
- The terminals appear as it is. Sometimes they can be denoted with quotes.
- The productions can be denoted using the symbol ::= which means "can be". The symbol " | " means OR can be used to denote the alternative definitions at the right hand side of the production.

- Example -

Consider following BNF rules

$\langle \text{stmt} \rangle ::= \langle \text{type} \rangle \langle \text{list} \rangle;$

$\langle \text{type} \rangle ::= \text{int} \mid \text{float}$

$\langle \text{list} \rangle ::= \langle \text{list} \rangle, \text{id}$

$\langle \text{list} \rangle ::= \text{id}$

Here stmt , type and list are nonterminal symbols. The int , float , id , $,$ and $;$ are terminal symbols. The $\langle \text{stmt} \rangle$ is a starting nonterminal.

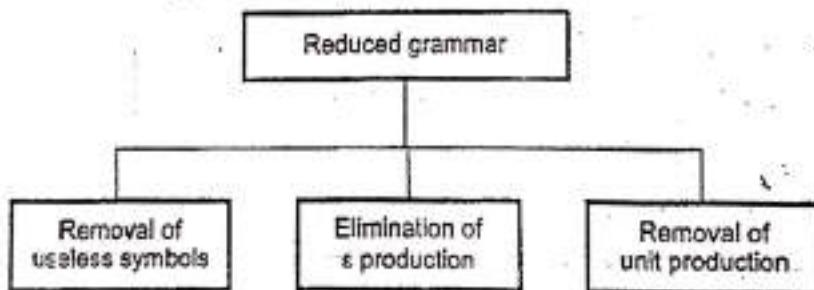
The alternatives are separated by $|$

3.9 Simplification of Grammar

As we have seen various languages can effectively be represented by context free grammar. All the grammars are not always optimized. That means grammar may consist of some extra symbols (non terminals). Having extra symbols unnecessary increases the length of grammar. Simplification of grammar means reduction of grammar by removing useless symbols. The properties of reduced grammar are given below.

1. Each variable (i.e. non terminal) and each terminal of G appears in the derivation of some word in L .
2. There should not be any production as $X \rightarrow Y$ where X and Y are non terminals.
3. If ϵ is not in the language L then there need not be the production $X \rightarrow \epsilon$.

We see the reduction of grammar as below.



Let us study the reduction process in detail.

3.9.1 Removal of Useless Symbols

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

A symbol p is useful if there exists some derivation in the following form.

$$S \xrightarrow{*} \alpha p \beta$$

and $\alpha p\beta \Rightarrow w$

Then p is said to be useful symbol.

Where α and β may be some terminal or non terminal symbol and will help us to derive certain string w in combination with P.

Example 3.9.1 : Consider the CFG $G = (V, T, P, S)$
where $V = \{S, A, B\}$ $T = \{0, 1\}$

$$P = \{ S \rightarrow A \ 11 \ B \mid 11A \}$$

$$S \rightarrow 1 \ B \mid 11$$

$$A \rightarrow 0$$

$B \rightarrow BB \}$ For removing useless symbols

Solution : Now in the given CFG if we try to derive any string A gives some terminal symbol as 0 but B does not give any terminal string. By following the rules with B we simply get ample number of B's and no significant string. Hence we can declare B as useless symbol and can remove the rules associated with it. Hence after removal of useless symbols we get,

$$S \rightarrow 11 \ A \mid 11$$

$$A \rightarrow 0$$

Example 3.9.2 : Find CFG with no useless symbols equivalent to

$$\begin{array}{ll} S \rightarrow AB \mid CA & B \rightarrow BC \mid AB \\ A \rightarrow a & C \rightarrow aB \mid b \end{array}$$

Solution : Consider, the rule

$$S \rightarrow AB \quad 1$$

$$S \rightarrow CA \quad 2$$

In the rule 2, C and A can be replaced by some terminating string but in rule 1, B cannot be replaced by terminating string. It tends to form a never ending loop.

E.g. : $S \rightarrow AB \rightarrow aAB \rightarrow aaAB \rightarrow aaaaAB$ and so on.

Thus we come to know as B is an useless symbol. Removing B the rules are now,

$$S \rightarrow CA$$

$$A \rightarrow a$$

$$C \rightarrow b$$

3.9.2 Elimination of ϵ Productions from Grammar

In context free grammar, if at all there is ϵ production we can remove it, without changing the meaning of the grammar. Thus ϵ productions are not necessary in a grammar.

For example,

$$\text{If } S \rightarrow 0S|1S|\epsilon$$

Then we can remove ϵ production. But we have to take a care of meaning of CFG. i.e. meaning of CFG should not get changed if we place $S \rightarrow \epsilon$ in other rules we get $S \rightarrow 0$ when $S \rightarrow 0S$ and $S \rightarrow \epsilon$

as well as $S \rightarrow 1$ when $S \rightarrow 1S$ and $S \rightarrow \epsilon$

Hence we can rewrite the rules as

$$S \rightarrow 0S|1S|0|1$$

Thus ϵ production is removed.

3.9.3 Removing Unit Productions

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

For example if $X \rightarrow Y$

$$Y \rightarrow Z \quad Z \rightarrow X$$

Then X , Y and Z are unit productions. To optimize the grammar we need to remove the unit productions.

If $A \Rightarrow B$ is a unit production and $B \rightarrow X_1 X_2 X_3 \dots X_n$ then while removing $A \rightarrow B$ production we should add a rule $A \rightarrow X_1 X_2 X_3 \dots X_n$:

Let us solve something.

→ Example 3.9.3 : If the CFG is as below.

$$S \rightarrow 0A|1B|C$$

$$A \rightarrow 0S|00$$

$$B \rightarrow 1|A$$

$$C \rightarrow 01$$

then remove the unit productions.

Solution : Clearly $S \rightarrow C$ is a unit production. But while removing $S \rightarrow C$ we have to consider what C gives. So, we can add a rule to S .

$$S \rightarrow 0A|1B|01$$

Similarly $B \rightarrow A$ is also a unit production so we can modify it as

$$B \rightarrow 1|0S|00$$

Thus finally we can write CFG without unit production as

$$S \rightarrow 0A|1B|01$$

$$A \rightarrow 0S|00$$

$$B \rightarrow 1|0S|00$$

$$C \rightarrow 01$$

⇒ Example 3.9.4 : Consider following grammar :

$$S \rightarrow ASB|\lambda$$

$$A \rightarrow aAS|a$$

$$B \rightarrow Sbs|A|bb$$

i) Eliminate useless symbols, if any.

ii) Eliminate λ productions.

GTU : Summer-17, Marks 4

Solution : i) If we observe CFG then both A and B derive terminal symbols, and therefore so does S. Thus there is no useless symbol in S.

ii) Only S is nullable. Hence we will place λ in the body of CFG whenever S occurs. The equivalent grammar will be

$$S \rightarrow ASB|AB$$

$$A \rightarrow aAS|aA|a$$

$$B \rightarrow Sbs|A|bb|Sb|bs|b$$

⇒ Example 3.9.5 : Eliminate useless symbols, ϵ - productions and unit productions for the following grammar :

$$S \rightarrow 0A0|1B1|BB, A \rightarrow C, B \rightarrow S|A, C \rightarrow S|\epsilon$$

GTU : Summer-19, Marks 7

Solution : In given set of production rules, following are unit productions

$$\begin{array}{c} A \rightarrow C \\ C \rightarrow S \end{array} \quad \begin{array}{c} B \rightarrow S|A \end{array}$$

Similarly $A \rightarrow C \rightarrow \epsilon$ and $B \rightarrow A \rightarrow C \rightarrow \epsilon$ are ϵ - productions, on eliminating unit and null productions, we get,

$$S \rightarrow 0S0|1S1|11|00$$

⇒ Example 3.9.6 : Prove that - "If there is a CFG for the language L that has no λ productions," then there is a CFG for L with no λ - productions and no unit productions". Support your answer with the help of the following CFG. :

$$S \rightarrow A|bb$$

$$A \rightarrow B|b$$

$$B \rightarrow S|a$$

GTU : Winter-19, Marks 3

Solution : In the given grammar, there is no null production, but there exists unit productions as

$$S \rightarrow A$$

$$A \rightarrow B$$

And

$$B \rightarrow S$$

If we eliminate all the unit productions then, we get

$$S \rightarrow bb|b|a$$

3.10 Normal Forms

Normal Form of a grammar means arranging each production rule in some specific form. It is always necessary to simplify the production rules before converting them to any normal form. Simplifying the grammar means - i) eliminating the null or epsilon productions ii) Eliminating unit productions and iii) removing the unit productions from the grammar.

In this section we will discuss one important normal form i.e Chomsky's Normal Form(CNF).

3.10.1 Chomsky's Normal Form

The Chomsky's Normal Form can be defined as

Non terminal \rightarrow Non terminal · Non terminal
Non terminal \rightarrow Terminal

The given CFG should be converted in the above format then we can say that the grammar is in CNF. Before converting the grammar into CNF it should be in reduced form. That means remove all the useless symbols, ϵ productions and unit productions from it. Thus this reduced grammar can be then converted to CNF.

Let us solve some examples for Chomsky's Normal Form.

→ Example 3.10.1 : Convert the following CFG into CNF

$S \rightarrow aaaaS, \quad S \rightarrow aaaa$

Solution : As we know the rule for Chomsky's normal form is

Non terminal \rightarrow Non terminal · Non terminal

Non terminal \rightarrow Terminal

But the CFG given is

$S \rightarrow aaaaS \quad \text{rule 1}$

$S \rightarrow aaaa \quad \text{rule 2}$

If we add a rule

Then our rule 1 and 2 becomes,

$A \rightarrow AAAAS$

$S \rightarrow AAAA$

Let us take

$S \rightarrow A \boxed{AAAS}$ can be replaced by P_1

If we define

$P_1 \rightarrow AAAS$ then the rule becomes

$S \rightarrow A P_1$ which is in CNF

But the new rule P_1 is not CNF, so let us convert it

$P_1 \rightarrow A \boxed{AAS}$ can be replaced by P_2

then $P_1 \rightarrow A P_2$ which is in CNF

and $P_2 \rightarrow AAS$ which is not in CNF so let us convert it to CNF

$P_2 \rightarrow A \boxed{AS}$ can be replaced by P_3

$P_2 \rightarrow A P_3$ where $P_3 \rightarrow AS$

Now both P_2 and P_3 are in CNF.

Collectively rewrite these rules (these all indicate rule 1 of the given CPG).

$$\begin{array}{l} S \rightarrow A P_1 \\ P_1 \rightarrow A P_2 \\ P_2 \rightarrow A P_3 \\ P_3 \rightarrow AS \end{array}$$

Now consider rule 2

$S \rightarrow AAAA$

If we break the rule 2 as

$S \rightarrow \boxed{AA} \boxed{AA}$	
↓	↓
P_4	P_5

indicated by

The rule becomes

$S \rightarrow P_4 P_5$ which is in CNF.

But P_4 and P_5 indicate the same rule. So we can eliminate either of them.

Hence rule 2 becomes

$S \rightarrow P_4 P_4$

Finally we can collectively show the CFG converted to CNF as

$S \rightarrow A P_1$

$$P_1 \rightarrow A P_2$$

$$P_2 \rightarrow A P_3$$

$$P_3 \rightarrow A S$$

$$S \rightarrow P_4 P_4$$

$$P_4 \rightarrow AA$$

$$A \rightarrow a$$

→ Example 3.10.2 : Convert the given CFG to CNF $S \rightarrow aSa \mid bSb \mid a \mid b$

Solution : Let us start by adding new symbols for the terminals.

$$S \rightarrow ASA$$

$$A \rightarrow a$$

$$S \rightarrow BSB$$

$$B \rightarrow b$$

Note that although $s \rightarrow a$ and $A \rightarrow a$ are similar still we are not replacing any a by S . This is because, S is not simply giving a . It has other productions also. We want such a non terminal having only one production and that is a . Same is with B . Hence we have added the rules $A \rightarrow a$ and $B \rightarrow b$.

Let us take $S \rightarrow ASA$ for converting to CNF.

$$S \rightarrow A \boxed{SA}$$

replace it by A_1

$$S \rightarrow A A_1$$

$$A_1 \rightarrow SA$$

Take,

$$S \rightarrow B \boxed{SB}$$

replace it by A_2

$$S \rightarrow B A_2 \text{ both are in CNF}$$

$$A_2 \rightarrow SB$$

Hence we can write the rule in CNF as

$$S \rightarrow A A_1$$

$$A_1 \rightarrow SA$$

$$A \rightarrow a$$

$$S \rightarrow BA_2$$

$$A_2 \rightarrow SB$$

$$B \rightarrow b$$

$$S \rightarrow a$$

$$S \rightarrow b$$

Example 3.10.3 : Convert the given CFG to CNF.

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

Where $V = \{S, A; B\}$

$T = \{a, b\}$

P consists of

$$S \rightarrow aB \quad A \rightarrow bAA$$

$$S \rightarrow bA \quad B \rightarrow b$$

$$A \rightarrow a \quad B \rightarrow aS$$

$$A \rightarrow aS \quad B \rightarrow aBB$$

GTU : Summer-16, Marks 7

Solution : Let us start with first rule.

$$R_1 \rightarrow a$$

$S \rightarrow R_1B$ is in CNF for $S \rightarrow aB$

$$R_2 \rightarrow b$$

$S \rightarrow R_2A$ is in CNF for $S \rightarrow bA$

$$A \rightarrow a$$

$A \rightarrow aS$ can be written as

$$A \rightarrow R_1S$$

Now $A \rightarrow bAA$ can be written as

$A \rightarrow R_2 [AA]$ replace it by R_3

i.e. $R_3 \rightarrow AA$

then $A \rightarrow R_2R_3$

$$B \rightarrow a$$

$B \rightarrow bS$ can be written as

$$B \rightarrow R_2S$$
 is in CNF

Now, $B \rightarrow aBB$

i.e. $B \rightarrow R_1BB \quad \because R_1 \rightarrow a$

Let $R_4 \rightarrow BB$

then $B \rightarrow R_1 R_4$

Finally we can write,

$$S \rightarrow R_1 B$$

$$S \rightarrow R_2 A$$

$$A \rightarrow R_1 S$$

$$A \rightarrow R_2 R_3$$

$$B \rightarrow b$$

$$B \rightarrow R_2 S$$

$$B \rightarrow R_1 R_4$$

$$R_1 \rightarrow a$$

$$R_2 \rightarrow b$$

$$R_3 \rightarrow AA$$

$$R_4 \rightarrow BB$$

is a Chomsky's normal form.

Example 3.10.4 : Convert the following CFG to CNF

$$S \rightarrow ABA$$

$$A \rightarrow aA | \epsilon$$

$$B \rightarrow bB | \epsilon$$

Solution : In the Chomsky's normal form the ϵ production is not allowed. So first we will eliminate ϵ productions.

$$A \rightarrow \epsilon \text{ and } B \rightarrow \epsilon$$

If put ϵ instead of A and B

We can get,

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

$$\text{Similarly, } A \rightarrow aA | a$$

$$B \rightarrow bB | b$$

Now $S \rightarrow A$ and $S \rightarrow B$ is unit production getting introduced in the grammar after removal of ϵ production. So we will remove unit production also,

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

$$A \rightarrow aA | a$$

$$B \rightarrow bB | b$$

Now let us convert this grammar to Chomsky's normal form.

Let

$$S \rightarrow ABA$$

$$\boxed{S \rightarrow ABA}$$

$$\boxed{R_1 \rightarrow BA}$$

then

$$S \rightarrow AR_1$$

Now

$$\boxed{S \rightarrow AB} \quad \boxed{S \rightarrow BA} \quad \boxed{S \rightarrow AA} \quad \text{already in CNF}$$

Let

$$S \rightarrow aA$$

We will define $R_2 \rightarrow a$

$$\boxed{S \rightarrow R_2 A}$$

$$\boxed{R_2 \rightarrow a}$$

$$\boxed{S \rightarrow a}$$

already in CNF

$$S \rightarrow bB$$

Let

$$\boxed{R_3 \rightarrow b}$$

then

$$\boxed{S \rightarrow R_3 B}$$

$$S \rightarrow b \quad \text{already in CNF}$$

Now consider the rules,

$$\boxed{A \rightarrow aA}$$

We can rewrite it as

$$\boxed{A \rightarrow R_2 A}$$

$$\boxed{A \rightarrow a}$$

Also, $B \rightarrow bB$ can be rewritten as

$$B \rightarrow R_3 B$$

$$B \rightarrow b$$

Collect all the CNF's to integrate and we will get

$$S \rightarrow A R_1 | AB | BA | AA | R_2 A | R_3 B | a | b$$

$$A \rightarrow R_2 A | a$$

$$B \rightarrow R_3 B | b$$

$$R_1 \rightarrow BA$$

$$R_2 \rightarrow a$$

$$R_3 \rightarrow b$$

is done.

Example 3.10.5 : Convert the following grammar to Chomsky's normal form
 $S \rightarrow -S | [S \supset S] | p | q$

Solution : Chomsky's normal form is -

Non terminal \rightarrow Non terminal · Non terminal

Non terminal \rightarrow Terminal

Consider given grammar rule by rule and let us convert it into CNF

$$S \rightarrow -S$$

Can be written as

$$S \rightarrow AS$$

$$A \rightarrow -$$

Similarly

$$S \rightarrow [S \supset S]$$

Can be written as

$$S \rightarrow BC$$

$$B \rightarrow DE$$

$$E \rightarrow SF$$

$$D \rightarrow I$$

$$F \rightarrow D$$

$$C \rightarrow SG$$

$$G \rightarrow I$$

And,

$$S \rightarrow p$$

$$S \rightarrow q$$

is already in CNF.

Hence the complete grammar written in CNF is -

$$S \rightarrow AS$$

$$A \rightarrow \sim$$

$$S \rightarrow BC$$

$$B \rightarrow DE$$

$$E \rightarrow SF$$

$$D \rightarrow [$$

$$F \rightarrow \square$$

$$C \rightarrow SG$$

$$G \rightarrow]$$

$$S \rightarrow p$$

$$S \rightarrow q$$

→ Example 3.10.6 : Given the CFG G, find a CFG G' in Chomsky Normal form generating L(G) - {A}

$$S \rightarrow AaA \mid CA \mid BaB$$

$$A \rightarrow aaBa \mid CDA \mid aa \mid DC$$

$$B \rightarrow bB \mid bAB \mid bb \mid aS$$

$$C \rightarrow Ca \mid bC \mid D$$

$$D \rightarrow bD \mid A$$

GTU : Summer-12, 18, Marks 7

Solution : We will simplify given CFG first by removing null productions and by eliminating unit productions.

$$S \rightarrow AaA \mid CA \mid BaB$$

$$A \rightarrow aaBa \mid CDA \mid aa \mid DC$$

$$B \rightarrow bB \mid bAB \mid bb \mid aS$$

$$C \rightarrow Ca \mid bC \mid bD \mid b$$

$$D \rightarrow bD \mid b$$

Now we will convert above grammar to Chomsky's normal form. This form is

Non-terminal → Non-terminal Non-terminal | terminal

1. Let,

$$S \rightarrow AaA \mid CA \mid BaB$$

becomes

$$S \rightarrow R_1 A \mid CA \mid R_2 B$$

$$R_1 \rightarrow AP \text{ and } R_2 \rightarrow BP$$

$$P \rightarrow a$$

$$S \rightarrow R_1 A \mid CA \mid R_2 B$$

$$R_1 \rightarrow AP$$

$$R_2 \rightarrow BP$$

$$P \rightarrow a$$

2. Let,

$$B \rightarrow bB \mid bAB \mid bb \mid aS$$

gives

$$B \rightarrow QB$$

$$B \rightarrow QQ$$

$$Q \rightarrow b$$

$$B \rightarrow PS$$

$$B \rightarrow R_3 B$$

$$R_3 \rightarrow QA$$

3. Let,

$$C \rightarrow Ca \mid bC \mid bD \mid b$$

gives

$$C \rightarrow CP \mid QC \mid QD$$

$C \rightarrow b$ is already in CNF.

4. Let,

$$D \rightarrow bD \mid b$$

gives

$$D \rightarrow QD$$

$D \rightarrow b$ is already in CNF.

→ Example 3.10.7 : Given the CFG G, find a CFG G' in Chomsky normal form generating

$$L(G) = \{\Lambda\}$$

$$S \rightarrow A \mid B \mid C$$

$$A \rightarrow aAa \mid B$$

$$B \rightarrow bB \mid bb$$

$$C \rightarrow aCaa \mid D$$

$$D \rightarrow baD \mid abD \mid aa$$

GTU : Summer-14. Marks 7

Solution : Before converting given CFG to CNF we will simplify the given grammar by eliminating unit productions.

Rule	Simplification
$C \rightarrow aCaa \mid D$	$C \rightarrow aCaa \mid baD \mid abD \mid aa$
$A \rightarrow aAa \mid B$	$A \rightarrow aAa \mid bB \mid bb$
$S \rightarrow A \mid B \mid C$	$S \rightarrow aAa \mid bB \mid bb \mid aCaa \mid baD \mid abD \mid aa$

The conversion to CNF is as follows :

Rule	CNF rule
$S \rightarrow aAa$	$S \rightarrow XY$ $X \rightarrow ZA$ $Y \rightarrow AZ$ $Z \rightarrow a$
$S \rightarrow bB$	$S \rightarrow TB$ $T \rightarrow b$
$S \rightarrow bb$	$S \rightarrow TT$
$S \rightarrow aCaa$	$S \rightarrow ZW$ $W \rightarrow CP$ $P \rightarrow ZZ$
$S \rightarrow baD$	$S \rightarrow TQ$ $Q \rightarrow ZD$
$S \rightarrow abD$	$S \rightarrow ZR$ $R \rightarrow TD$
$S \rightarrow aa$	$S \rightarrow ZZ$
$A \rightarrow aAa$	$A \rightarrow XY$
$A \rightarrow bB$	$A \rightarrow TB$
$A \rightarrow bb$	$A \rightarrow TT$
$B \rightarrow bB$	$B \rightarrow TB$
$B \rightarrow bb$	$B \rightarrow TT$
$C \rightarrow aCaa$	$C \rightarrow ZW$
$C \rightarrow baD$	$C \rightarrow TQ$
$C \rightarrow abD$	$C \rightarrow ZR$
$C \rightarrow aa$	$C \rightarrow ZZ$

► Example 3.10.8 : Given the context free grammar G , find a CFG G' in Chomsky Normal Form generating $L(G) - \{\}$

$$1) S \rightarrow aY \mid Ybb \mid Y$$

$$X \rightarrow \wedge \mid a$$

$$Y \rightarrow aYY \mid bb \mid XXa$$

$$2) S \rightarrow AA$$

$$A \rightarrow B \mid BB$$

$$B \rightarrow abB \mid b \mid bb$$

GTU : Summer-15, Winter-17, Marks 7

Solution : 1) We will eliminate \wedge and unit productions from given grammar. First we will eliminate unit production $S \rightarrow Y$. It will be -

$$S \rightarrow aY \mid Ybb \mid aYY \mid bb \mid XXa$$

$$X \rightarrow \wedge \mid a$$

$$Y \rightarrow aYY \mid bb \mid XXa$$

Now eliminating \wedge

$$S \rightarrow aY \mid Ybb \mid aYY \mid bb \mid a \mid Xa \mid XXa$$

$$X \rightarrow a$$

$$Y \rightarrow aYY \mid bb \mid Xa \mid a$$

The Chomsky's Normal Form will be

$$S \rightarrow aY \xrightarrow{\text{becomes}} S \rightarrow XY$$

$$S \rightarrow Ybb \xrightarrow{\text{becomes}} S \rightarrow YZ$$

$$Z \rightarrow BB$$

$$B \rightarrow b$$

$$S \rightarrow aYY \xrightarrow{\text{becomes}} S \rightarrow XT$$

$$T \rightarrow YY$$

$$S \rightarrow bb \xrightarrow{\text{becomes}} S \rightarrow BB$$

$$S \rightarrow XXa \xrightarrow{\text{becomes}} S \rightarrow PX$$

$$P \rightarrow XX$$

$X \rightarrow a$ is already in CNF

$$Y \rightarrow aYY \xrightarrow{\text{becomes}} Y \rightarrow XT$$

$$T \rightarrow YY$$

$$Y \rightarrow bb \xrightarrow{\text{becomes}} Y \rightarrow BB$$

$$Y \rightarrow Xa \xrightarrow{\text{becomes}} Y \rightarrow XX$$

$Y \rightarrow a$ is already in CNF.

2) In this grammar there is unit production $A \rightarrow B$. We will eliminate it first.

$$S \rightarrow AA$$

$$A \rightarrow BB \mid abB \mid b \mid bb$$

$$B \rightarrow abB \mid b \mid bb$$

The CNF will be

$$S \rightarrow AA \text{ is already in CNF}$$

$$A \rightarrow BB \text{ is already in CNF}$$

$$\begin{aligned} A \rightarrow abB &\xrightarrow{\text{becomes}} A \rightarrow XB \\ &\quad X \rightarrow PQ \\ &\quad P \rightarrow a \\ &\quad Q \rightarrow b \end{aligned}$$

$$A \rightarrow b \text{ is already in CNF}$$

$$A \rightarrow bb \xrightarrow{\text{becomes}} A \rightarrow QQ$$

$$B \rightarrow abB \xrightarrow{\text{becomes}} B \rightarrow XB$$

$$B \rightarrow b \text{ is already in CNF}$$

$$B \rightarrow bb \xrightarrow{\text{becomes}} B \rightarrow QQ$$

Example 3.10.9 : Convert the CFG, $G(\{S, A, B\}, \{a, b\}, P, S)$ to CNF, where P is as follows $S \rightarrow aAbB \mid A \rightarrow Ab \mid b \mid B \rightarrow Ba \mid a$.

GTU : Winter-16, Marks 7

Solution : Chomsky Normal Form is

Non-terminal \rightarrow Non-terminal \cdot Non-terminal

Non-terminal \rightarrow terminal

We will convert each production rule to CNF form one by one

Production Rule	CNF
$S \rightarrow aAbB$	$S \rightarrow XY$ $X \rightarrow PA$ $P \rightarrow a$ $Y \rightarrow QB$ $Q \rightarrow b$
$A \rightarrow Ab$	$A \rightarrow AQ$
$A \rightarrow b$	It is already in CNF
$B \rightarrow Ba$	$B \rightarrow BP$ $P \rightarrow a$
$B \rightarrow a$	It is already in CNF

→ Example 3.10.10 : What is CNF ? Convert the following CFG into CNF.

$$S \rightarrow ASA \mid aB$$

$$A \rightarrow B \mid S,$$

$$A \rightarrow b \mid \epsilon$$

GTU : Winter18, Marks 7

Solution : Let, the grammar be

$$S \rightarrow ASA \mid aB$$

$$A \rightarrow B \mid S$$

$$B \rightarrow b \mid \epsilon$$

Before the conversion, we should eliminate ϵ and unit productions

$$S \rightarrow ASA \mid aB \mid a$$

$$A \rightarrow b \mid ASA \mid aB \mid a$$

$$B \rightarrow b$$

The conversion to CNF is

$S \rightarrow ASA$	$S \rightarrow AP$
	$P \rightarrow SA$
$S \rightarrow aB$	$S \rightarrow QB$
	$Q \rightarrow a$
$S \rightarrow a$	
$A \rightarrow b$	
$A \rightarrow ASA$	$A \rightarrow AP$
$A \rightarrow aB$	$A \rightarrow QB$
$A \rightarrow a$	
$B \rightarrow b$	

Review Questions

1. Give definition of context free grammars.

GTU : Winter-11, Marks 2

2. Explain the term - Regular grammar.

GTU : Summer-11, Marks 3, Winter-11, Marks 2

3.11 Short Questions and Answers

Q.1 Context free grammar is _____.

- a compiler b language expression
 c language translator d none of these

[Ans. : b]

Q.2 The CFG stands for _____.

Ans. : Context Free Grammar.

Q.3 Which of the following does not belong to context free grammar ?

- a Non terminal symbol b Terminal symbol
 c Start symbol d End symbol

[Ans. : d]

Q.4 Regular grammar is _____.

- a context free grammar b non context free grammar
 c simple english grammar d none of theses

[Ans. : a]

Q.5 $S \rightarrow aSa | bSb | a | b$; The language generated by the above grammar over the alphabet {a,b} is the set of _____.

- a all palindromes b all odd length palindromes.
 c strings that begin and end with the same symbol
 d all even length palindromes

[Ans. : d]

Q.6 The CFG $S \rightarrow aS | bS | a | b$ is equivalent to regular expression

- a $(a + b)$ b $(a + b)(a + b)^*$
 c $(a + b)(a + b)$ d none of these

[Ans. : b]

Explanation : The rules $S \rightarrow aS | bS$ generates $(a + b)^*$. In the given list of production rules $S \rightarrow a | b$ are the two rules. Hence the given CFG represents the regular expression $(a + b)(a + b)^*$.

Q.7 Context free grammar is closed under _____.

- a union b kleen star
 c concatenation d all of them

[Ans. : d]

Q.8 What is the use of context free grammar in compiler ?

Ans. : The context free grammar is used for checking the correctness of syntax of programming statements.

Q.9 In CFG normally the terminals are denoted by _____ case and non terminals are denoted by _____ case.

Ans. : lower, upper

Q.10 The language $\{a^m b^m c^{m+n} \mid m, n \geq 1\}$ is _____.

- a regular b context free but non regular
 c context sensitive but not context free
 d none of these

[Ans. : b]

Q.11 The context free grammar $S \in \{S \mid 0S1 \mid 1S0 \mid \epsilon\}$ generates _____.

- a equal number of 0's and 1's
 b unequal number of 0's and 1's
 c any number of 0's followed by any number of 1's
 d none of these

[Ans. : a]

Q.12 Which of the following is format of unit production ?

- a $A \rightarrow B$ b $A \rightarrow a$
 c $A \rightarrow \epsilon$ d All of these

[Ans. : a]



Pushdown Automata, CFL and NCFL

4.1 Definition of Push Down Automata (PDA)

The PDA can be defined as a collection of seven components.

1. The finite set of states Q .
2. The input set Σ .
3. Γ is a stack alphabet.
4. q_0 is initial stage, $q_0 \in Q$.
5. Z_0 is a start symbol which is in Γ .
6. Set of final states $F \subseteq Q$.
7. δ is mapping function used for moving from current state to next state.

 Example 4.1.1 : Compare PDA with FSM.

GTU : Summer-19, Marks 3

Solution :

Sr. No.	FSM	PDA
1.	FSM stands for finite machine.	PDA stands for push down automata.
2.	It does not have capability to remember previous input.	As PDA contains stack, it can remember already read input.
3.	FSM is constructed for type-3 grammar.	PDA is constructed for type-2 grammar.
4.	NFSM and FSM are equivalent.	NPDA has more capability than PDA.

Review Question

1. Define Push Down Automata (PDA).

GTU : Summer-16, Marks 2

4.2 Deterministic PDA

The deterministic Pushdown automata is a kind of pushdown automata which shows unique move for each corresponding input.

We can describe the behaviour of PDA by means of instantaneous description. It is given by ID. The moves of ID's are as shown below.

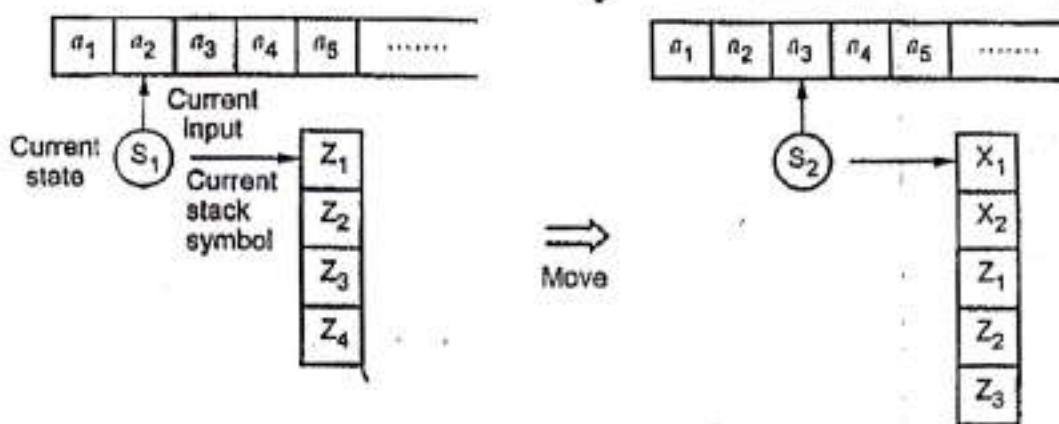


Fig. 4.2.1

From Fig. 4.2.1, if we are reading the current symbol a_2 , at current state S_1 and current stack symbol Z_1 then after a move we will reach to state S_2 and there will be some new symbol on the top of the stack. This description can be represented as -

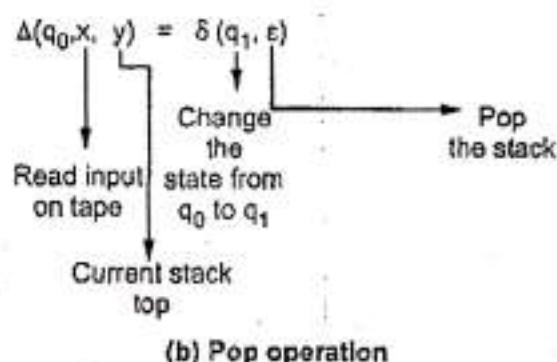
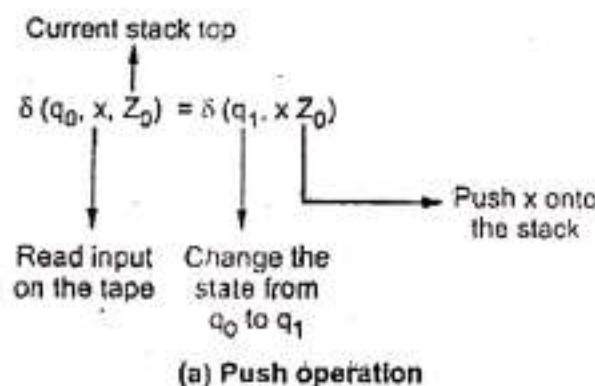


Fig. 4.2.2 Instantaneous description

Now let us solve some examples based on it.

→ **Example 4.2.1 :** Design a PDA for accepting a language $\{L = a^n b^n \mid n \geq 1\}$.

Solution : This is a language in which equal number of a 's are followed by equal number of b 's. The logic for this PDA can be applied as : first we will push all a 's onto the stack. Then on reading every single b each a is popped from the stack. If we read all b and remove all a 's and if we get stack empty then that string will be accepted. The instantaneous description can be given as -

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$

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

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

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

$$\delta(q_1, \epsilon, Z_0) = (q_2, \epsilon)$$

where q_0 is a start state and q_2 is accept state.

We will simulate this PDA for following string -

$$\begin{aligned}
 (q_0, aabb, Z_0) &\vdash (q_0, abbb, aZ_0) \\
 &\vdash (q_0, abbb, aaZ_0) \\
 &\vdash (q_0, bbb, aaaZ_0) \\
 &\vdash (q_1, bb, aaZ_0) \\
 &\vdash (q_1, b, aZ_0) \\
 &\vdash (q_1, \epsilon, Z_0) \\
 &\vdash (q_2, \epsilon)
 \end{aligned}$$

ACCEPT state.

Example 4.2.2 : Construct PDA for the language $L = \{a^n b^{2n} \mid n \geq 1\}$.

Solution : In this language n number of a 's should be followed by $2n$ number of b 's. Hence we will apply a very simple logic and that is if we read single ' a ' we will push two ' a 's onto the stack. As soon as we read ' b ' then for every single ' b ' only one ' a ' should get popped from the stack. This basically maintains the $a^n b^{2n}$ count and sequence. The ID can be constructed as follows -

$$\delta(q_0, a, Z_0) = (q_0, aaZ_0)$$

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

Now when we read b we will change the state from q_0 to q_1 and start popping corresponding ' a '. Hence,

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

Thus this process of popping will be repeated unless all the symbols are read. Note that popping action occurs in state q_1 only.

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

After reading all b 's all the corresponding a 's should get popped. Hence when we read ϵ as input symbol there should be nothing in the stack. Hence the move will be -

$$\delta(q_1, \epsilon, Z_0) = (q_2, \epsilon)$$

where the PDA $P = (Q, q_0, q_1, q_2, \Sigma, \Gamma, \delta)$

We can summarize the ID as

$$\delta(q_0, a, Z_0) = (q_0, aaZ_0)$$

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

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

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

$$\delta(q_1, \epsilon, Z_0) = (q_2, \epsilon)$$

Let us simulate this PDA for some input string "aaabbbbbbb".

$$\delta(q_0, aaabbbbbbb, Z_0) \vdash (q_0, abbbbbbb, aaZ_0)$$

$$\vdash (q_0, abbbbbbb, aaaaZ_0)$$

$$\vdash (q_0, bbbbbbb, aaaaaaZ_0)$$

$$\vdash (q_1, bbbbb, aaaaaZ_0)$$

$$\vdash (q_1, bbbb, aaaaZ_0)$$

$$\vdash (q_1, bbb, aaaZ_0)$$

$$\vdash (q_1, bb, aaZ_0)$$

$$\vdash (q_1, b, aZ_0)$$

$$\vdash (q_1, \epsilon, Z_0)$$

$$\vdash (q_2, \epsilon)$$

Final state or ACCEPT state.

Thus input gets accepted by using the constructed PDA.

Example 4.2.3 : Design PDA to accept the language

$$L = \{w \mid w \in (a + b)^* \text{ and } n_a(w) = n_b(w)\}.$$

Solution : This is a language of equal number of 'a's and equal number of 'b's. Initially when stack is empty then whatever we read either 'a' or 'b' we will simply push it onto the stack. Now if we read 'a' and at the top of the stack if 'b' is present then it will be erased by ϵ . (i.e. we pop it) Similarly if we read 'b' and top of the stack contains 'a' then erase it by ϵ . The instantaneous description for this language is as given below.

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$

$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$

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

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

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

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

$$\delta(q_0, \epsilon, Z_0) = (q_1, Z_0)$$

where q_0 is a start state and q_1 is a final state. When we reach to this state with ϵ input and having an empty stack. We move to accept/final state. Let us simulate this for a string 'aabbbb'.

$$\delta(q_0, aabb, Z_0) \vdash (q_0, ababb, aZ_0)$$

$$\vdash (q_0, babbb, aaZ_0)$$

$$\vdash (q_0, abb, aZ_0)$$

$$\vdash (q_0, bb, aaZ_0)$$

$$\vdash (q_0, b, aZ_0)$$

$$\vdash (q_0, \epsilon, Z_0)$$

$$\vdash (q_1, Z_0)$$

ACCEPT state.

→ Example 4.2.4 : Design a PDA for the language

$$L = \{w \mid w \in (a + b)^* \text{ and } n_a(w) > n_b(w)\}.$$

Solution : $n_a(w)$ means total number of 'a's in input string and $n_b(w)$ means total number of 'b's in input string. The problem states that total number of 'a's are more than total number of 'b's in input string. The logic for this PDA will be -

If we read 'a' or 'b' we will simply push it onto the stack. If the stack top has a symbol 'a' and we read 'a', then also push it onto the stack. Same is true for 'b'. But if we read 'a' and stack top symbol is 'b' then pop. Also if we read 'b' and stack top symbol is 'a' then pop the stack. Finally if we read ϵ (i.e. Complete string is read) then stack should contain 'a' on the top. This means that total number of 'a's are more than total number of 'b's. The ID can be

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$

$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$

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

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

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

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

$$\delta(q_0, \epsilon, a) = (q_f, a)$$

where $P = (Q_0, Q_f, \{a, b\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_f\})$

Simulation : We will take some string to simulate this PDA for some input string.
Consider the input $abbabab$.

$$\begin{aligned}\delta(q_0, abbabab, Z_0) &\vdash (q_0, abbabab, aZ_0) \\ &\vdash (q_0, babab, aZ_0) \\ &\vdash (q_0, abab, aZ_0) \\ &\vdash (q_0, bab, aaZ_0) \\ &\vdash (q_0, ab, aZ_0) \\ &\vdash (q_0, b, aaZ_0) \\ &\vdash (q_0, \epsilon, aZ_0) \\ &\vdash (q_f, a)\end{aligned}$$

Accept state.

Thus input gets accepted.

→ **Example 4.2.5 :** Design PDA for the language that accepts strings with $n_a(w) < n_b(w)$ where $w \in (a + b)^*$.

Solution : This problem is similar to previous problem. The only difference is that in this problem the language requires more number of b's than total number of a's.

Hence the ID will be -

$$\begin{aligned}\delta(q_0, a, Z_0) &= (q_0, aZ_0) \\ \delta(q_0, b, Z_0) &= (q_0, bZ_0) \\ \delta(q_0, a, a) &= (q_0, aa) \\ \delta(q_0, b, b) &= (q_0, bb) \\ \delta(q_0, b, a) &= (q_0, b) \\ \delta(q_0, a, b) &= (q_0, \epsilon) \\ \delta(q_0, \epsilon, b) &= (q_f, b) \leftarrow \text{The b's are more in number.}\end{aligned}$$

Now we will simulate this PDA for the input "abbab".

$$\begin{aligned}\delta(q_0, abbab, Z_0) &\vdash (q_0, bbab, aZ_0) \\ &\vdash (q_0, bab, Z_0) \\ &\vdash (q_0, ab, bZ_0)\end{aligned}$$

$$\vdash (q_0, b, Z_0)$$

$$\vdash (q_0, \epsilon, bZ_0)$$

$$\vdash (q_f, b)$$

Accept state.

Example 4.2.6 : Design and draw a deterministic PDA accepting "Balanced strings of Brackets" which are accepted by following CFG.

$$S \rightarrow SS \mid \{ S \} \mid (S) \mid \epsilon$$

GTU : Summer-11, Marks 6

Solution : The problem of checking well formedness of parenthesis is similar to the problem of checking equal number of a's and equal number of b's. But always (, [or { i.e. left parenthesis come first and then corresponding),] or } i.e. right parenthesis appear. This is what is a meaning of well-formedness. The ID can be constructed as below :

Initially i.e. in state q_0 either (, [, or { will be scanned and at that time the stack will be empty. Then we will simply push it onto stack.

$$\therefore \delta(q_0, (, Z_0) \vdash (q_1, (Z_0))$$

$$\delta(q_0, [, Z_0) \vdash (q_1, [Z_0))$$

$$\delta(q_0, {, Z_0) \vdash (q_1, {Z_0})}$$

Further if we read more left parenthesis then we will simply push them onto the stack.

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

$$\delta(q_1, ([)) = (q_1, ([))$$

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

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

Thus if we read any opening parenthesis we go on pushing it onto the stack. But as soon as we read closing parenthesis or right parenthesis we start popping the stack contents.

$$\therefore \delta(q_1, ((),)) = (q_1, \epsilon)$$

$$\delta(q_1, ((),)) = (q_1, \epsilon)$$

$$\delta(q_1, ((),)) = (q_1, \epsilon)$$

After reading the complete input string we will get ϵ to read but at the same time the contents of stack should be removed and it should simply contain Z_0 . At this time the PDA should enter the final state and that too in state q_0 , so that all steps can be repeated if needed. This transition can be as shown -

$$\delta(q_1, \epsilon, Z_0) = (q_0, Z_0)$$

To summarize these steps we will write ID as -

$$\delta(q_0, (, Z_0) = (q_1, (Z_0))$$

$$\delta(q_0, [, Z_0) = (q_1, [Z_0))$$

$$\delta(q_0,), (Z_0) = (q_1, (Z_0))$$

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

$$\delta(q_1, [, ()) = (q_1, ([))$$

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

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

$$\delta(q_1, [, ()) = (q_1, [))$$

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

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

$$\delta(q_1, [, ()) = (q_1, [))$$

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

$$\text{The PDA } P = (q_0, q_1, \{ (, (),), , \}, \{ L, R, Z_0 \}, \delta, c_0, Z_0, \{ q_0 \})$$

Let us simulate it for some input string.

(())

$$\delta(q_0, (()), Z_0) \vdash (q_1, (()), (Z_0))$$

$$\vdash (q_1, (()), ((Z_0))$$

$$\vdash (q_1, (()), (Z_0))$$

$$\vdash (q_1, (()), ((Z_0))$$

$\vdash (q_1, , (Z_0))$
 $\vdash (q_1, \epsilon, Z_0)$
 (q_0, Z_0)
 ACCEPT state.

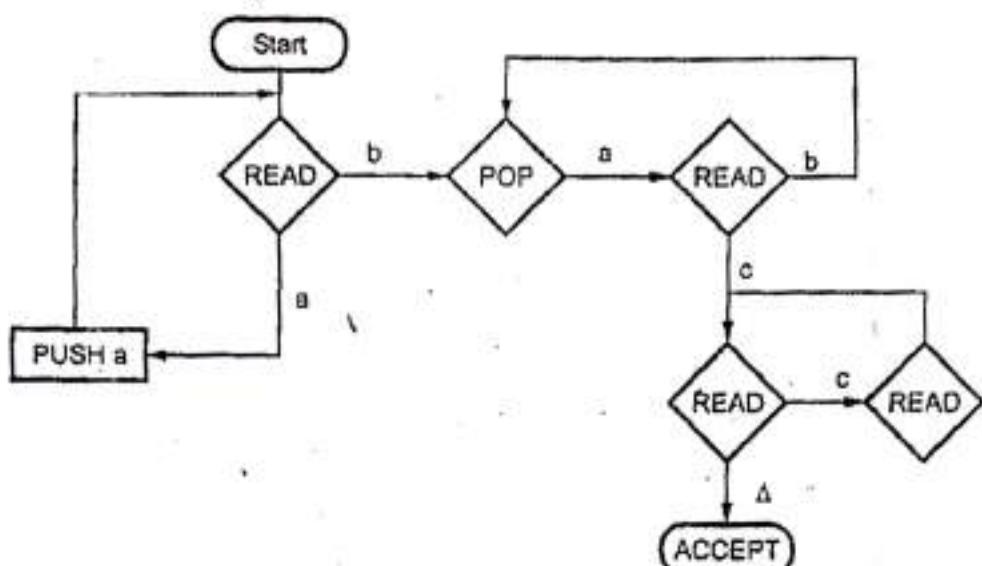
Example 4.2.7 : Construct PDA for the language $\{L = a^m b^m c^n \mid m, n \geq 1\}$.

Solution : This is the language in which all the a's are followed by equal number of b's followed by any number of c's. The simple logic that we can apply is : As we read go on pushing each a onto the stack. As soon as we read b, pop one a from the stack. Repeat this process while reading all b's. Now when we read c, simply read c and do nothing, because number of c's are arbitrary in given language. The Instantaneous Description (ID) for this logic can be as follows.

$$\begin{aligned}
 \delta(q_0, a, z_0) &= \delta(q_0, aZ_0) \\
 \delta(z_0, a, a) &= \delta(q_0, aa) \\
 \delta(z_0, b, a) &= \delta(q_1, \epsilon) \\
 \delta(z_1, b, a) &= \delta(q_1, \epsilon) \\
 \delta(q_1, c, Z_0) &= \delta(q_1, Z_0) \\
 \delta(q_1, \epsilon, Z_0) &= \delta(q_2, Z_0)
 \end{aligned}$$

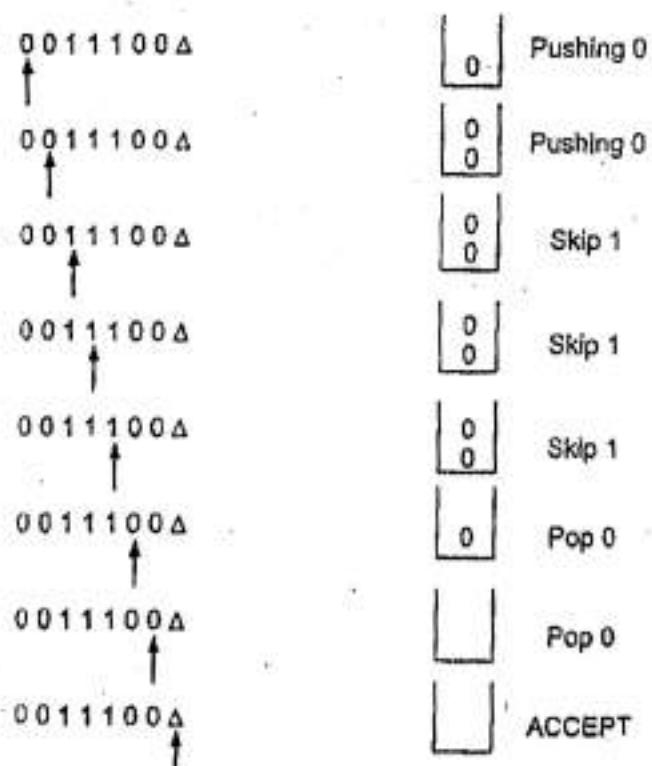
Let us derive this PDA for the same example.

$\delta(q_0, aabbcc, Z_0) \vdash \delta(q_0, abbccc, aZ_0)$
 $\vdash \delta(q_0, bbccc, aaZ_0)$
 $\vdash \delta(q_1, bccc, aZ_0)$
 $\vdash \delta(q_1, ccc, Z_0)$
 $\vdash \delta(q_1, cc, Z_0)$
 $\vdash \delta(q_1, c, Z_0)$
 $\vdash \delta(q_1, \epsilon, Z_0)$
 $\vdash \delta(q_2, Z_0)$
 ACCEPT

Fig. 4.2.3 PDA for $a^m b^n c^n$

→ Example 4.2.8 : Build a PDA for the language $L = \{0^m 1^n 0^n \mid m, n \geq 1\}$ by empty stack.

Solution : In this PDA n number of 0's are followed by any number of 1's followed by n number of 0's. Hence the logic for design of such PDA will be as follows. Push all 0's onto the stack on encountering first zeros. Then if we read 1, just do nothing. Then read 0 and on each read of 0, pop one 0 from the stack. For instance :



This scenario can be written in the ID form as

$$\begin{aligned}\delta(q_0, 0, Z_0) &= \delta(q_0, 0Z_0) \\ \delta(q_0, 0, 0) &= \delta(q_0, 00) \\ \delta(q_0, 1, 0) &= \delta(q_1, 0) \\ \delta(q_0, 1, 0) &= \delta(q_1, 0) \\ \delta(q_1, 0, 0) &= \delta(q_1, \epsilon) \\ \delta(q_0, \epsilon, Z_0) &= \delta(q_2, Z_0) \leftarrow \text{ACCEPT state}\end{aligned}$$

For example :

$$\begin{aligned}\delta(q_0, 0011100, Z_0) &\vdash \delta(q_0, 011100, 0Z_0) \\ &\vdash \delta(q_0, 11100, 00Z_0) \\ &\vdash \delta(q_0, 1100, 00Z_0) \\ &\vdash \delta(q_1, 100, 00Z_0) \\ &\vdash \delta(q_1, 00, 00Z_0) \\ &\vdash \delta(q_1, 0, 0Z_0) \\ &\vdash \delta(q_1, \epsilon, Z_0) \\ &\vdash \delta(q_2, Z_0)\end{aligned}$$

ACCEPT

→ Example 4.2.9 : For the language $L = \{ xcx^r / x \in \{a,b\}^* \}$ design a PDA (Push Down Automata) and trace it for string "abcbab".

GTU : Summer-11, Marks 8; Winter-12, 17, Marks 7

Solution : The required PDA will be -

$$\text{PDA} = (\{q_0, q_1, q_2\}, \{a, b, C\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

The mapping of δ is -

$$\begin{aligned}\delta(q_0, a, Z_0) &= (q_0, a, Z_0) \\ \delta(q_0, b, Z_0) &= (q_0, b, Z_0) \\ \delta(q_0, a, a) &= (q_0, a a) \\ \delta(q_0, b, a) &= (q_0, b a) \\ \delta(q_0, a, b) &= (q_0, a b) \\ \delta(q_0, b, b) &= (q_0, b b)\end{aligned}$$

$$\delta(q_0, C, Z_0) = (q_1, Z_0)$$

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

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

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

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

$$\delta(q_1, \epsilon, Z_0) = (q_2, \Sigma_0)$$

The required CFG can be $G = (\{S\}, \{a, b\}, P, S)$. The $P = \{ S \rightarrow aSb \mid bSa \mid C \}$ is the production rules.

String read	Action
a b c b a ↑	Read a, push it onto the stack, remain in q_0 state.
a b c b a ↑	Read b, push it onto the stack, be in q_0 state.
a b c b a ↑	Just read out c and change state from q_0 and q_1 .
a b c b a ↑	Read b, POP b.
a b c b a ↑	Read a, POP a.
a b c b a ↑	Input ends POP blank symbol from stack and ACCEPT.

Example 4.2.10 : Give transition table for deterministic PDA recognizing the following language.

$$\{ a^n b^{n+m} a^m \mid n, m \geq 0 \}$$

GTU : Winter-11, Marks 7

Solution : The logic for building this PDA is as follows - Read a's and go on pushing each onto the stack. On reading each b, pop one a from the stack. There will be a situation that stack becomes empty. Now when you read b's go on pushing them onto the stack. Next when the stream of a comes, on reading each a, pop corresponding b. Finally on reading Δ , the stack should be empty.

The Instantaneous Description (ID) will be -

$$\delta(q_0, a, Z_0) = \delta(q_0, aZ_0) \quad \text{Push a onto the stack}$$

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

$$\delta(q_0, b, a) = \delta(q_1, \epsilon) \quad \text{on reading b, pop a.}$$

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

$\delta(q_1, b, Z_0) = \delta(q_2, bZ_0)$	push b onto the stack
$\delta(q_2, b, b) = \delta(q_2, bb)$	
$\delta(q_2, a, b) = \delta(q_3, \epsilon)$	on reading a, pop b
$\delta(q_3, a, b) = \delta(q_3, \epsilon)$	
$\delta(q_3, \epsilon, Z_0) = \delta(q_4, \epsilon)$	Accept

The transition table will be

Current state	Current Input	Next state	Stack
Start			Z_0
q_0	a	q_0	aZ_0
q_0	a	q_0	aa
q_0	b	q_1	a (i.e. pop a)
q_1	b	q_1	a (i.e. pop a)
q_1	b	q_2	bZ_0 (i.e. push b)
q_2	b	q_2	bb
q_2	a	q_3	b (i.e. pop b)
q_3	a	q_3	b (i.e. pop b)
q_3	ϵ	q_4	Z_0
q_4	ACCEPT STATE		

→ Example 4.2.11 : Give transition table for deterministic PDA recognizing the following language.
 $\{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } j = k\}$

GTU : Winter-11, Summer-13, 16, Winter-14, Marks 7

Solution : There are two cases for this PDA.

Case 1 : $i = j$

According to this condition, the logic for building PDA will be as follows -

Read a, push it onto the stack. When you start reading b, pop one a for every single b. When you start reading c, the stack must be empty. Just go on reading c because there are any number of c's in the language.

The ID for this logic will be

$$\delta(q_0, a, Z_0) = \delta(q_0, aZ_0)$$

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

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

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

$$\delta(q_1, c, Z_0) = \delta(q_2, \epsilon)$$

$$\delta(q_2, c, Z_0) = \delta(q_2, \epsilon)$$

$$\delta(q_2, \epsilon, Z_0) = \delta(q_3, \epsilon) \quad \text{ACCEPT state}$$

The transition table will be :

Current state	Current input	Next state	Stack
Start	-	-	Z_0
q_0	a	q_0	aZ_0
q_0	a	q_0	aa
q_0	b	q_1	ϵ (i.e. pop a)
q_1	b	q_1	ϵ (i.e. pop a)
q_1	c	q_2	Z_0
q_2	c	q_3	Z_0
q_3 ACCEPT STATE	Δ	q_3	Z_0

Case 2 : $j = k$

The logic for this PDA will be just go on reading a's. When you read out b, just push it onto the stack. As soon as c is read pop b for each corresponding c.

The ID will be

$$\delta(q_0, a, Z_0) = \delta(q_0, Z_0)$$

$$\delta(q_0, b, Z_0) = \delta(q_1, bZ_0)$$

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

$$\delta(q_1, c, b) = \delta(q_2, \epsilon)$$

$$\delta(q_2, c, b) = \delta(q_2, \epsilon)$$

$$\delta(q_2, \epsilon, Z_0) = \delta(q_3, \epsilon) \quad \text{ACCEPT STATE}$$

The transition table will be

Current state	Current Input	Next state	Stack
Start			Z_0
q_0	a	q_0	Z_0
q_0	b	q_1	bZ_0
q_1	b	q_1	bb
q_1	c	q_2	ϵ i.e. pop b
q_2	c	q_2	ϵ
q_2	ϵ	q_3	Z_0

⇒ Example 4.2.12 : Design and draw a deterministic PDA accepting strings with more a's than b's. Trace it for the string "abbabaa".

GTU : Winter-12, Marks 6, Summer-16, Marks 7

Solution : PDA : Refer example 4.2.4.

Tracing for the string abbabaa

$$\begin{aligned}
 \delta(q_0, abbabaa, Z_0) &\vdash (q_0, bbabaa, aZ_0) \\
 &\vdash (q_0, babaa, Z_0) \\
 &\vdash (q_0, abaa, bZ_0) \\
 &\vdash (q_0, baa, bZ_0) \\
 &\vdash (q_0, aa, Z_0) \\
 &\vdash (q_0, a, Z_0) \\
 &\vdash (q_0, \epsilon, aZ_0) \\
 &\vdash (q_f, a)
 \end{aligned}$$

⇒ Example 4.2.13 : Prove : The language $pal = \{x \in \{a,b\}^* \mid x=x'\}$ cannot be accepted by any deterministic pushdown automaton.

GTU : Summer-14, Marks 7

Solution : To prove that given language pal can not be accepted by deterministic pushdown automaton, we obtain the instantaneous description for this PDA.

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$

$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$

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

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

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

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

$$\delta(q_0, \epsilon, Z_0) = (q_0, Z_0)$$

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

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

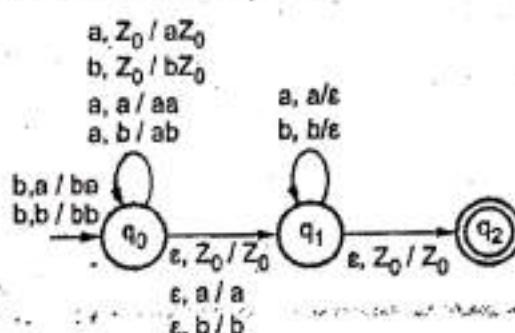
$$\delta(q_1, a, a) = (q_1, \epsilon)$$

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

$$\delta(q_1, \epsilon, Z_0) = (q_2, Z_0)$$

ACCEPT

The NFA can be designed for instantaneous description as

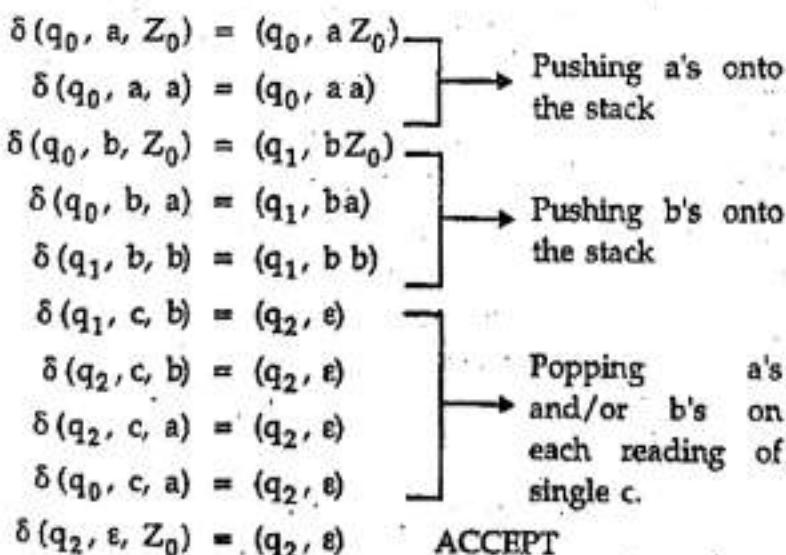


This shows that the given language pal cannot be represented using deterministic pushdown automata.

Example 4.2.14 : For the language $L = \{ a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i + j = k \}$ design a PDA (Push Down Automata) and trace it for string "bbbbbbcccccc".

GTU : Summer-15, Marks 7

Solution : The logic for developing this PDA is that read a (if any) push it onto the stack. Then if b is read then push it onto the stack. Continue pushing until C is read. On reading c, POP the stack. Thus for each reading of c, just POP single symbol from the stack. Finally when blank character is read the stack must be empty. The instantaneous description can be given as



Simulation of bbbbbcccccc

$$\begin{aligned}
 \delta(q_0, \underline{bbbb}cccccc; Z_0) &\vdash (q_1, \underline{bb}bbcccccc, \underline{b} Z_0) \\
 &\vdash (q_1, \underline{bb}cccccc, \underline{b} \underline{b} Z_0) \\
 &\vdash (q_1, \underline{b}cccccc, \underline{b} \underline{b} \underline{b} Z_0) \\
 &\vdash (q_1, ccccc, \underline{b} \underline{b} \underline{b} \underline{b} Z_0) \\
 &\vdash (q_1, \underline{cccc}, \underline{b} \underline{b} \underline{b} \underline{b} Z_0) \\
 &\vdash (q_1, \underline{ccc}, \underline{b} \underline{b} \underline{b} \underline{b} Z_0) \\
 &\vdash (q_2, cc, \underline{b} \underline{b} \underline{b} Z_0) \\
 &\vdash (q_2, c, \underline{b} Z_0) \\
 &\vdash (q_2, \epsilon, Z_0) = (q_2, \epsilon)
 \end{aligned}$$
ACCEPT

Example 4.2.15 : Give transition table for PDA accepting the language of all odd-length strings over {a, b} with middle symbol a. Also draw a PDA for the same.

GTU : Winter-19, Marks 7

Solution : For constructing this PDA we must follow two conditions -

- 1) The middle symbol is a
- 2) The string w_1 , a w_2 should be such that $|w_1| = |w_2|$

Hence there would be three phases

Phase 1 : Read the string from $\Sigma = \{a, b\}$ which is of length w_1 . Push each and every symbol read, onto the stack.

Phase 2 : Just read the middle symbol a. This is transition from phase 1 to phase 2.

Phase 3 : Pop the stack on every reading of single symbol.

This will ensure $|w_1| = |w_2|$

The PDA will be

$$\left. \begin{array}{l}
 \delta(q_0, a, Z_0) = \delta(q_0, aZ_0) \\
 \delta(q_0, b, Z_0) = \delta(q_0, bZ_0) \\
 \delta(q_0, a, a) = \delta(q_0, aa) \\
 \delta(q_0, a, b) = \delta(q_0, ab) \\
 \delta(q_0, b, a) = \delta(q_0, ba) \\
 \delta(q_0, b, b) = \delta(q_0, bb)
 \end{array} \right\} \text{Pushing of symbols (Phase 1)}$$

$$\begin{aligned}
 \delta(q_0, a, Z_0) &= \delta(q_2, Z_0) \\
 \delta(q_0, a, a) &= \delta(q_1, a) \\
 \delta(q_0, a, b) &= \delta(q_1, b) \\
 \delta(q_1, a, a) &= \delta(q_2, \epsilon) \\
 \delta(q_1, a, b) &= \delta(q_2, \epsilon) \\
 \delta(q_1, b, b) &= \delta(q_2, \epsilon) \\
 \delta(q_1, b, a) &= \delta(q_2, \epsilon) \\
 \delta(q_0, \epsilon, Z_0) &= \delta(q_3, \epsilon)
 \end{aligned}
 \quad \left. \begin{array}{l} \text{Reading middle } a \text{ (Phase 2)} \\ \text{Popping of symbols (Phase 2)} \\ \text{Accept state} \end{array} \right\}$$

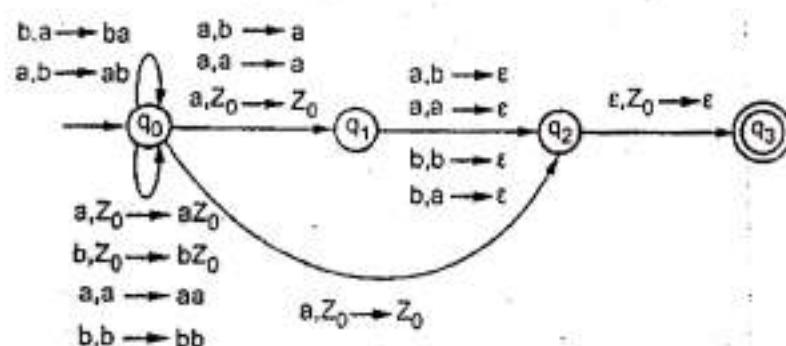


Fig. 4.2.4

Transition table

Current state	Current input	Next state	Stack
q_0	a	q_0	Z_0
q_0	b	q_0	Z_0
q_0	a	q_0	a
q_0	b	q_0	b
q_0	a	q_0	a
q_0	a	q_1	Z_0
q_0	a	q_1	a
q_0	a	q_1	b
q_0	a	q_2	Z_0

q_1	a	q_2	b
q_1	a	q_2	a
q_1	b	q_2	b
q_1	b	q_2	a
q_2	ϵ	q_3	Z_0
q_3	ϵ	ϕ	Z_0

4.3 Equivalence of CFG and PDA

The context free languages can be modeled using push down automata. Hence there exists equivalence between the two. That is CFG can be converted to equivalent PDA , similarly PDA can be converted to equivalent CFG. Let us understand this equivalence with the help of examples.

4.3.1 CFG to PDA

Step 1 : Convert the CFG to the following Form.

Nonterminal->terminal.
Set of Non terminals

Nonterminal->terminal

Step 2 : The δ function is to be developed for the grammar of the form.

$A \rightarrow aB \quad \text{as}$

$\delta(q_1, a, A) \rightarrow \delta(q_1, B)$

Step 3 : Finally add the rule

$\delta(q_1, \epsilon, Z_0) \rightarrow (q_1, \epsilon)$

where Z_0 is the stack symbol. This is the accept state.

Let us solve some examples based on this method.

→ **Example 4.3.1 :** Construct PDA for the following grammar

$$\begin{array}{ll} S \rightarrow AB & A \rightarrow CD \\ B \rightarrow b & C \rightarrow a \\ D \rightarrow a & \end{array}$$

Solution : The above grammar is first converted to simplistic form as -.

$$\begin{array}{ll} S \rightarrow CDB & S \rightarrow aDB \\ A \rightarrow CD & \Rightarrow A \rightarrow aD \\ B \rightarrow b & B \rightarrow b \\ C \rightarrow a & C \rightarrow a \\ D \rightarrow a & D \rightarrow a \end{array}$$

Now the equivalent PDA can be -

$$\delta(q_1, a, S) \rightarrow (q_1, DB)$$

$$\delta(q_1, a, A) \rightarrow (q_1, D)$$

$$\delta(q_1, b, B) \rightarrow (q_1, \epsilon)$$

$$\delta(q_1, a, C) \rightarrow (q_1, \epsilon)$$

$$\delta(q_1, a, D) \rightarrow (q_1, \epsilon)$$

Then add the rule

$$\delta(q_1, \epsilon, Z_0) \rightarrow (q_f, \epsilon) \text{ is the accept state.}$$

We will now simulate the string "ab" for the same.

$$\begin{aligned} \delta(q_1, ab, S) &\vdash \delta(q_1, a, b, DB) \\ &\vdash \delta(q_1, b, B) \\ &\vdash \delta(q_1, \epsilon, Z_0) \\ &\vdash \delta(q_f, \epsilon) \quad \text{Accepting configuration} \end{aligned}$$

Example 4.3.2 : Construct a PDA equivalent of the grammar given below :

$$S \rightarrow aAA$$

$$A \rightarrow aS \mid bS \mid a$$

Solution : The given grammar is already in required form having first symbol as terminal on RHS. Hence the PDA can be -

$$\delta(q_1, a, S) \rightarrow (q_1, AA)$$

$$\delta(q_1, a, A) \rightarrow (q_1, S)$$

$$\delta(q_1, b, A) \rightarrow (q_1, S)$$

$$\delta(q_1, a, A) \rightarrow (q_1, \epsilon)$$

$$\delta(q_1, \epsilon, Z_0) \rightarrow (q_1, \epsilon) \text{ ACCEPT}$$

The simulation of abaaaa is

$$\begin{aligned} \delta(q_1, abaaaa, S) &\vdash \delta(q_1, baaaa, AA) \\ &\vdash \delta(q_1, aaaaa, SA) \\ &\vdash \delta(q_1, aaaa, AAA) \\ &\vdash \delta(q_1, aaa, AAA) \\ &\vdash \delta(q_1, aa, AA) \\ &\vdash \delta(q_1, a, A) \\ &\vdash \delta(q_1, \epsilon, Z_0) \\ &\vdash \delta(q_1, \epsilon) \text{ ACCEPT} \end{aligned}$$

Example 4.3.3 : Construct an NPDA that accept the language generated by the grammar
 $S \rightarrow aSbb \mid ab.$

GTU : Summer-09, Marks 6

Solution : First we will convert the given CFG as -

$$\begin{aligned} S \rightarrow aSbb &\Rightarrow S \rightarrow aSBB \\ &\quad B \rightarrow b \\ S \rightarrow ab &\Rightarrow S \rightarrow aAB \\ &\quad A \rightarrow a \end{aligned}$$

Now consider the grammar

$$\begin{aligned} S &\rightarrow aSBB \\ S &\rightarrow aAB \\ S &\rightarrow a \\ B &\rightarrow b \end{aligned}$$

The PDA can be

$$\begin{aligned} \delta(q_0, a, S) &= \delta(q_0, SBB) \\ \delta(q_0, a, S) &= \delta(q_0, AB) \\ \delta(q_0, a, A) &= \delta(q_0, \epsilon) \\ \delta(q_0, b, B) &= \delta(q_0, \epsilon) \\ \delta(q_0, \epsilon, z_0) &= \delta(q_0, \epsilon) \text{ ACCEPT.} \end{aligned}$$

Example 4.3.4 : Given a CFG, $G = (S, A, B), \{0, 1\}, P, S$ with P as follows

$$S \rightarrow 0B \mid 1A \quad A \rightarrow 0S \mid 1AA \mid 0 \quad B \rightarrow 1S \mid 0BB \mid 1$$

Design a PDA M corresponding to CFG, G . Show that the string 0001101110 belongs to $CFL, L(G)$

GTU : Winter-16, Marks 7

Solution : The PDA for given CFG can be given by δ transition as,

$$\begin{aligned} \delta(q, 0, S) &\rightarrow (q, B) \\ \delta(q, 1, S) &\rightarrow (q, A) \\ \delta(q, 0, A) &\rightarrow (q, S) \\ \delta(q, 1, A) &\rightarrow (q, AA) \\ \delta(q, 0, A) &\rightarrow (q, \epsilon) \\ \delta(q, 1, B) &\rightarrow (q, S) \\ \delta(q, 0, B) &\rightarrow (q, BB) \\ \delta(q, 1, B) &\rightarrow (q, \epsilon) \end{aligned}$$

Simulation of 0001101110

$$\begin{aligned}\delta(q, 0001101110, S) &\vdash (q, 001101110, B) \\ &\vdash (q, 01101110, BB) \\ &\vdash (q, 1101110, BBB) \\ &\vdash (q, 101110, BBB) \\ &\vdash (q, 01110, B) \\ &\vdash (q, 1110, BB) \\ &\vdash (q, 110, B) \\ &\vdash (q, 10, S) \\ &\vdash (q, 0, A) \\ &\vdash (q, \epsilon) \\ &\text{ACCEPT}\end{aligned}$$

⇒ Example 4.3.5 : Convert the following grammar to a PDA :

$$\begin{aligned}I &\rightarrow a|b|Ia|Ib|IO|II \\ E &\rightarrow I|E^*E|E+E|(E)\end{aligned}$$

GTU : Summer-17, Marks 7

Solution : The equivalent PDA is

$$\begin{aligned}\delta(q, \epsilon, I) &= \rightarrow (q, Ia) \\ \delta(q, \epsilon, I) &\rightarrow (q, Ib) \\ \delta(q, \epsilon, I) &\rightarrow (q, IO) \\ \delta(q, \epsilon, I) &\rightarrow (q, II) \\ \delta(q, \epsilon, I) &= (q, a) \\ \delta(q, \epsilon, I) &= (q, b) \\ \delta(q, \epsilon, E) &= (q, I) \\ \delta(q, \epsilon, E) &= (q, E^*E) \\ \delta(q, \epsilon, E) &= (q, E+E) \\ \delta(q, \epsilon, E) &= (q, (E)) \\ \delta(q, a, a) &= (q, \epsilon) \\ \delta(q, b, b) &= (q, \epsilon)\end{aligned}$$

⇒ Example 4.3.6 : Construct PDA for

$$\begin{aligned}S &\rightarrow 0AB \\ A &\rightarrow 1A|1 \\ B &\rightarrow 0B|1A|0\end{aligned}$$

Trace the string 01011 using PDA.

GTU : Summer-18, Marks 4

Solution :

$$\delta(q, \epsilon, S) = \{(q, 0AB)\}$$

$$\delta(q, \epsilon, A) = \{(q, 1A), (q, 1)\}$$

$$\delta(q, \epsilon, B) = \{(q, 0B), (q, 1A), (q, 0)\}$$

$$\delta(q, 1, 1) = \{(q, \epsilon)\}$$

$$\delta(q, 0, 0) = \{(q, \epsilon)\}$$

ACCEPT

Simulation of 01011

$$\delta(q, 01011, S) \vdash \delta(q, 01011, 0AB)$$

$$\vdash \delta(q, 1011, \underline{AB})$$

$$\vdash \delta(q, 1011, 1B)$$

$$\vdash \delta(q, 011, \underline{B})$$

$$\vdash \delta(q, 011, 0B)$$

$$\vdash \delta(q, 11, B)$$

$$\vdash \delta(q, 11, 1A)$$

$$\vdash \delta(q, 1, A)$$

$$\vdash \delta(q, 1, 1)$$

$$\vdash \delta(q, \epsilon, \epsilon)$$

ACCEPT

→ **Example 4.3.7 :** Convert the following CFG into its equivalent PDA.

$$S \rightarrow AB$$

$$A \rightarrow BB$$

$$B \rightarrow AB$$

$$A \rightarrow a$$

$$B \rightarrow a|b$$

GTU : Winter-19, Marks 3

Solution : The equivalent PDA is

$$\delta(q, \epsilon, S) \rightarrow (q, AB)$$

$$\delta(q, \epsilon, A) \rightarrow (q, BB)$$

$$\delta(q, \epsilon, B) \rightarrow (q, AB)$$

$$\delta(q, \epsilon, A) \rightarrow (q, a)$$

$$\delta(q, \epsilon, B) \rightarrow (q, a)$$

$$\delta(q, \epsilon, B) \rightarrow (q, b)$$

$$\delta(q, a, a) = (q, \epsilon)$$

$$\delta(q, b, b) = (q, \epsilon)$$

ACCEPT

4.3.2 PDA to CFG

Algorithm for getting production rules of CFG

1. If q_0 is start state in PDA and q_n is final state of PDA then $[q_0 \ Z \ q_n]$ becomes start state of CFG. Here Z represents the stack symbol.

2. The production rule for the ID of the form $\delta(q_i, a, Z_0) = (q_{i+1}, Z_1 Z_2)$ can be obtained as

$$\delta(q_i \ Z_0 \ q_{i+k}) \rightarrow a (q_i \ Z_1 \ q_m) (q_m \ Z_2 \ q_{i+k})$$

where q_i, q_m represents the intermediate states, Z_0, Z_1, Z_2 are stack symbols and a is input symbol.

3. The production rule for the ID of the form

$$\delta(q_i, a, Z_0) = (q_{i+1}, \epsilon) \text{ can be converted as } (q_i \ Z_0 \ q_{i+1}) \rightarrow a$$

Let us understand this algorithm with the help of some examples.

Example 4.3.8 : Obtain CFG for the PDA given as below.

$A = ([q_0, q_1], \{0, 1\}, \{A, Z\}, \delta, Z, \{q_1\})$ where δ is as given below

$$\delta(q_0, 0, Z) = (q_0, AZ)$$

$$\delta(q_0, 1, A) = (q_0, AA)$$

$$\delta(q_0, 0, A) = (q_1, \epsilon)$$

Solution : Now let us apply algorithm for each δ transition and obtain production rules as follows.

$\delta(q_0, 0, Z) = (q_0, AZ)$ can be converted using rule 2 of algorithm. According to rule 2, $\delta(q_i, a, Z_0) = (q_{i+1}, Z_1 Z_2)$ gives $\delta(q_i \ Z_0 \ q_{i+k}) \rightarrow a (q_{i+1} \ Z_1 \ q_m) (q_m \ Z_2 \ q_{i+k})$.

Here $q_i = q_0$, $q_{i+1} = q_0$, $a = 0$, $Z_1 = A$ and $Z_2 = Z$. So we will get,

$$\delta(q_0, 0, Z) = (q_0, AZ) \text{ is}$$

$$1. \quad (q_0 \ Z \ q_0) \rightarrow 0(q_0 \ A \ q_0)(q_0 \ Z \ q_0) \mid 0(q_0 \ A \ q_1)(q_1 \ Z \ q_0)$$

$$2. \quad (q_0 \ Z \ q_1) \rightarrow 0(q_0 \ A \ q_0)(q_0 \ Z \ q_1) \mid 0(q_0 \ A \ q_1)(q_1 \ Z \ q_1)$$

Now consider

$$\delta(q_0, 1, A) = (q_0, AA) \text{ is}$$

$$3. \quad (q_0 \ A \ q_0) \rightarrow 1(q_0 \ A \ q_0)(q_0 \ A \ q_0) \mid 1(q_0 \ A \ q_1)(q_1 \ A \ q_0)$$

$$4. \quad (q_0 \ A \ q_1) \rightarrow 1(q_0 \ A \ q_0)(q_0 \ A \ q_1) \mid 1(q_0 \ A \ q_1)(q_1 \ A \ q_1)$$

Then,

$$\delta(q_0, 0, A) = (q_1, \epsilon) \text{ is obtained by applying rule 3.}$$

The rule 3 states that

if $\delta(q_i, a, Z_0) = (q_{i+1}, \epsilon)$ then

$$(q_i \ Z_0 \ q_{i+1}) \rightarrow a$$

Hence we get

$$5. \quad (q_0 \ A \ q_1) \rightarrow 0$$

To summarize this we can write equivalent CFG as -

$$(q_0 \ Z \ q_0) \rightarrow 0(q_0 \ A \ q_0)(q_0 \ Z \ q_0) \mid 0(q_0 \ A \ q_1)(q_1 \ Z \ q_0)$$

$$(q_0 \ Z \ q_1) \rightarrow 0(q_0 \ A \ q_0)(q_0 \ Z \ q_1) \mid 0(q_0 \ A \ q_1)(q_1 \ Z \ q_1)$$

$$(q_0 \ A \ q_0) \rightarrow 1(q_0 \ A \ q_0)(q_0 \ A \ q_0) \mid 1(q_0 \ A \ q_1)(q_1 \ A \ q_0)$$

$$(q_0 \ A \ q_1) \rightarrow 1(q_0 \ A \ q_0)(q_0 \ A \ q_1) \mid 1(q_0 \ A \ q_1)(q_1 \ A \ q_1)$$

$$(q_0 \ A \ q_1) \rightarrow 0$$

As given in the definition of PDA A the $\{q_1\}$ is a final state. Hence according to rule 1 $(q_0 \ Z \ q_1)$ is a start symbol. Hence $(q_0 \ Z \ q_1)$ is a start state.

Example 4.3.9 : Obtain CFG for the PDA as given below.

$$P = (\{q_0, q_1\}, \{0, 1\}, \{A, Z\}, \delta, q_0, Z, \phi)$$

The δ transition are ,

$$\delta(q_0, 0, Z) = (q_0, AZ)$$

$$\delta(q_0, 0, A) = (q_0, AA)$$

$$\delta(q_0, 1, A) = (q_1, \epsilon)$$

$$\delta(q_1, 1, A) = (q_1, \epsilon)$$

$$\delta(q_1, \epsilon, A) = (q_1, \epsilon)$$

$$\delta(q_1, \epsilon, Z) = (q_1, \epsilon)$$

Solution : Consider δ transition

$$\delta(q_0, 0, Z) = (q_0, AZ)$$

... Apply rule 1

$$1. \quad (q_0 \ Z \ q_0) \rightarrow 0(q_0 \ A \ q_0)(q_0 \ Z \ q_0) \mid 0(q_0 \ A \ q_1)(q_1 \ Z \ q_0)$$

$$2. \quad (q_0 \ Z \ q_1) \rightarrow 0(q_0 \ A \ q_0)(q_0 \ Z \ q_1) \mid 0(q_0 \ A \ q_1)(q_1 \ Z \ q_1)$$

$$\delta(q_0, 0, A) = (q_0, AA)$$

... Apply rule 1

$$3. \quad (q_0 \ A \ q_0) \rightarrow 0(q_0 \ A \ q_0)(q_0 \ A \ q_0) \mid 0(q_0 \ A \ q_1)(q_1 \ A \ q_0)$$

$$4. \quad (q_0 \ A \ q_1) \rightarrow 0(q_0 \ A \ q_0)(q_0 \ A \ q_1) \mid 0(q_0 \ A \ q_1)(q_1 \ A \ q_1)$$

- $\delta(q_0, 1, A) = (q_1, \epsilon)$... Apply rule 2
5. $(q_0 A q_1) \rightarrow 1$
- $\delta(q_1, 1, A) = (q_1, \epsilon)$... Apply rule 2
6. $(q_1 A q_1) \rightarrow 1$
- $\delta(q_1, \epsilon, A) = (q_1, \epsilon)$... Apply rule 2
7. $(q_1 A q_1) \rightarrow \epsilon$
- $\delta(q_1, \epsilon, Z) = (q_1, \epsilon)$... Apply rule 2
8. $(q_1 Z q_1) \rightarrow \epsilon$

The rules 1 to 8 indicate the production rules CFG.

⇒ Example 4.3.10 : For the PDA, $(\{q_0, q_1\}, \{0, 1\}, \{0, 1, z_0\}, \delta, q_0, z_0, \emptyset)$ where δ is

$$\begin{aligned}\delta(q_0, \epsilon, z_0) &= \{(q_1, \epsilon)\} \\ \delta(q_0, 0, z_0) &= \{(q_0, 0z_0)\} \\ \delta(q_0, 0, 0) &= \{(q_0, 00)\} \\ \delta(q_0, 1, 0) &= \{(q_0, 10)\} \\ \delta(q_0, 1, 1) &= \{(q_0, 11)\} \\ \delta(q_0, 0, 1) &= \{(q_1, \epsilon)\} \\ \delta(q_1, 0, 1) &= \{(q_1, \epsilon)\} \\ \delta(q_0, 0, 0) &= \{(q_1, \epsilon)\} \\ \delta(q_1, \epsilon, z_0) &= \{(q_1, \epsilon)\}\end{aligned}$$

Obtain CFG accepted by the above PDA.

GTU : Summer-18, Marks 7

Solution : 1) We will construct production rules for start symbol S.

$$S \rightarrow (q_0, z_0, q_0)$$

$$S \rightarrow (q_0, z_0, q_1)$$

2) Consider the rule $\delta(q_0, \epsilon, z_0) = (q_0, \epsilon)$

$$\text{i)} (q_0, z_0, q_0) \rightarrow \epsilon$$

3) Consider the rule $\delta(q_0, 0, z_0) = (q_0, 0z_0)$

$$\text{ii)} (q_0, z_0, q_0) \rightarrow 0(q_0, 0, q_0) (q_0, z_0, q_0)$$

$$\text{iii)} (q_0, z_0, q_0) \rightarrow 0(q_0, 0, q_1) (q_1, z_0, q_0)$$

$$\text{iv)} (q_0, z_0, q_1) \rightarrow 0(q_0, 0, q_0) (q_0, z_0, q_1)$$

$$\text{v)} (q_0, z_0, q_1) \rightarrow 0(q_0, 0, q_1) (q_0, z_0, q_1)$$

4) Consider the rule $\delta(q_0, 0, 0) = (q_0, 00)$

$$\text{vi)} (q_0, 0, q_0) \rightarrow 0(q_0, 0, q_0) (q_0, 0, q_0)$$

$$\text{vii)} (q_0, 0, q_0) \rightarrow 0(q_0, 0, q_1) (q_1, 0, q_0)$$

- viii) $(q_0, 0, q_1) \rightarrow 0(q_0, 0, q_0)(q_0, 0, q_1)$
ix) $(q_0, 0, q_1) \rightarrow 0(q_0, 0, q_1)(q_1, 0, q_1)$
5) Consider the rule $\delta(q_0, 1, 0) = (q_0, 10)$
x) $(q_0, 0, q_0) \rightarrow 1(q_0, 1, q_0)(q_0, 0, q_0)$
xi) $(q_0, 0, q_0) \rightarrow 1(q_0, 1, q_1)(q_1, 0, q_0)$
xii) $(q_0, 0, q_1) \rightarrow 1(q_0, 1, q_0)(q_0, 0, q_1)$
xiii) $(q_0, 0, q_1) \rightarrow 1(q_0, 1, q_1)(q_1, 0, q_1)$
6) Consider the rule $\delta(q_0, 1, 1) = (q_0, 11)$
xiv) $(q_0, 1, q_0) \rightarrow 1(q_0, 1, q_0)(q_0, 1, q_0)$
xv) $(q_0, 1, q_0) \rightarrow 1(q_0, 1, q_1)(q_1, 1, q_0)$
xvi) $(q_0, 1, q_1) \rightarrow 1(q_0, 1, q_0)(q_0, 1, q_1)$
xvii) $(q_0, 1, q_1) \rightarrow 1(q_0, 1, q_1)(q_1, 1, q_1)$
7) Consider the rule $\delta(q_0, 0, 1) = (q_1, \epsilon)$
xviii) $(q_0, 1, q_1) \rightarrow 0$
8) Consider the rule $\delta(q_1, 0, 1) = (q_1, \epsilon)$
xix) $(q_1, 1, q_1) \rightarrow 0$
9) Consider the rule $\delta(q_1, 0, 0) = (q_1, \epsilon)$
xx) $(q_1, 0, q_1) \rightarrow 0$
10) Consider the rule $\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$
xxi) $(q_1, z_0, q_1) \rightarrow \epsilon$

4.4 Non CFL

There is a class of languages that can not be represented using Context Free Grammar (CFG). Such a class of languages is called as non context free languages (non CFL). It is difficult to find out whether a context free grammar can be written for a language or not. To solve this problem, the pumping lemma for CFL can be used.

4.4.1 Pumping Lemma for CFL

The pumping lemma for regular sets states that every sufficiently long string in a regular set contains a short substring that can be pumped. In other words, if a long string is given and if we push or pump any number of substrings in any number of times then we always get a regular set. According to pumping lemma for CFL's there are always two short substrings which are close to each other and these both the substrings can be repeated as many times as required.

Lemma : Let L be any context free language, then there is a constant n , which depends only upon L , such that there exist a string $w \in L$ and $|w| \geq n$ where $w = pqrst$ such that

1. $|qs| \geq 1$
2. $|prs| \leq n$ and
3. For all $i \geq 0$ $p q^i r s^i t$ is in L .

Proof : This pumping lemma states that if there is a language L which is without unit productions and without a null production and there exist w where $w \in L$. The string w can be derived by a context free grammar G . The G be a grammar which is in Chomsky's Normal Form. The grammar G generates language L . For the string w , we can obtain a parse tree which derives the string w . Then if the length of the path to w is less than equal to i then the length of the word w is less than or equal to 2^{i-1} . We can prove this by induction step.

Basis : If $i = 1$

Let G contains the rule $S \rightarrow a$ where length of the derived string is 1 i.e. $i = 1$. Now according to the rule the word length should be $\leq 2^{i-1}$ i.e. $2^0 = 1$.

Observe that we have a word ' a ' which is of length 1. Also observe that the grammar G is in Chomsky's normal form. This language is regular since $|w| = |pqrs| = 1$.



Fig. 4.4.1

Induction step : Let w be a string which is derived by grammar G . Let k be a variable such that $n = 2^k$, $|w| \geq n$ then $|w| > 2^{k-1}$ while deriving w string we may get some non-terminals of CFG - G can be repeated for any number of times and will give the string w . If we pump the substrings to w such that the path length of this newly formed string w' ($w +$ Pumped string = w') is i and the word length of w' is $\leq 2^{i-1}$ then the grammar G deriving w' is called a regular grammar. The necessary condition is that grammar G is in Chomsky's Normal Form.

Let us consider a grammar

$$G = (\{A, B, C\}, \{a\}, \{A \rightarrow BC, B \rightarrow BA, C \rightarrow BA, A \rightarrow a, B \rightarrow b\}, A)$$

$$B \rightarrow BA$$

$$C \rightarrow BA$$

$$A \rightarrow a$$

$$B \rightarrow b, A)$$

Thus

$$A \xrightarrow{*} bba = w$$

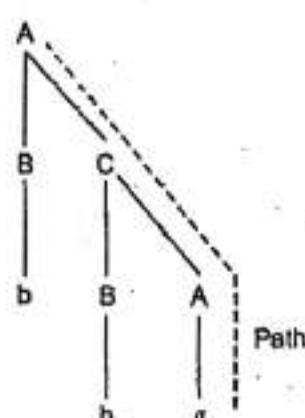


Fig. 4.4.2

i.e. Path length $i = 3$

$$|w| \leq 2^{i-1} \text{ i.e. } 3 \leq 2^2$$

If we pump a substring into w which satisfies the condition as $i \leq |w| \leq 2^{i-1} \leq n$ the grammar producing string w is a regular grammar.

Example 4.4.1 : Show that $L = \{a^n b^n c^n | n \geq 0\}$ is not a context free language.

Solution : Let us assume L is a context free language and $w \in L$. The w can be written as -

$$w = a^n b^n c^n \text{ which can also be written as -}$$

$$w = \underbrace{a \dots a}_{n} \underbrace{b \dots b}_{n} \underbrace{c \dots c}_{n}$$

Consider following cases -

Case 1 : Let,

$$w = \underbrace{a \dots a}_{n} \underbrace{b \dots b}_{n} \underbrace{c \dots c}_{n}$$

$\downarrow \quad \downarrow \quad \downarrow$
p qrs t

By pumping lemma, $w = pq^i rs^t \in L$ when $i \geq 0$. If we assume $i = 0$ then

$$w = \underbrace{a \dots a}_{n} \underbrace{b \dots b}_{n} \underbrace{c \dots c}_{n}$$

$\downarrow \quad \downarrow \quad \downarrow$
p $\underbrace{q^0 r s^0}_{r} \quad t$

$$w = a^n b^{n-m} c^n \notin a^n b^n c^n$$

Hence $w \notin L$.

Case 2 : Let,

$$w = \underbrace{a \dots a}_{n} \underbrace{b \dots b}_{n} \underbrace{c \dots c}_{n}$$

$\downarrow \quad \downarrow \quad \downarrow$
pqrs t

By pumping lemma $w = pq^i rs^i t \in L$. If $i = 2$ then

$$w = \underbrace{a \dots a}_{\downarrow}^{n+m+k} \underbrace{b \dots b}_t^n \underbrace{c \dots c}_t^n$$

$$pq^2 rs^2$$

Thus $w = a^{n+m+k} b^n c^n \notin L$.

From above cases we conclude that our assumption of L being a CFL is wrong. Hence it is proved that $L = \{a^n b^n c^n\}$ is not a context free language.

Example 4.4.2 : Determine whether the language given by $L = \{a^{n^2} | n \geq 1\}$ is context free or not.

Solution : We assume that given language is CFL.

Let Z be some string $\in L$. Then map

$$\begin{aligned} Z &= pqrst \\ &= a^{n^2} \\ &= \underbrace{a \dots a}_{n \times n} \end{aligned}$$

By pumping lemma, $Z = pq^i rs^i t \in L$.

Case 1 : Let,

$$\begin{aligned} Z &= pq^2 rs^2 t \\ Z &= \underbrace{a \dots a}_{n^2-m} \underbrace{a \dots a}_m \\ &\quad pq^2 rs^2 t \end{aligned}$$

If $i = 2$ then,

$$Z = \underbrace{a \dots a}_{n^2-m+t} \underbrace{a \dots a}_m$$

i.e. $|Z| = |n^2 - m + t + m|$

But $n^2 - m + t + m \notin n^2$

Hence our assumption of L being CFG is wrong.

Case 2 :

$$\text{Let } Z = \underbrace{\begin{matrix} n^2 - m \\ a, \dots, a \end{matrix}}_{pqrs} \quad \underbrace{\begin{matrix} m \\ a, \dots, a \end{matrix}}_t$$

If $i = 0$ then $Z = pr$ then

$$Z = \overbrace{a \dots a}^{n^2-m-1} \overbrace{a \dots a}^m$$

$$\text{i.e., } |Z| = |n^2 - t|$$

Hence our assumption of L being CFG is wrong. This proves that given language is not a CFG.

Example 4.4.3 : Define pumping Lemma for context free languages. Show that $L = \{a^i b^j c^k : i < j < k\}$ is not context free language.

Solution : Pumping Lemma - Refer section 4.4.1.

The languages { L= $a^i b^j c^k \mid i \leq j \leq k$ } is given.

If we consider some string $Z \in L$, such that

$$z = a^i b^j c^k$$

According to pumping lemma, if we pump some strings in between then also the $Z \in L$.

We consider following cases assuming L as CFL.

Case 1 : Strings of b's only

Let.

If $i = 2$, then $Z = \text{pqrsst}$ then

But then $j+m > k$ can be possible. Hence Z will not belong to L .

$$Z = \underbrace{a \dots a}_i \underbrace{b \dots b}_j \underbrace{c \dots c}_k$$

Case 2 : Strings of a's and b's

$$Z = \underbrace{a_1 \dots a_i}_{\text{i}}, \underbrace{b_1 \dots b_j}_{\text{j+m}}, \underbrace{c_1 \dots c_k}_{\text{k}}.$$

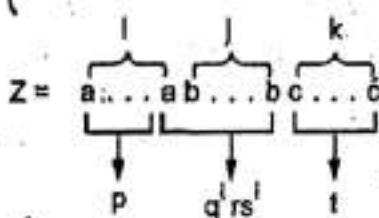
If $i = 2$ then $Z = pqqrst$. Then

$$Z = a^i(ab)^m b^{i-m} c^k \notin a^i b^i c^k \notin L$$

Thus our assumption of L being CFL is wrong. Hence given L is not CFL.

⇒ Example 4.4.4 : Prove that $L = \{a^m b^{m+1} c^{m+2} \mid m > 0\}$ is not a CFL.

Solution : We assume that given language is CFL. Let Z be some string and $Z \in L$. We can write Z as



$$\begin{aligned} Z &= pqqrst \\ &= a^m b^{m+1} c^{m+2} \end{aligned}$$

By pumping Lemma, $Z = pq^i rs^i t$

Case 1 : $i = 2$

$$\begin{aligned} Z &= pq^2 rs^2 t \\ &= \underbrace{a...ab...b}_{pq^2 rs^2} \underbrace{c...c}_{t} \\ &\quad \text{m+n m+1 m+2} \end{aligned}$$

$$Z = a^{m+n} b^{m+1} c^{m+2} \notin L$$

Case 2 : $i = 0$

$$Z = pq^0 rs^0 t$$

$$\begin{aligned} &= prt \\ Z &= \underbrace{a...a}_{pq} \underbrace{b...b}_{r} \underbrace{c...c}_{t} \\ &\quad \text{m-n m+1 m+2} \in L \end{aligned}$$

From above two cases, our assumption of L being CFL is wrong. Hence given L is not a CFL is proved.

⇒ Example 4.4.5 : Decide whether the given language is a CFL and prove your answer.
 $L = \{xyx/x, y \in \{a, b\}^*\text{ and }|x| \geq 1\}$

GTU : Summer-18, Marks 3

Solution : We assume that given L is CFL. There must exist some words, p, q, r, s, t , such that $Z = pqrst$ such that,

$|qs| > 0$, $|qrs| \leq n$ and $pq^i rs^i \in L$ for every $i \geq 0$. There are two cases.

Assume $x = a^n b^n$ and $y = \epsilon$

1. Assume qs consists of only a's from the first group of a's in Z

i.e.

$$Z = a \underbrace{a \dots a}_{n-1} \underbrace{b \dots b}_n$$

↑ ↓ ↓

p qrs t

If $i = 2$ then

$$\begin{aligned} Z &= aa^{n+m}b^n \\ &= a^{n+m+1}b^n \notin L \end{aligned}$$

2. Assume qs consists of only b's from the second group of Z

i.e.

$$Z = \underbrace{a^n}_p b \underbrace{b \dots b}_{n-1}$$

↑ ↓ ↓

qrs t

If $i = 2$ then

$$Z = a^n b^{n+m} \notin L$$

From above two cases, clearly our assumption of L being CFL is not valid. This proves that given L is not CFL.

► Example 4.4.6 : Show using pumping lemma that the given language is not a CFL.

$$L = \{a^n b^{2n} a^n | n \geq 0\}$$

GTU : Summer-18, Marks 3

Solution : We assume that given language is CFL. Let Z be some string $Z \in L$. We can write Z as,

$$Z = a^n b^{2n} a^n$$

$$= \underbrace{a \dots a}_n \underbrace{b \dots b}_{2n} \underbrace{a \dots a}_n$$

↑ ↓ ↑

p q'rs' t

$$Z = pqrst$$

By pumping lemma if $Z = pq^i rs^i t \in L$ then given language L is regular

Case 1 : Let, $i = 2$

$$\begin{aligned} Z &= pq^2rs^2t \\ &= a^n b^{2n+m} a^n \notin L \end{aligned}$$

Case 2 : Let, $i = 0$

$$\begin{aligned} Z &= pq^0rs^0t \\ &= a^n b^{2n-m} a^n \notin L \end{aligned}$$

From above two cases, our assumption of L being CFL is wrong.

Hence given L is not CFL.

4.5 Intersections and Complements of CFL

There are some special properties of the context free languages which can be represented by the notations union, concatenations and $*$. These properties are called **Closure properties** of the context free languages. These properties represent whether or not the CFL is closed under union, concatenation or closure operation. The CFLs are closed under some operation means after performing that operation on the CFL the resultant language is Context Free Language (CFL) only.

1. The context free languages are closed under union.
2. The context free languages are closed under concatenation.
3. The context free languages are closed under Kleen closure.
4. The context free languages are not closed under intersection.
5. The context free languages are not closed under complement.

We will discuss the above mentioned closure properties of CFL with the help of proofs and examples.

Theorem 1 : If L_1 and L_2 are context free languages then $L = L_1 \cup L_2$ is also context free. That is, the CFLs are closed under union.

Proof : We will consider two languages L_1 and L_2 which are context free languages. We can give these languages using context free grammars G_1 and G_2 such that $G_1 \in L_1$ and $G_2 \in L_2$. The G_1 can be given as $G_1 = \{V_1, \Sigma, P_1, S_1\}$ where P_1 can be given as

$$\begin{aligned} P_1 = \{ & \\ & S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S_1 B_1 \mid \epsilon \\ & A_1 \rightarrow a \\ & B_1 \rightarrow b \} \end{aligned}$$

Here $V_1 = \{S_1, A_1, B_1\}$ and S_1 is a start symbol.

Similarly, we can write $G_2 = \{V_2, \Sigma, P_2, S_2\}$

$N_2 = \{S_2, A_2, B_2\}$ and S_2 is a start symbol.

P_2 can be given as :

$P_2 = \{$

$S_2 \rightarrow a A_2 A_2 \mid b B_2 B_2$

$A_2 \rightarrow b$

$B_2 \rightarrow a$

}

Now $L = L_1 \cup L_2$ gives $G \in L$. This G can be written as

$G = \{V, \Sigma, P, S\}$

$V = \{S_1, A_1, B_1, S_2, A_2, B_2\}$

$P = \{P_1 \cup P_2\}$

S is a start symbol

$P = \{ S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S B_1 \mid \epsilon$

$A_1 \rightarrow a$

$B_1 \rightarrow b$

$S_2 \rightarrow a A_2 A_2 \mid b B_2 B_2$

$A_2 \rightarrow b$

$B_2 \rightarrow a$

}

Thus grammar G is a context free grammar which produces languages L which is context free language.

Theorem 2 : If L_1 and L_2 are two context free languages then $L_1 L_2$ is CFG. That means context free languages are closed under concatenation.

Proof : Let L_1 is a context free language which can be represented by a context free grammar G_1 , such that $G_1 \in L_1$ and

$G_1 = \{V_1, \Sigma, P_1, S_1\}$

$V_1 = \{S_1, A_1, B_1\}$

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

S_1 is a start symbol and P_1 is a set of production rules,

$$P_1 = \{ \quad S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S_1 B_1 \mid \epsilon \\ A_1 \rightarrow a \\ B_1 \rightarrow b \\ \}$$

Similarly, L_2 is a context free language which can be represented by a context free grammar G_2 , such that $G_2 \in L_2$ and

$$G_2 = [V_2, \Sigma, P_2, S_2] \\ V_2 = \{S_2, A_2, B_2\} \\ \Sigma = \{a, b\}$$

S_2 is a start symbol and P_2 is a set of production rules,

$$P_2 = \{ \quad S_2 \rightarrow aA_2A_2 \mid bB_2B_2 \\ A_2 \rightarrow a \\ B_2 \rightarrow b \\ \}$$

Now $L = L_1 L_2$ can be obtained by G such that $G = G_1 \cup G_2$. Therefore

$$G = [V, \Sigma, P, S] \\ V = \{S, S_1, A_1, B_1, S_2, A_2, B_2\}$$

where S is a start symbol. The production rules, P can be given as

$$P = \{ \quad S \rightarrow S_1 \mid S_2 \\ S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S_1 B_1 \mid \epsilon \\ A_1 \rightarrow a \\ B_1 \rightarrow b \\ S_2 \rightarrow aA_2A_2 \mid bB_2B_2 \\ A_2 \rightarrow a \\ B_2 \rightarrow b \\ \}$$

As grammar G is context free grammar the language L produced by G is also context free language. Hence context free languages are closed under concatenation.

Theorem 3 : If L_1 is context free language then L_1^* is also context free. That means CFL is closed under kleen closure.

Proof : Let, L_1 be a context free language represented by G_1 such that $G_1 \rightarrow \epsilon \in L_1$.
The CFG G_1 can be given as

$G_1 = [V_1, \Sigma, P_1, S_1]$ where S_1 is a start symbol

$$\begin{aligned} P_1 = \{ & \quad S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S_1 B_1 \mid \epsilon \\ & \quad A_1 \rightarrow a \\ & \quad B_1 \rightarrow b \end{aligned}$$

}

Now $L = L_1^*$ can be represented by a grammar G such that

$$G = \{ (V, \Sigma, P, S) \}$$

$$V = \{S, S_1, A_1, B_1\}$$

$$\text{and } P = S \rightarrow S_1 S \mid \epsilon$$

$$S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S_1 B_1$$

$$A_1 \rightarrow a$$

$$B_1 \rightarrow b$$

}

Thus grammar G is a context free grammar and language L produced by G is also context free language. Hence context free language are closed under kleen closure.

Theorem 4 : If L_1 and L_2 are two CFLs then $L = L_1 \cap L_2$ may be CFL or may not be CFL. That means L is not closed under intersection.

Proof : Let, $L_1 = \{0^n 1^n 2^i \mid n \geq 1, i \geq 1\}$

$$L_2 = \{0^n 1^n 2^n \mid n \geq 1, i \geq 1\}$$

The grammar for L_1 is

$$S \rightarrow AB$$

$$A \rightarrow 0A1 \mid 01$$

$$B \rightarrow 2B \mid 2$$

Similarly L_2 can be represented by grammar,

$$S \rightarrow AB$$

$$A \rightarrow 0A \mid 0$$

$$B \rightarrow 1B2 \mid 12$$

Now if we try to obtain

$L = L_1 \cap L_2$ then we get sometimes context free languages and sometimes non context free languages. Thus we can say that CFLs are not closed under intersection.

Theorem 5 : If L_1 is a CFL then L'_1 may or may not be CFL. That means CFL is not closed under complement.

Proof :

Let L_1 and L_2 are two CFLs. We will assume that complement of a context free language is a CFL itself. Hence L'_1 and L'_2 both are CFLs. We can also state that $(L'_1 \cup L'_2)$ is context free (since CFLs are closed under union). But $(L'_1 \cup L'_2) = L_1 \cap L_2$ i.e. $L = L_1 \cap L_2$ may or may not be CFL. The L_1 and L_2 are arbitrary CFLs, there may exist L'_1 and L'_2 which are not CFL. Hence complement of certain language may be context free or may not be. Therefore we can say that CFL is not closed under complement operation.

Review Question

1. Write theorem : If L_1 and L_2 are context free languages, then the language
 $L_1 \cup L_2$, $L_1 L_2$ and L_1^* are also CFLs.

GTU : Summer-13, Marks 7

4.6 Short Questions and Answers

- Q.1** Which graphical representation is used for modeling the context free grammar ?

Ans. : Push down automata.

- Q.2** The idea of automation with stack is executed in ____.

- a finite automata b push down automata
 c none of these

[Ans. : b]

- Q.3** Mention two ways of representation of PDA.

Ans. : The PDA can be represented using a. Transition diagram and b. Instantaneous description.

- Q.4** In push down automata what are the two types of halting states

Ans. : Accept state and reject states.

- Q.5** A pushdown automata can be defined as : $(Q, \Sigma, G, q_0, z_0, A, d)$

What does the symbol z_0 represents ?

Ans. : The stack symbol.

- Q.6** PDA is more powerful than _____.

- a turing Machine b finite automata
 c both a and b

[Ans. : b]

- Q.7** For every CFL, G , there exists a PDA M such that $L(G) = L(M)$ and vice versa.
 (True/False)

Ans. : True

Q.8 What is the difference between the finite automata and push down automata ?

Ans. : The push down automata has auxiliary memory in the form of stack but the finite automata does not have memory.

Q.9 The instantaneous description in PDA shows _____ .

- a present state b stack symbol
 c string to be processed d all of these

[Ans. : d]

Q.10 If the PDA does not stop on an accepting state and the stack is not empty, the string is :

- a Rejected b Goes into loop forever
 c Both (a) and (b) d None of the mentioned

[Ans. : a]

Q.11 Push down machine represents _____ .

- a Type 0 grammar b Type 1 grammar
 c Type - 2 grammar d Type - 3 grammar

[Ans. : c]



Turing Machine

5.1 Definition of Turing Machine (TM)

Alan Turing is a father of this model. This model has computing capability of general purpose computer.

The turing machine is a collection of following components.

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \Delta \text{ or } B, F)$$

- 1) Q is a finite set of states.
- 2) Γ is finite set of external symbols.
- 3) Σ is a finite set of input symbols.
- 4) Δ or b or $B \in \Gamma$ is a blank symbol majorly used as end marker for input.
- 5) δ is a transition or a mapping function.

→ Example 5.1.1 : What is Turing machine ? Write advantages of TM over FSM.

GTU : Winter-18, Marks 3

Solution : Turing machine : Refer section 5.1.

Advantages of TM over FSM

- 1) TM can remember previous inputs.
- 2) TM can output the symbols for corresponding symbols read.
- 3) TM can act as computer by computing arithmetic function, recognizing well formed parenthesis, reverse or concatenate strings and so on.

5.2 Model of Computation

The turing machine can be modelled with the help of following representation.

- 1) The input tape having infinite number of cells, each cell containing one input symbol and thus the input string can be placed on a tape. The empty tape is filled by blank characters. (As shown Fig. 5.2.1)

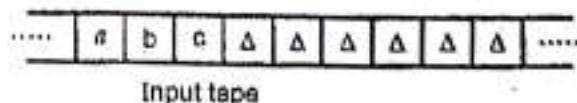


Fig. 5.2.1 Input tape

- 2) The finite control and the tape head which is responsible for reading the current input symbol. The tape head can move to left or right.
 - 3) A finite set of states through which machine has to undergo.
 - 4) Finite set of symbols called external symbols which are used in building the logic of turing machine.
- (As shown Fig. 5.2.2)

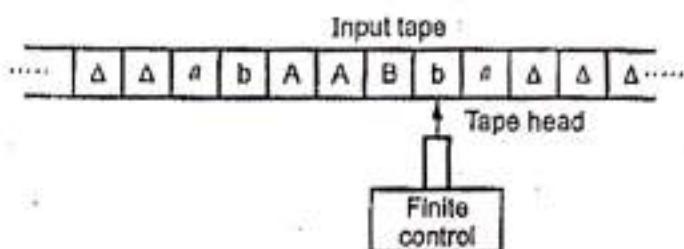


Fig. 5.2.2 Turing machine

5.2.1 Instantaneous Description

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \Delta \text{ or } B, F)$ be TM then the Instantaneous Description (ID) of a TM can be given by a triplet or a program which is as given below -

$$(q_0, a) \rightarrow (q_1, A, L)$$

That means currently we are reading the input symbol 'a' and we are in q_0 state then we can go to q_1 state by replacing or printing 'a' by A and then we must move in left direction.

5.3 Church's Turing Thesis

Hypothesis means proposing certain facts. The Church's hypothesis or Church's turing thesis can be stated as,

"The assumption that the intuitive notion of computable functions can be identified with partial recursive functions".

However, this hypothesis cannot be proved. The computability of recursive functions is based on following assumptions -

- 1) Each elementary function is computable.
- 2) Let f be the computable function and g be the another function which can be obtained by applying an elementary operation to f , then g becomes a computable function.
- 3) Any function becomes computable if it is obtained by rule 1) and 2).

Review Question

1. Explain Church Turing hypothesis.

GTU : Summer-16, Marks 3; Summer-18, Winter-18, Marks 4

5.4 Computing Functions with TM

The turing machine accepts all the languages even though there are recursively enumerable. Recursive means repeating the same set of rules for any number of times. And enumerable means a list of elements. The TM also accepts the computable functions, such as addition, multiplication, subtraction, division, power function, square function, logarithmic function and many more. We will solve some examples based on languages and function for construction of turing machine.

Example 5.4.1 : Construct a turing machine which accepts the language of aba over $\Sigma = \{a, b\}$.

Solution : This TM is only for $L = \{aba\}$

We will assume that on the input tape the string ' aba ' is placed like this

The tape head will read out the sequence upto the Δ character if ' aba ' is readout the TM will halt after reading Δ .

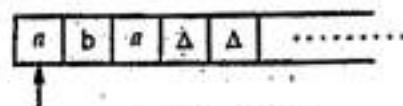


Fig. 5.4.1

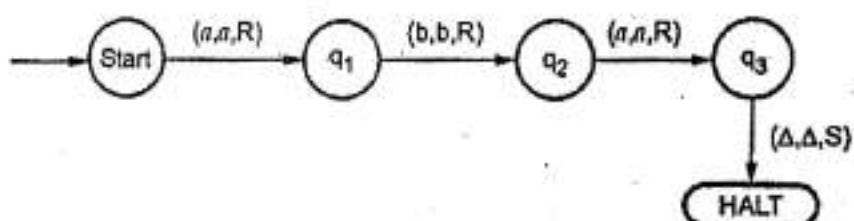


Fig. 5.4.2 TM for ' aba '

The triplet along the edge written is (input read, output to be printed, direction)

Consider the transition between start state and q_1 which is (a, a, R) that is the current symbol read from the tape is a then output it as a and then move the tape head to the right. The tape will look like this, as shown Fig. 5.4.3.

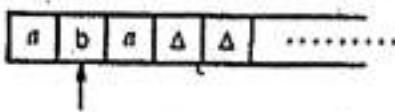


Fig. 5.4.3

Again the transition between q_1 and q_2 is (b, b, R) . That means read b , print b and move right. Note that as tape head is moving ahead the states are getting changed. (As shown Fig. 5.4.4)

The TM will accept the language when it reaches to halt state. Halt state is always an accept state for any TM. Hence the transition between q_3 and halt is (Δ, Δ, S) . This means read Δ , print Δ and stay there or there is no move left or right. Even though we write (Δ, Δ, L) or (Δ, Δ, R) it is equally correct. Because after all the complete input is already recognized and now we simply want to enter into an accept state or final state. Note that for invalid inputs such as abb or ab or bab ... there is either no path reaching to final state and for such inputs the

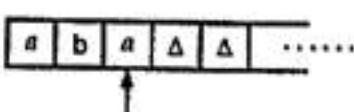


Fig. 5.4.4

TM gets stucked in between. This indicates that these all invalid inputs cannot be recognized by our TM.

The same TM can be represented by another method of transition table.

	a	b	Δ
Start	(q_1, a, R)	-	-
q_1	-	(q_2, b, R)	-
q_2	(q_3, a, R)	-	-
q_3	-	-	(HALT, Δ , S)
HALT	-	-	-

In the given transition table, we write the triplet in each row as :

(Next state, output to be printed, direction)

Thus TM can be represented by any of these methods.

→ Example 5.4.2 : Construct TM for language consisting of strings having any number of 0's and only even number of 1's over the input set $\Sigma = \{0, 1\}$.

Solution : The number of zero's can be any ! but the even number of 1's are to be allowed. FSM for this problem is shown in Fig. 5.4.5.

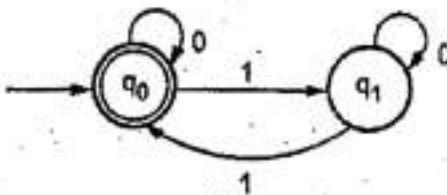


Fig. 5.4.5 FSM for even number of 1's

The same idea can be used to draw TM. We always assume that at the end of input Δ is placed.

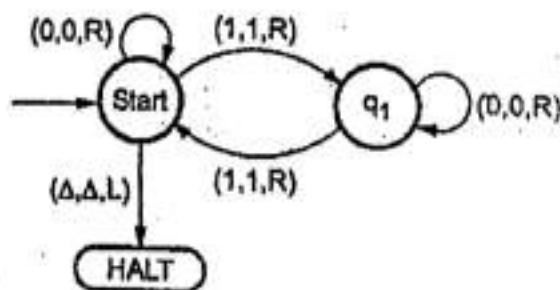


Fig. 5.4.6

Let us simulate the above TM for the input 1 1 0 1 0 1 which has even number of 1's. We assume that this input is placed on a input tape.

input head state	1	1	0	1	0	1	Δ
q ₀	1	1	0	1	0	1	Δ
	1	1	0	1	0	1	Δ
	Start						
	1	1	0	1	0	1	Δ
		Start					
	1	1	0	1	0	1	Δ
			↑				
	q ₁						
	1	1	0	1	0	1	Δ
				↑			
	q ₁						
	1	1	0	1	0	1	Δ
					↑		
	Start						
	1	1	0	1	0	1	Δ
						↑	
	HALT						

Thus this input is accepted by TM.

→ Example 5.4.3 : Construct a TM for the language of even number of 1's and even number of 0's over $\Sigma = \{0, 1\}$.

Solution : For this type of problem even can drawn FSM.

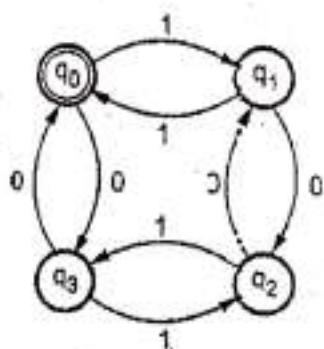


Fig. 5.4.7

And now it will be very much easy for us to convert this FSM to TM.

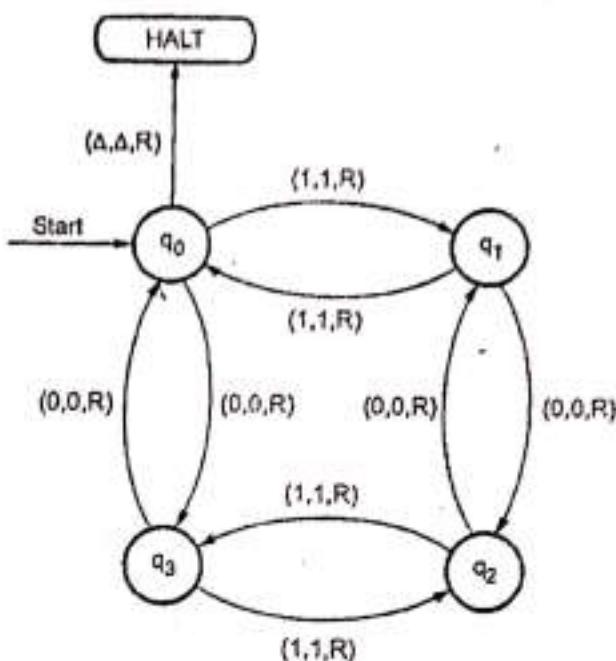


Fig. 5.4.8

Example 5.4.4 : Construct TM for the language $L = \{a^n b^n\}$ where $n \geq 1$.

Solution : We have already solved this problem by PDA. In PDA we have stack to remember the previous symbol. The main advantage of TM is we have a tape head which can be moved forward or backward, and the input symbol can be scanned. The simple logic which we will apply is read out each a mark it by A and then move ahead along the input tape and find out the b convert it to B. Repeat this process for all a 's and b 's. If there is some kind of inequality in number of a 's and b 's the TM will not end up in HALT state. Let us formulize the logic for $a^n b^n$ and then construct it,

$a \ a \ b \ b \ \Delta$	Convert it A and move right in search of b.
\uparrow	
$A \ a \ b \ b \ \Delta$	Skip it, move ahead
\uparrow	
$A \ a \ b \ b \ \Delta$	Convert it to B and move left till A
\uparrow	
$A \ a \ B \ b \ \Delta$	Move left
\uparrow	
$A \ a \ B \ b \ \Delta$	Move right
\uparrow	

$\alpha \# B b \Delta$	Convert α to A and move right in search of b
↑	
$A A B b \Delta$	Move right
↑	
$A A B b \Delta$	Convert b to B and move left
↑	
$A A B B \Delta$	Move left
↑	
$A A B B \Delta$	Now immediately before B is A that means all the α 's are marked by A. So we will move right to ensure that no b is present
↑	
$A A B B \Delta$	Move right
↑	
$A A B B \Delta$	Move right
↑	
$A A B B \Delta$	HALT
↑	

Thus equality between α 's and b's can be checked by moving tape head to and fro. Let us try to design a TM using above logic.

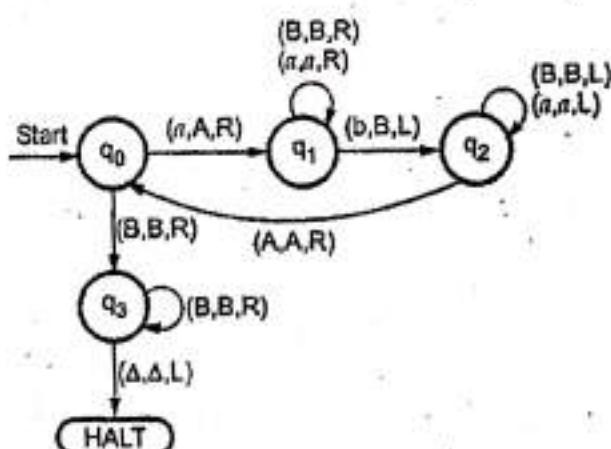


Fig. 5.4.9

Note that for the invalid inputs the machine does not go to HALT state.

For example

$\alpha \# b b b \Delta$	Convert to A, move right and next state = q_1
↑	
$A \# b b b \Delta$	Move right, keeping α as it is, state = q_1
↑	
$A \# b b b \Delta$	Convert to B, move to left
↑	
$A \# B b b \Delta$	Move to left, state = q_2
↑	

A a B b b Δ	Move to right and state = q ₀
↑	
A A B b b Δ	Convert to A, move to right now in state q ₁
↑	
A A B b b Δ	Move right keeping B as it is.
↑	
A A B b b Δ	Convert b to B and move left state is q ₂
↑	
A A B B b Δ	Go on moving left
↑	
A A B B b Δ	Now state q ₀ , move right
↑	
A A B B b Δ	Move right skipping all B's
↑	
A A B B b Δ	From state q ₃ there is no path for input b and TM stops there

Let us design the transition table for the same.

	a	b	A	B	Δ
q ₀	(q ₁ , A, R)	-	-	(q ₃ , B, R)	-
q ₁	(q ₁ , a, R)	(q ₂ , B, L)	-	(q ₁ , B, R)	-
q ₂	(q ₂ , a, L)	-	(q ₀ , A, R)	(q ₂ , B, L)	-
q ₃	-	-	-	(q ₃ , B, R)	(HALT, Δ, L)
HALT	-	-	-	-	-

Table 5.4.1 Transition table for example 5.4.4

Thus TM does not lead to HALT state for such a invalid input.

→ Example 5.4.5 : Construct a TM for $L = \{a^n b^n c^n \mid n \geq 1\}$.

GTU : Summer-16, Winter-16, Marks 7

Solution : This problem cannot be solved even by PDA. To solve this problem by PDA we require two stacks. The Turing machine can solve this problem. The same logic which we have used in previous example could also be used to solve this problem.

The simulation for $a a b b c c$ can be shown as below.

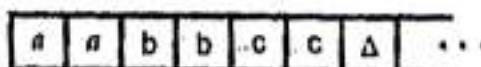


Fig. 5.4.10

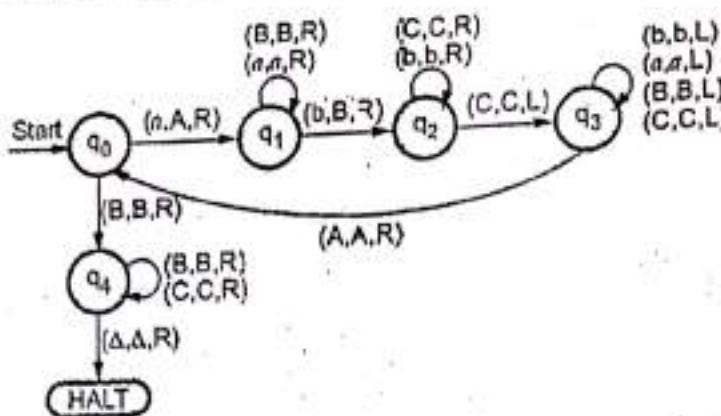


Fig. 5.4.11

- $\pi \# b \ b \ c \ c \Delta$ Convert π to A, move right transition from q_0 to q_1
 \uparrow
 $A \# b \ b \ c \ c \Delta$ Move right upto c
 \uparrow
 $A \# b \ b \ c \ c \Delta$ Convert b to B, move right transition from q_1 to q_2
 \uparrow
 $A \# B \ b \ c \ c \Delta$ Move right upto c
 \uparrow
 $A \# B \ b \ c \ c \Delta$ Convert c to C, move right transition from q_2 to q_3
 \uparrow
 $A \# B \ b \ C \ c \Delta$ Move left upto C
 \uparrow
 $A \# B \ b \ C \ c \Delta$ Move left till A, skipping π , b, B, C, see state q_3 then move one step right
 \uparrow
 $A \# B \ b \ C \ c \Delta$ Convert π to A, move right
 \uparrow
 $A \# B \ b \ C \ c \Delta$ Move right
 \uparrow
 $A A B \ b \ C \ c \Delta$ Convert b to B and move right, transition from q_1 to q_2
 \uparrow
 $A A B \ B \ C \ c \Delta$ Move right
 \uparrow
 $A A B \ B \ C \ c \Delta$ Convert c to C, move left till A, refer q_3 state
 \uparrow
 $A A B \ B \ C \ C \Delta$ Move right, keep B as it is refer transition from q_0 to q_4
 \uparrow

A A B B C C Δ
↑ Go on moving right by ignoring B, C refer q₄ state

A A B B C C Δ
↑ Since Δ is read TM will reach to HALT state

A A B B C C Δ
↑

Thus TM can very effectively recognize $a^n b^n c^n$ and more powerful than PDA.

⇒ Example 5.4.6 : Construct a TM machine for checking the palindrome of the string of even length.

Solution : The same type problem we have solved for push down automata. Now we are going to construct TM for it. The logic is that we will read first symbol from left and then we compare it with the first symbol from right to check whether it is the same.

Again we compare second symbol from left with the second symbol from right. We repeat this process for all the symbols. If we found any symbol not matching, we cannot lead the machine to HALT state. The simulation of machine for a valid input such as

* b a b b a b a Δ
↑ We will mark it by * and move to right end in search of a

* b a b b a b a Δ
↑ Move right

* b a b b a b a Δ
↑ Moved right upto Δ

* b a b b a b a Δ
↑ Moved left, checked if it is a

* b a b b a b a Δ
↑ If it is a, replaced it by Δ

* b a b b a b a Δ
↑ Move to left, upto *

* b a b b a b a Δ
↑ Move right, read it

* b a b b a b a Δ
↑ Convert it to * and move right

* a b b a b a Δ
↑ Move right upto Δ in search of b

* a b b a b a Δ
↑ Move left, if left symbol is b convert it to Δ

* a b b a b a Δ
↑ Move left till *

* * a b b a Δ Goto right
 ↑
 * * a b b a Δ Replace a by * and move right, upto Δ
 ↑
 * * * b b a Δ We will move left and check if it is a, then replace it by Δ
 ↑
 * * * b b a Δ It is a so replace by Δ
 ↑
 * * * b b a Δ Move to left till *
 ↑
 * * * b b a Δ Move right
 ↑
 * * * b b a Δ Replace it by *
 ↑
 * * * b b a Δ Move right upto Δ
 ↑
 * * * b a Δ Move left to see if it is b, if so then replace it by Δ
 ↑
 * * * a Δ Move left till *
 ↑
 * * * a Δ Move right and check whether it is a, if so
 ↑
 * * * a Δ Goto HALT state

Let us draw TM for it,

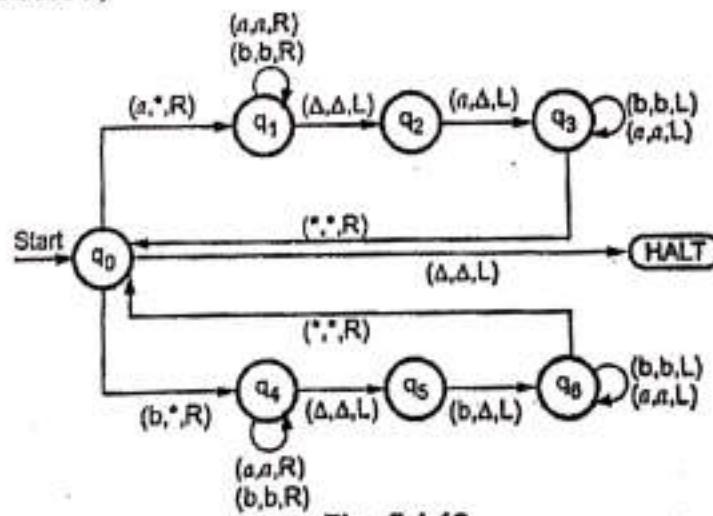


Fig. 5.4.12

If you observe the turing machine drawn for palindrome, you will find it very logically constructed. Even there is no need to insert any special symbol in between to separate out two halves.

Example 5.4.7 : Construct a TM for checking the palindrome of a string of odd palindrome for $\Sigma = \{0, 1\}$.

Solution : The TM will be very much similar to previous one but with a slight difference.

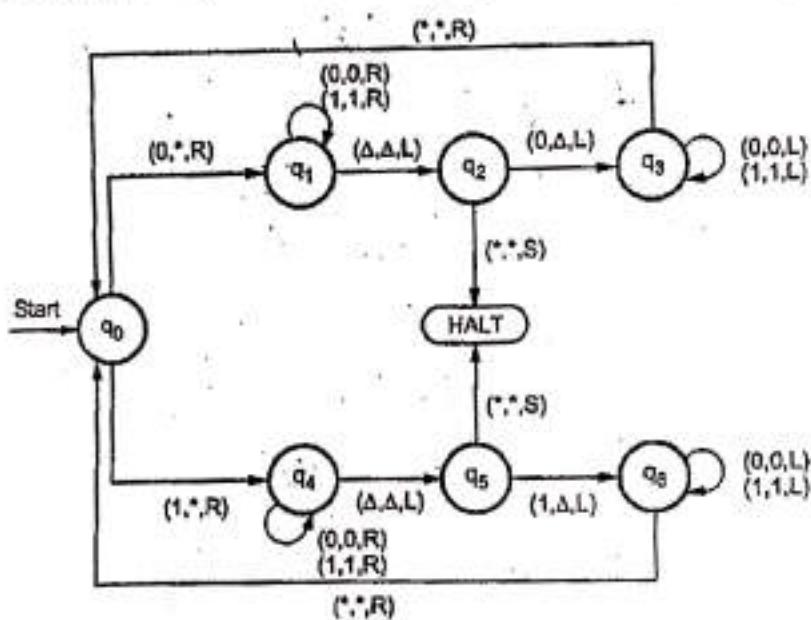


Fig. 5.4.13

For example : If the string is

- | | |
|---------------------------|-----------------------------------------------|
| 0 0 1 0 0 Δ | Move right by making 0 to * |
| ↑ | |
| * 0 1 0 0 Δ | Move right upto Δ |
| ↑ | |
| * 0 1 0 0 Δ | Move left, replace 0 by Δ |
| ↑ | |
| * 0 1 0 Δ Δ | Move left |
| ↑ | |
| * 0 1 0 Δ | Move left upto * |
| ↑ | |
| * 0 1 0 Δ | Move right convert 0 to * and then move right |
| ↑ | |
| ** 1 0 Δ | Move right upto Δ |
| ↑ | |

•• 1 0 Δ	Move left and replace 0 by Δ
•• 1 Δ Δ	Move left till *
•• 1 Δ Δ	Move right and convert 1 to *
•• * Δ Δ	Move right
•• * Δ Δ	Move left
•• * Δ Δ	Since it is *, goto HALT state

→ Example 5.4.8 : Define turing machine. Draw a Turing Machine (TM) to accept palindromes over $\{a, b\}$. (Even as well as odd palindromes)

GTU : Winter-12,13,16,17,18, Summer-15,16,18, Marks 7

Solution : The TM for palindrome string of $\Sigma = \{a, b\}$ will be as follows. The TM handles both even and odd palindromes.

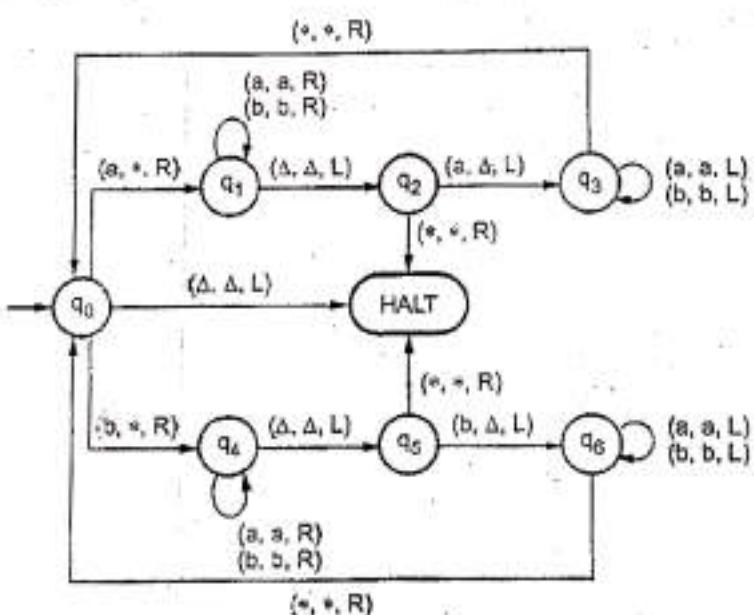


Fig. 5.4.14 Turing machine to accept palindrome

→ Example 5.4.9 : Construct TM for concatenation of the two strings of unary numbers. This TM is for a concatenate function.

Solution : The TM will be constructed for concatenate two strings of unary numbers. The strings are placed as follows

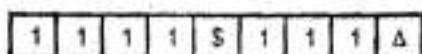


Fig. 5.4.15

The two strings are separated by \$ we will replace that \$ by next 1. That is we go on forwarding \$ ahead.

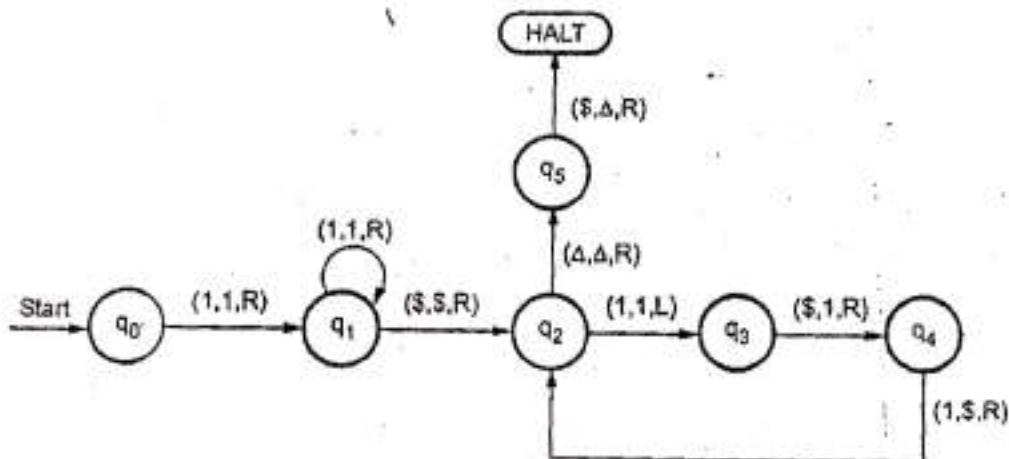


Fig. 5.4.16

Let us assume that on input tape the input is -

1 1 1 \$ 1 1 1 Δ

Move right till \$

1 1 1 \$ 1 1 1 Δ

Move right

1 1 1 \$ 1 1 1 Δ

If current symbol is 1 move left and convert \$ to 1

1 1 1 1 1 1 1 Δ

Move right and convert 1 to \$

1 1 1 1 \$ 1 1 Δ

Move right

1 1 1 1 \$ 1 1 Δ

If current symbol is 1 move left and convert \$ to 1 and move right

1 1 1 1 1 1 1 Δ

Convert 1 to \$ and move right

1 1 1 1 1 \$ 1 Δ

Moved right

1 1 1 1 1 \$ 1 Δ If current symbol is 1, move left convert \$ to 1
 ↑
 1 1 1 1 1 1 1 Δ Move right convert 1 to \$
 ↑
 1 1 1 1 1 1 \$ Δ Move right
 ↑
 1 1 1 1 1 1 \$ Δ Since it is end of input we will move left convert \$ to Δ and HALT
 ↑
 1 1 1 1 1 1 Δ

→ Example 5.4.10 : Construct a TM for a successor function for a given unary number i.e. $f(n) = n+1$.

Solution : Since the TM has to be constructed for unary number system we assume input set $\Sigma = \{1\}$. The logic is very much simple. We just go on moving towards rightmost end of input string. The end marker of the string Δ is replaced by 1.

For example, the input tape consists of 4 the successor function will give output as 5.

1 1 1 1 Δ We move to extreme right by keeping all 1's as it is
 ↑
 1 1 1 1 Δ Move right
 ↑
 1 1 1 1 Δ Move right
 ↑
 1 1 1 1 Δ Move right
 ↑
 1 1 1 1 Δ Convert Δ to 1
 ↑
 1 1 1 1 1 Δ HALT

The construction of TM will be

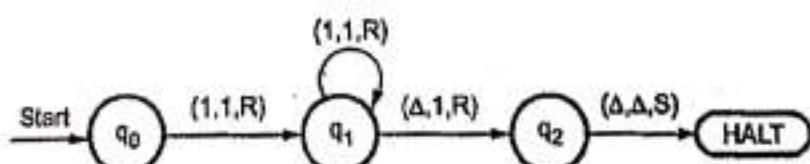


Fig. 5.4.17

The another method of representation of TM will be by transition table. It is always convenient to build transition table from the graph.

States \ Input	1	Δ
States		
q_0	$(q_1, 1, R)$	-
q_1	$(q_1, 1, R)$	$(q_2, 1, R)$
q_2	-	$(HALT, \Delta, S)$
HALT	-	-

In the transition table the triple is of (next state, output, direction).

→ Example 5.4.11 : Construct TM for the addition function for the unary number system.

Solution : The unary number is made up of only one character i.e. The number 5 can be written in unary number system as 11111. In this TM we are going to perform addition of two unary numbers.

For example : $2 + 3$

$$\begin{aligned} \text{i.e. } & 11 + 111 \\ & = 11111 \end{aligned}$$

If you observe this process of addition, you will find the resemblance with string concatenation function.

What we are trying to do is, we simply replace + by 1 and move ahead right for searching end of the string we will convert last 1 to Δ .

The input is $3 + 2$.

1 1 1 + 1 1 Δ	Move right upto + sign
↑	
1 1 1 + 1 1 Δ	Convert + to 1 and move right
↑	
1 1 1 1 1 1 Δ	Move right
↑	
1 1 1 1 1 1 Δ	Move right
↑	
1 1 1 1 1 1 Δ	Now Δ has encountered so just move left
↑	
1 1 1 1 1 1 Δ	Convert 1 to Δ
↑	
1 1 1 1 1 Δ Δ	Thus the tape now consists of the addition of two unary numbers

The TM will look like this

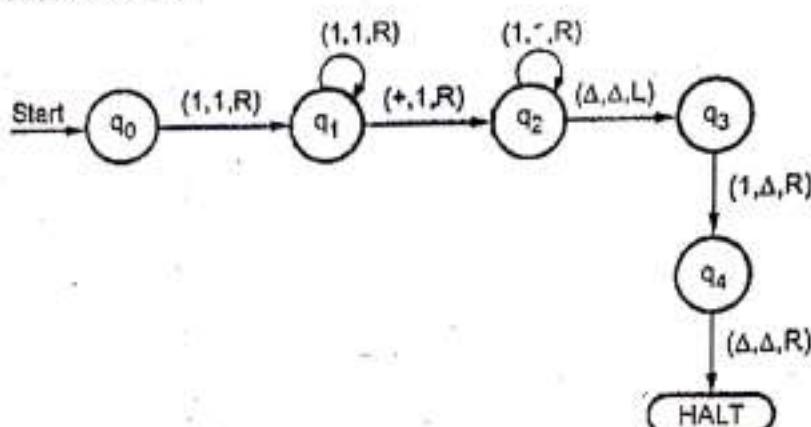


Fig. 5.4.18

Here we are implementing the function of $f(a + b) = c$. We assume a and b both are nonzero elements.

► Example 5.4.12 : Construct TM for performing subtraction of two unary numbers $f(a-b)=c$ where a is always greater than b .

Solution : Here we have certain assumption as first number is greater than second one. Let us assume that $a=3$, $b=2$, so the input tape will be as shown in Fig. 5.4.19.

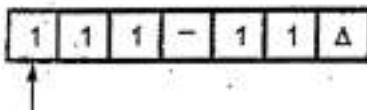
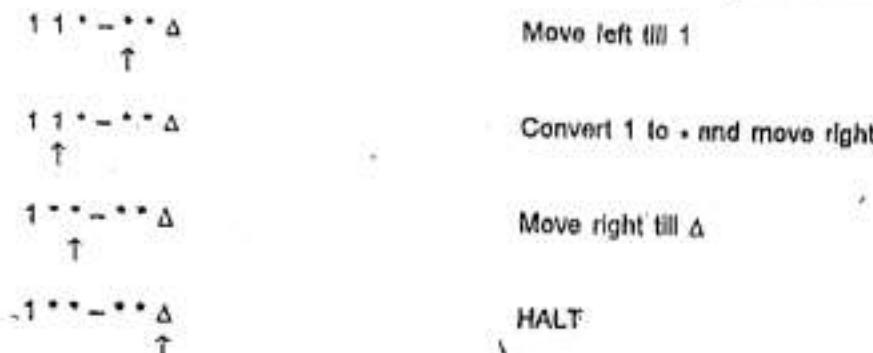


Fig. 5.4.19

We will move right to $-$ symbol as perform reduction of number of 1's from first number. Let us look at the simulation for understanding the logic.

1 1 1 - 1 1 Δ	Move right upto $-$
↑	
1 1 1 - * 1 Δ	Move right and convert 1 to *
↑	
1 1 1 - * 1 Δ	Now move left
↑	
1 1 1 - * 1 Δ	Move left
↑	
1 1 1 - * 1 Δ	Convert 1 to *
↑	
1 1 * - * 1 Δ	Move right
↑	
1 1 * - * 1 Δ	Move right till 1
↑	
1 1 * - * 1 Δ	Convert 1 to * and move left
↑	



Thus we get 1 on the input tape as answer for $f(3 - 2)$.

Let us draw TM for it.

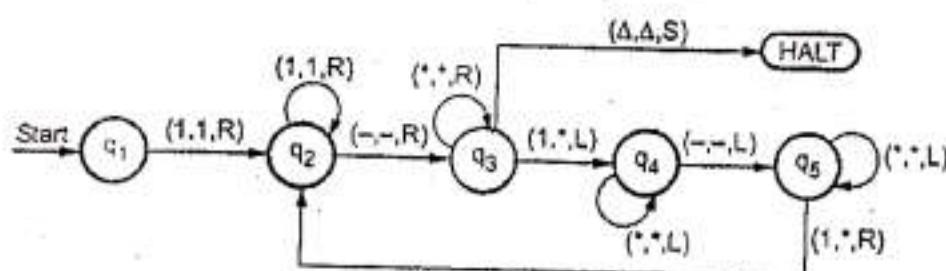


Fig. 5.4.20

Note that we have enforced a condition that first number is greater than the second. Even without such condition also the TM can be drawn by introducing some extra logic in existing TM. This part is left as exercise for the reader.

5.5 Combining TM

The power of Turing machine can be increased by designing the combined Turing Machine. The computations can be performed by executing them on TM1, TM2 and so on and then collecting the overall result. One such type of composite Turing machine is Multitape Turing Machine.

The multitape turing machine is a type of turing machine in which there are more than one input tapes. Each tape is divided into cells and each cell can hold any symbol of finite tape alphabet. The multitape turing machine is as shown in the Fig. 5.5.1.

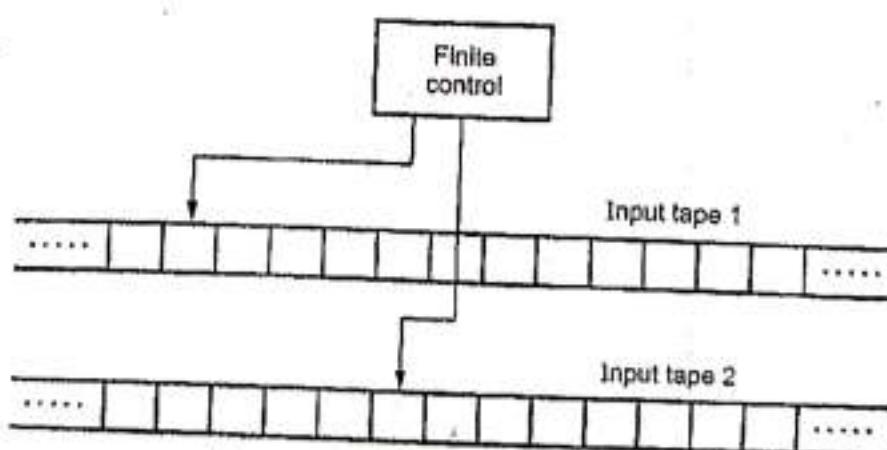


Fig. 5.5.1 A multitape turing machine

This TM is more powerful than the basic turing machine. Because finite control reads more than one input tape and more symbols can be scanned at a time.

→ **Example 5.5.1 :** Design a Turing machine M to implement the function multiplication using the subroutine 'copy'.

Solution : Initially the input is placed on the tape as follows.

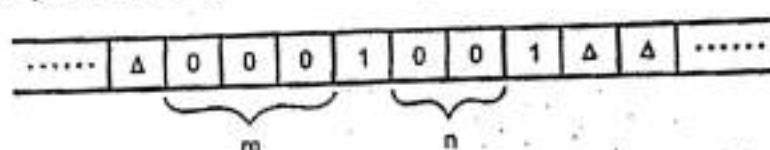


Fig. 5.5.2

The TM1 for performing multiplication can be -

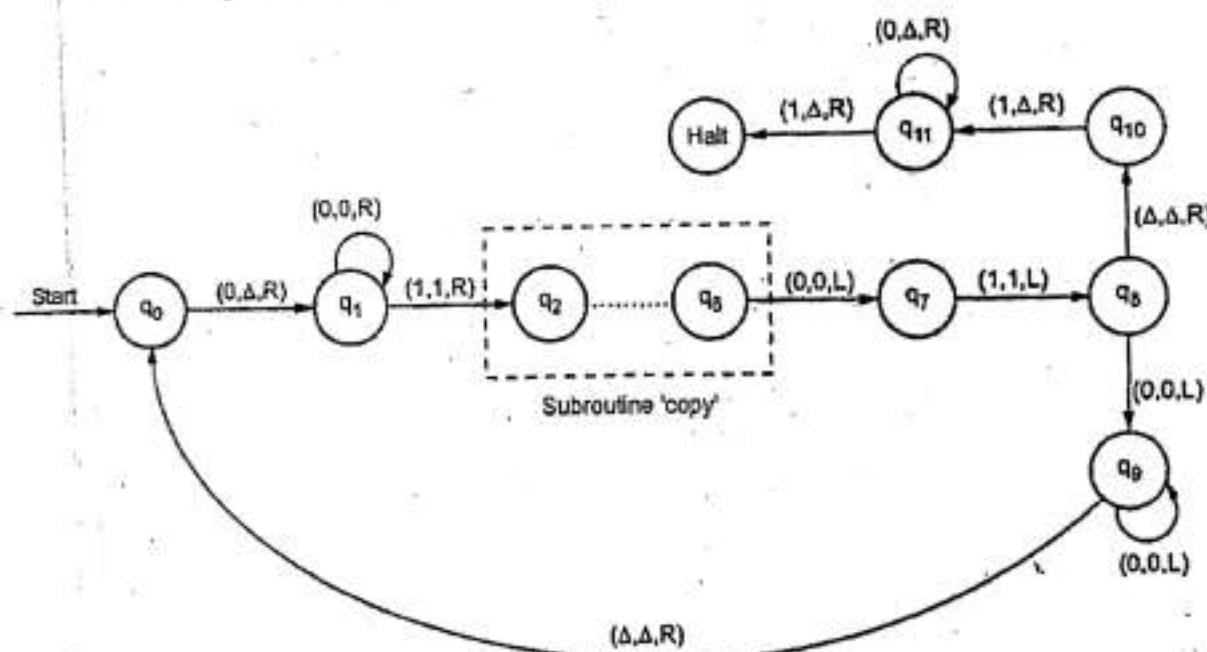


Fig. 5.5.3

The subroutine copy can be executed on another Turing Machine say TM2. It can be as shown in Fig. 5.5.4. (See Fig. 5.5.4 on next page)

To simulate above TM, let us take an example for multiplication of 3 and 2. The inputs are represented by unary 0 and separated by 1.

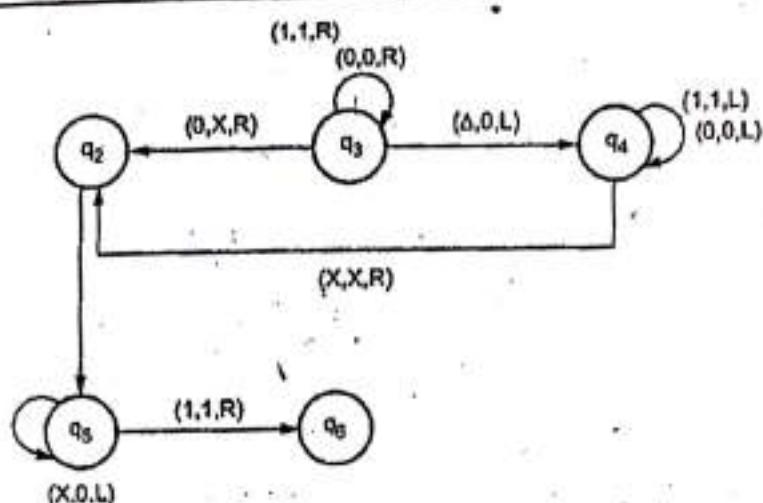


Fig. 5.5.4

$\Delta 0 0 0 1 0 0 1 \Delta \dots$	Convert to Δ and move right.
$\Delta \Delta 0 0 1 0 0 1 \Delta \Delta \Delta \Delta \Delta \dots$	Move right.
$\Delta \Delta 0 0 1 0 0 1 \Delta \Delta \Delta \dots$	Move right.
$\Delta \Delta 0 0 1 0 0 1 \Delta \Delta \Delta \dots$	Move right and call 'copy'.
$\Delta \Delta 0 0 1 0 0 1 \Delta \Delta \Delta \dots$	Convert 0 to X and move right.
$\Delta \Delta 0 0 1 X 0 1 \Delta \Delta \Delta \dots$	Move right.
$\Delta \Delta 0 0 1 X 0 1 \Delta \Delta \Delta \dots$	Move right.
$\Delta \Delta 0 0 1 X 0 1 \Delta \dots$	Convert to 0 and move left.
$\Delta \Delta 0 0 1 X 0 1 0 \Delta \Delta \dots$	Move left.
$\Delta \Delta 0 0 1 X 0 1 0 \Delta \Delta$	Move left.
$\Delta \Delta 0 0 1 X 0 1 0 \Delta \Delta$	Move right.
$\Delta \Delta 0 0 1 X X 1 0 \Delta \Delta$	Convert 0 to X, move right.
$\Delta \Delta 0 0 1 X X 1 0 \Delta \Delta$	Move right, skip all 0's and 1's.
$\Delta \Delta 0 0 1 X X 1 0 \Delta$	Convert Δ to 0 and move left.
$\Delta \Delta 0 0 1 X X 1 0 \Delta$	Skip all 1's and 0's and reach at X by moving left.
$\Delta \Delta 0 0 1 X X 1 0 0 \Delta$	Move right.

$\Delta \Delta 0 0 1 X X 1 0 0 \Delta$ Move left and convert X to 0's.
 ↑
 $\Delta \Delta 0 0 1 0 0 1 0 0 \Delta$ Move right.
 ↑
 $\Delta \Delta 0 0 1 0 0 1 0 0 \Delta$ Move left, skip all 0's and 1's and reach at Δ by moving left.
 ↑
 $\Delta \Delta 0 0 1 0 0 1 0 0 \Delta$ Move right convert 0 to Δ .
 ↑
 $\Delta \Delta \Delta 0 1 0 0 1 0 0 \Delta$

Now again perform copy operation as illustrated above. The tape will then consist of
 $\Delta \Delta \Delta 0 1 0 0 1 0 0 0 0 \Delta$ Move right convert 0 to Δ and repeat the above illustrated steps.
 ↑
 $\Delta \Delta \Delta \Delta 1 0 0 1 0 0 0 0 0 0$

As we have read out the value of m from $m \times n$ we will convert remaining symbols to Δ leaving only multiplication result on the tape.

Thus finally $\Delta \Delta \Delta \Delta \Delta \Delta \Delta 0 0 0 0 0 0 \Delta$ HALT

→ Example 5.5.2 : Draw a transition diagram for a turing machine accepting the language $\{SS \mid S \in \{a, b\}^*\}$. GTU : Summer-18, Marks 7

Solution : The TM is

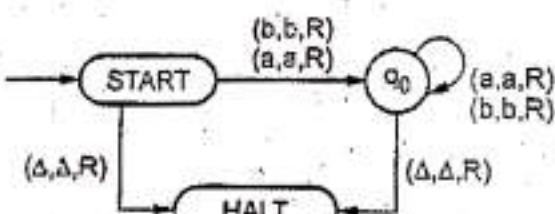


Fig. 5.5.5

5.6 Variations of TM

The turing machine is widely accepted as a model of computation because of its versatility in the power of computation. In this section we are going to see more power of TM by allowing the two way infinite tape as shown in Fig. 5.6.1.

The tape head as usual can move in forward and backward direction.

The turing machine with infinite tape can also be denoted by $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where Q is a finite, nonempty set of states.

Σ is nonempty finite set of input, Γ is the set of external symbols used.

δ is a mapping function which gives the mapping from current state to next state.

q_0 is the initial state.

B is a blank symbol. In the two way infinite tape it is placed at both the ends of the tape. In between these blank symbols the input string has to be placed.

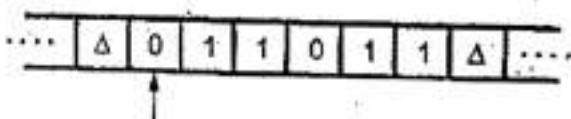


Fig. 5.6.1 Two way infinite tape

F denotes the final state. The TM halts after reaching in final state.

→ **Example 5.6.1 :** Construct a TM for a language having equal number of a 's and b 's in it over the input set $\Sigma = \{a, b\}$.

Solution : Here we will clearly take the advantage of infinite two ends of the input tape. The input can be placed as follows.

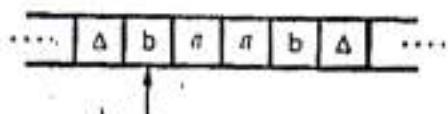


Fig. 5.6.2

We will have a simple logic as if we get a we will replace it by A and move right in search of b , if we get b we will replace it by B and move left. We will move to the leftmost end (i.e. Δ) and again repeat the same process of marking a by A and b by B . We may have string starting with a or b , so from initial state only we will have two separate paths shown for a and b . Remember we have to check both the ends, to check whether the string is completely scanned or not. Let us understand the logic first and then move on for TM.

$\Delta b a a b \Delta$	Convert b to B and move right in search of a
↑	
$\Delta B a a b \Delta$	a has to be converted to A and move left
↑	
$\Delta B A a b \Delta$	Skip B and move left
↑	
$\Delta B A a b \Delta$	Keep Δ as it is and move right
↑	
$\Delta B A a b \Delta$	Move right
↑	
$\Delta B A a b \Delta$	Move right
↑	
$\Delta B A a b \Delta$	Convert a to A and move right
↑	
$\Delta B A A b \Delta$	Convert b to B move left
↑	
$\Delta B A A B \Delta$	Move left
↑	
$\Delta B A A B \Delta$	Move left
↑	
$\Delta B A A B \Delta$	Move left
↑	

$\Delta B A A B \Delta$	If Δ comes move to right by ignoring all A's and B's
\uparrow	
$\Delta B A A B \Delta$	Move right
\uparrow	
$\Delta B A A B \Delta$	Move right
\uparrow	
$\Delta B A A B \Delta$	Move right
\uparrow	
$\Delta B A A B \Delta$	Move right
\uparrow	
$\Delta B A A B \Delta$	Move right
\uparrow	
$\Delta B A A B \Delta$	Δ is reached, goto HALT state
\uparrow	

The TM can be

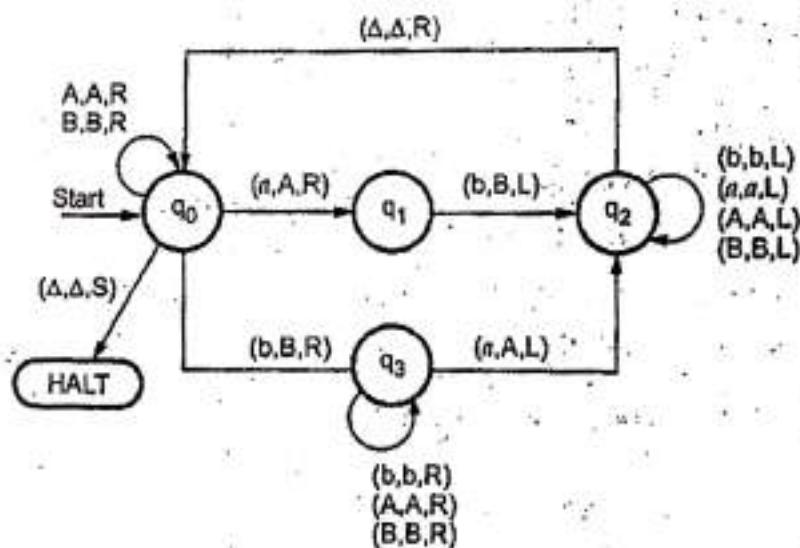


Fig. 5.6.3

Similarly if the string is

$\Delta a b a b \Delta$	Convert a to A move right
\uparrow	
$\Delta A b a b \Delta$	Convert b to B move left
\uparrow	
$\Delta A B a b \Delta$	Skip A and move left
\uparrow	
$\Delta A B a b \Delta$	Move right, ignore all A's and B's
\uparrow	
$\Delta A B a b \Delta$	Convert a to A, move right

$\Delta A B A b \Delta$	Convert b to B, move left
$\Delta A B A B \Delta$	Skip all A's and B's and move left
$\Delta A B A B \Delta$	Move right by skipping all A's and B's
$\Delta A B A B \Delta$	Since both the ends are checked and all the a's and b's are encountered as equal. So TM HALTs here.

Example 5.6.2 : Construct TM for obtaining two's complement of a given binary number.

Solution : The logic for computing two's complement is, we read the binary string from LSB to MSB. From LSB, we keep all the zero's as it is and move left till we do not get 1. After reading first 1 from LSB, we move left and thereafter we convert 0 to 1 and 1 to 0 and go on moving towards left. The process continues upto leftmost Δ .

For example : The binary number 0110 its two's complement can be computed as

$$\begin{array}{r}
 \underline{0 \ 1 \ 1 \ 0} \\
 1 \ 0 \ 0 \ 1 \quad \text{1's complement} \\
 + \quad \underline{1} \quad \text{Adding 1} \\
 = \quad 1 \ 0 \ 1 \ 0 \quad \text{Two's complement of 0110}
 \end{array}$$

By our idea also we will get the same result.

$\Delta 0 1 1 0 \Delta$	Move to rightmost end upto Δ
$\Delta 0 1 1 0 \Delta$	Move left
$\Delta 0 1 1 0 \Delta$	It is 0, so keep it as it is and move left
$\Delta 0 1 1 0 \Delta$	First 1 from LSB has encountered so keep it as it is and move left
$\Delta 0 1 1 0 \Delta$	Complement it to 0 and move left
$\Delta 0 0 1 0 \Delta$	Convert it to 1 and move left
$\Delta 1 0 1 0 \Delta$	Since Δ is reached, stop !

The TM could be

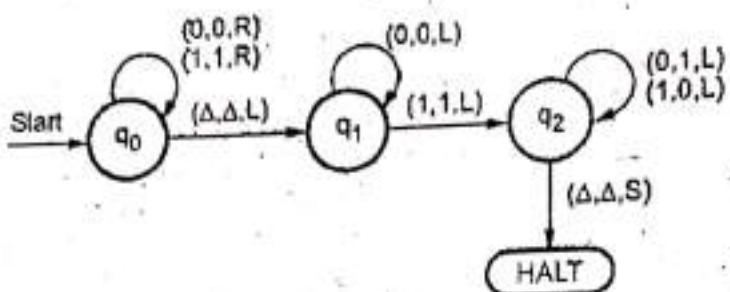


Fig. 5.6.4

Thus two way infinite tape brings significant power to turing machine.

→ Example 5.6.3 : Construct TM for checking well formness of the parenthesis.

Solution : We can solve this problem with PDA as well as with TM also. The parenthesis should be well formed in case of expression solving. The logic which will apply will be very much similar to the logic for finding equal number of a's and equal number of b's. We will start moving towards right, when we get any ')' we will mark it as * and move left in search of '(' . Mark it as *. Thus we will repeat this process for the complete input string.

For example

$\Delta () () \Delta$	Move right in search of ')', skipping '('
$\Delta () () \Delta$	Mark it as * and move left, in search of '('
$\Delta (* () \Delta$	Now ')' has encountered mark it as *. Move left in search of '(', mark it *
$\Delta **() \Delta$	Move right, skip *
$\Delta **() \Delta$	Move right
$\Delta **() \Delta$	Convert to * and move left
$\Delta **(* \Delta$	Convert (to *
$\Delta *** \Delta$	Move right till Δ
$\Delta **** \Delta$	HALT

The TM will look like this -

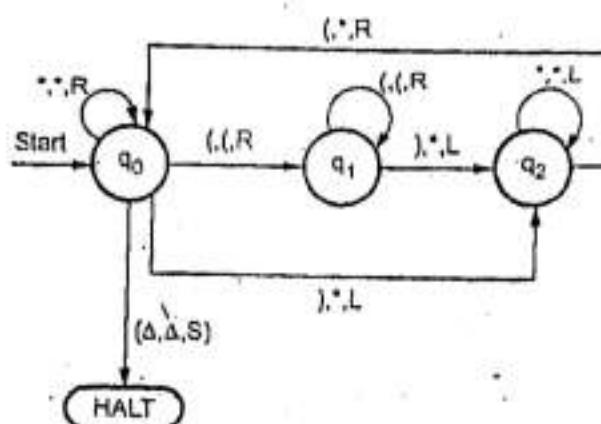


Fig. 5.6.5

The transition table for this TM will be

State	()	*	Δ
q_0	$(q_1, (, R)$	$(q_2, *, L)$	$(q_0, *, R)$	$(HALT, \Delta, S)$
q_1	$(q_1, (, R)$	$(q_2, *, L)$	-	-
q_2	$(q_2, *, R)$	-	$(q_2, *, L)$	-
HALT	-	-	-	-

→ Example 5.6.4 : Construct TM for reversing a binary string on the input tape.

Solution : Suppose on the input tape some binary string is placed, then we will read each character from left to right and copy it at the end of the tape.

For example

... $\Delta 1 0 1 0 0 1 \Delta \Delta \dots$
↑

We will move upto the end of the input string

$\Delta 1 0 1 0 0 1 \Delta \Delta \dots$
↑

Move left

$\Delta 1 0 1 0 0 1 \Delta \Delta \dots$
↑

Mark it * and copy it after Δ

$\Delta 1 0 1 0 0 \cdot \Delta 1$
↑

Move left upto *

$\Delta 1 0 1 0 0 \cdot \Delta 1 \Delta$
↑

Move left, convert 0 to * and place it at the end

$\Delta 1 0 1 0 \cdots \Delta 1 0 \Delta$
↑

Move left upto *

$\Delta 1 0 1 0 \dots \Delta 1 0 \Delta$	Move left, convert 0 to * and place it at the end
$\Delta 1 0 1 \dots \Delta 1 0 0 \Delta$	Move left upto *
$\Delta 1 0 1 \dots \Delta 1 0 0 \Delta$	Move left, convert 1 to * and place it at the end
$\Delta 1 0 \dots \Delta 1 0 0 1 \Delta$	Move left upto *
$\Delta 1 0 \dots \Delta 1 0 0 1 \Delta$	Move left, convert 0 to * and place it at the end
$\Delta 1 \dots \Delta 1 0 0 1 0 \Delta$	Move left skipping *
$\Delta 1 \dots \Delta 1 0 0 1 0 \Delta$	Copy 1 at the end of tape by converting 1 to *
$\Delta \dots \Delta 1 0 0 1 0 1 \Delta$	Move left skipping all *
$\Delta \dots \Delta 1 0 0 1 0 1 \Delta$	Left to * is a Δ character so we goto HALT state

The TM will be

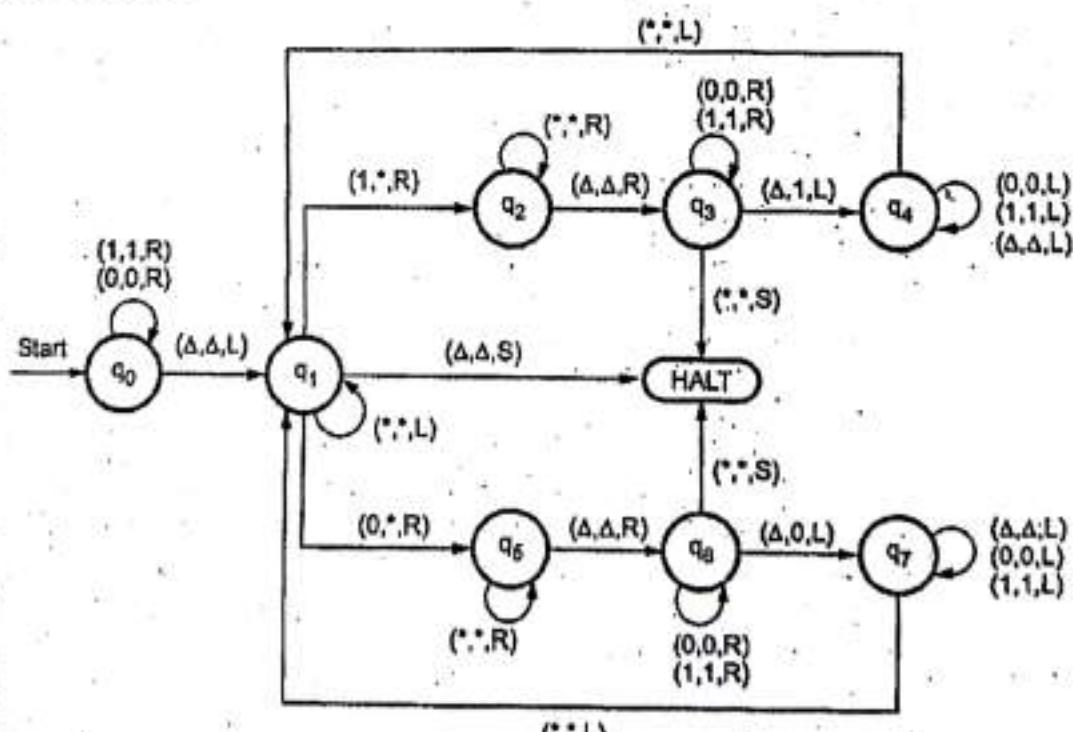


Fig. 5.6.6

The transition table for the above TM will be

	0	1	*	Δ
q_0	$(q_0, 0, R)$	$(q_0, 1, R)$	-	(q_2, Δ, L)
q_1	$(q_5, *, R)$	$(q_2, *, R)$	$(q_1, *, L)$	$(HALT, \Delta, S)$
q_2	-	-	$(q_2, *, R)$	(q_3, Δ, R)
q_3	$(q_3, 0, R)$	$(q_3, 1, R)$	-	$(q_4, 1, L)$
q_4	$(q_4, 0, L)$	$(q_4, 1, L)$	$(q_1, *, L)$	(q_4, Δ, L)
q_5	-	-	$(q_5, *, R)$	(q_6, Δ, R)
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	-	$(q_7, 0, L)$
q_7	$(q_7, 0, L)$	$(q_7, 0, L)$	$(q_1, *, L)$	(q_7, Δ, L)
HALT	-	-	-	-

Thus TM reverses the input string.

→ Example 5.6.5 : Draw a transition diagram for a turing machine accepting the following language.

$$\{ x \in \{a, b, c\}^* \mid n_a(x) = n_b(x) = n_c(x) \}$$

GTU : Winter-11, Marks 7

Solution : The logic for TM shown in Fig. 5.6.7 is represented by deriving following string. (See Fig. 5.6.7 on next page)

$\Delta a b a c b \Delta$ ↑	Read a, convert it to A and move right. The resultant state will be q_1 .
$\Delta A b a c b c \Delta$ ↑	Read b, convert it to B and move right. The resultant state will be q_5 .
$\Delta A B a c b c \Delta$ ↑	Read a, and move right.
$\Delta A B a c b c \Delta$ ↑	Read c convert it to C and move left. Thus we have read one a, one b and one c.
$\Delta A B a C b c \Delta$ ↑	Just move towards the right most symbol.
$\Delta A B a C b c \Delta$ ↑	Now we again repeat the above procedure.
$\Delta A B a C b c \Delta$ ↑	Now just skip the capital letters and move right.
$\Delta A B a C b c \Delta$ ↑	Read a, convert it to A and move right. The resultant state will be q_1 .
$\Delta A B A C b c \Delta$ ↑	Move right.
$\Delta A B A C b c \Delta$ ↑	Read b convert it to B and move right. The resultant state will be q_5 .

$\Delta A B A C B C \Delta$	Read C, convert it to C and move left. The resultant state will be q10.
$\Delta A B A C B C \Delta$	Now just skip all the capital letters and move left, towards leftmost Δ .
$\Delta A B A C B C \Delta$	Now skip all the capital letters and move towards rightmost Δ .
$\Delta A B A C B C \Delta$	Accept

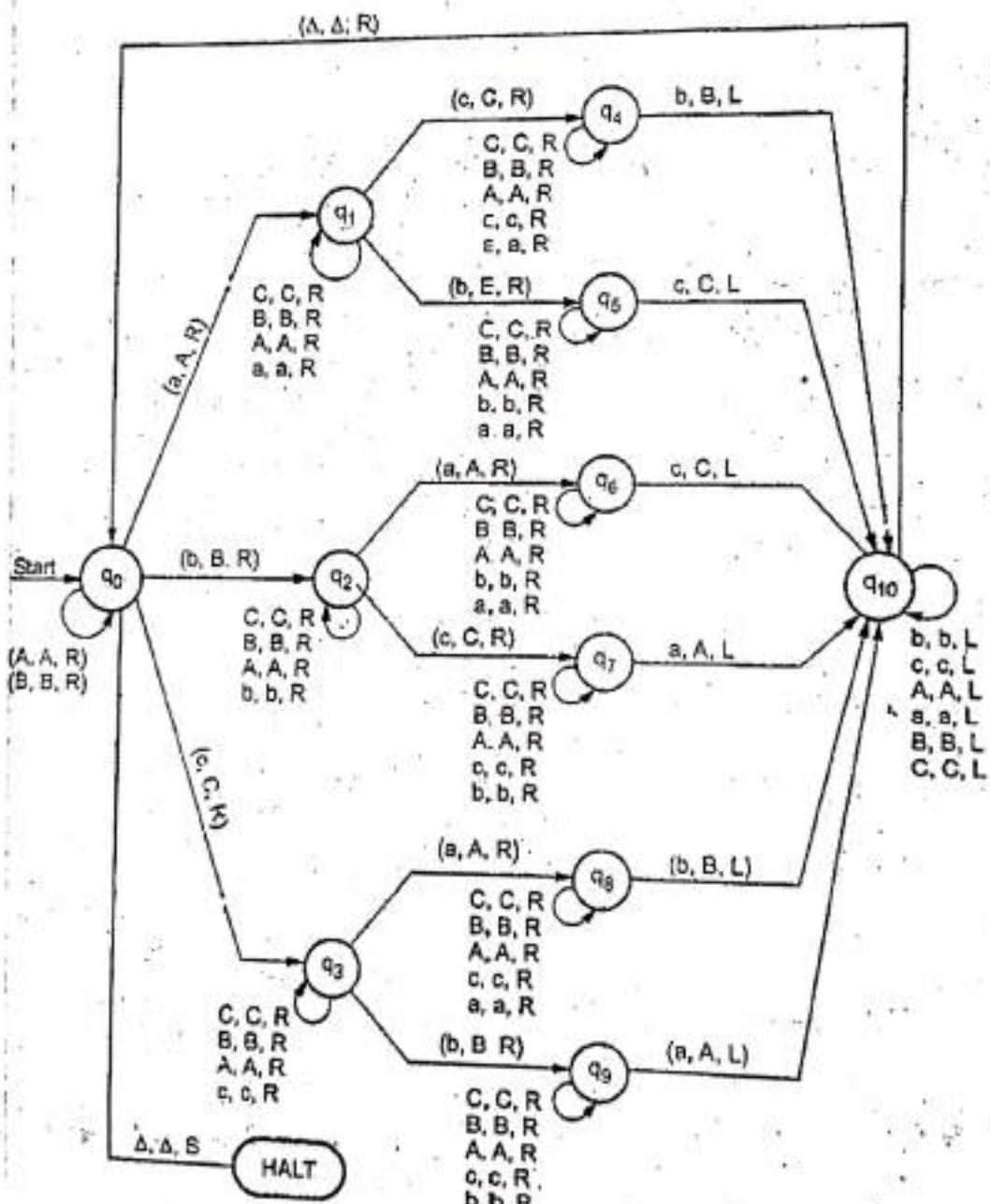


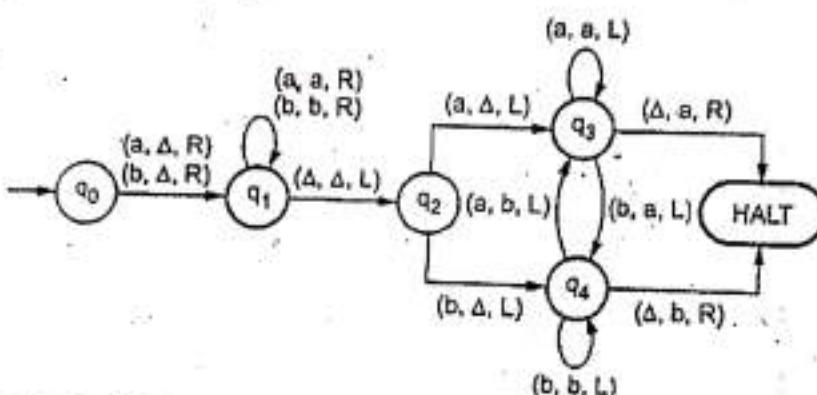
Fig. 5.6.7

Example 5.6.6 : Draw the TM to copy string and delete a symbol.

GTU : Summer-14, Winter-14, Marks 7

Solution : In this TM we will simply delete the first symbol and then copy the remaining string on the tape.

Consider input set $\Sigma = \{a, b\}$



Simulation of a b a b b b Δ

String	Action
a b a b b b Δ ↑	Convert a to Δ and move right (q ₀ to q ₁)
Δ b a b b b Δ ↑	Now skip all a's and b's and move on to Δ (q ₁ to q ₁)
Δ b a b b b Δ ↑	Keep Δ as it is and move left (q ₁ to q ₂)
Δ b a b b b Δ ↑	Convert b to Δ and move left (q ₂ to q ₄)
Δ b a b b Δ Δ ↑	Keep b as it is and move left.
Δ b a b Δ Δ ↑	Convert a to b and move left.
Δ b b b Δ Δ ↑	Convert b to a and move left.
Δ a b b Δ Δ ↑	Convert Δ to b and move right
b a b b Δ Δ ↑	HALT

Example 5.6.7 : Design a TM for simulating the copy function.

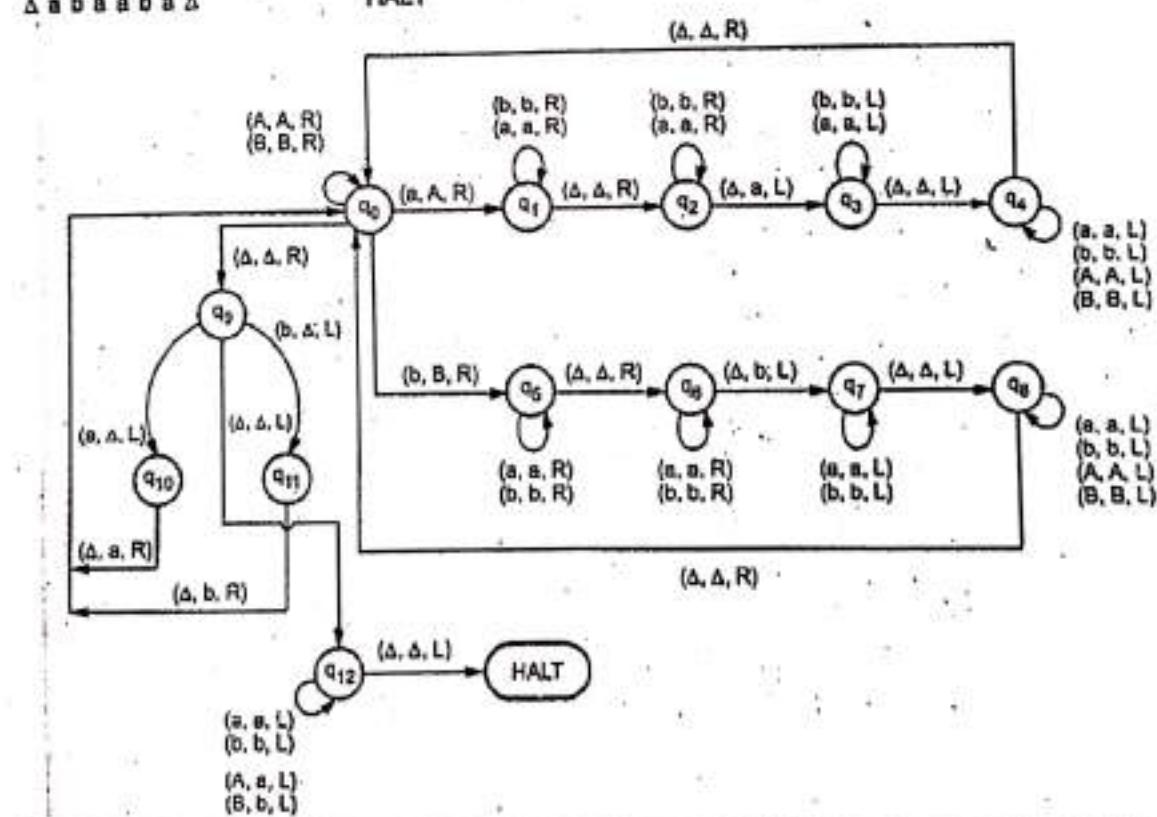
GTU : Winter-17, Marks 7

Solution : The logic will be read a and convert it to A. Move right at the end and copy a. Similarly if we read b convert it to B. Move right at the end and copy b. Finally we will get SΔS. Now eliminate this Δ and shift S ahead so that SSΔ remains on the tape.

Suppose we have input aba on the input tape then, the tape will be $\Delta a b a \Delta \dots$

After some transitions

$\Delta A B A \Delta a b a \Delta$	Move right convert a to Δ
$\Delta A B A \Delta \Delta b a \Delta$	Move left convert Δ to a
$\Delta A B A a \Delta b a \Delta$	Move right
$\Delta A B A a \Delta b a \Delta$	Move right
$\Delta A B A a \Delta b a \Delta$	Convert b to Δ and move left
$\Delta A B A a \Delta \Delta a \Delta$	Move Δ to be and move right
$\Delta A B A a b \Delta a \Delta$	Move right convert a to Δ
$\Delta A B A a b \Delta \Delta a$	Move left convert Δ to a
$\Delta A B A a b a \Delta \Delta$	Move right
$\Delta A B A a b a \Delta \Delta$	Move right
$\Delta A B A a b a \Delta \Delta$	Move left and convert all A to a and B to b
$\Delta a b a a b a \Delta$	HALT



Example 5.6.8 : Draw TM for $L = \{SS / S \in (a,b)^*\}$

GTU : Summer-12, Marks 7

Solution : This TM suggest to accept the strings of the form SS where $S \in (a,b)^*$. That is {aa, bb, abab, aaaa, bbbb, baba, ...} are valid strings. Also note that these strings are always even length strings.

The logic is as follows :-

- 1) Firstly we will find middle of the string which will help us to distinguish first and second part of the given string.
- 2) Then we will try to find out whether first part exactly matches with the second part or not. This can be done by reading symbol from second part and check the corresponding symbol in the first part.

The TM can be designed as follows :

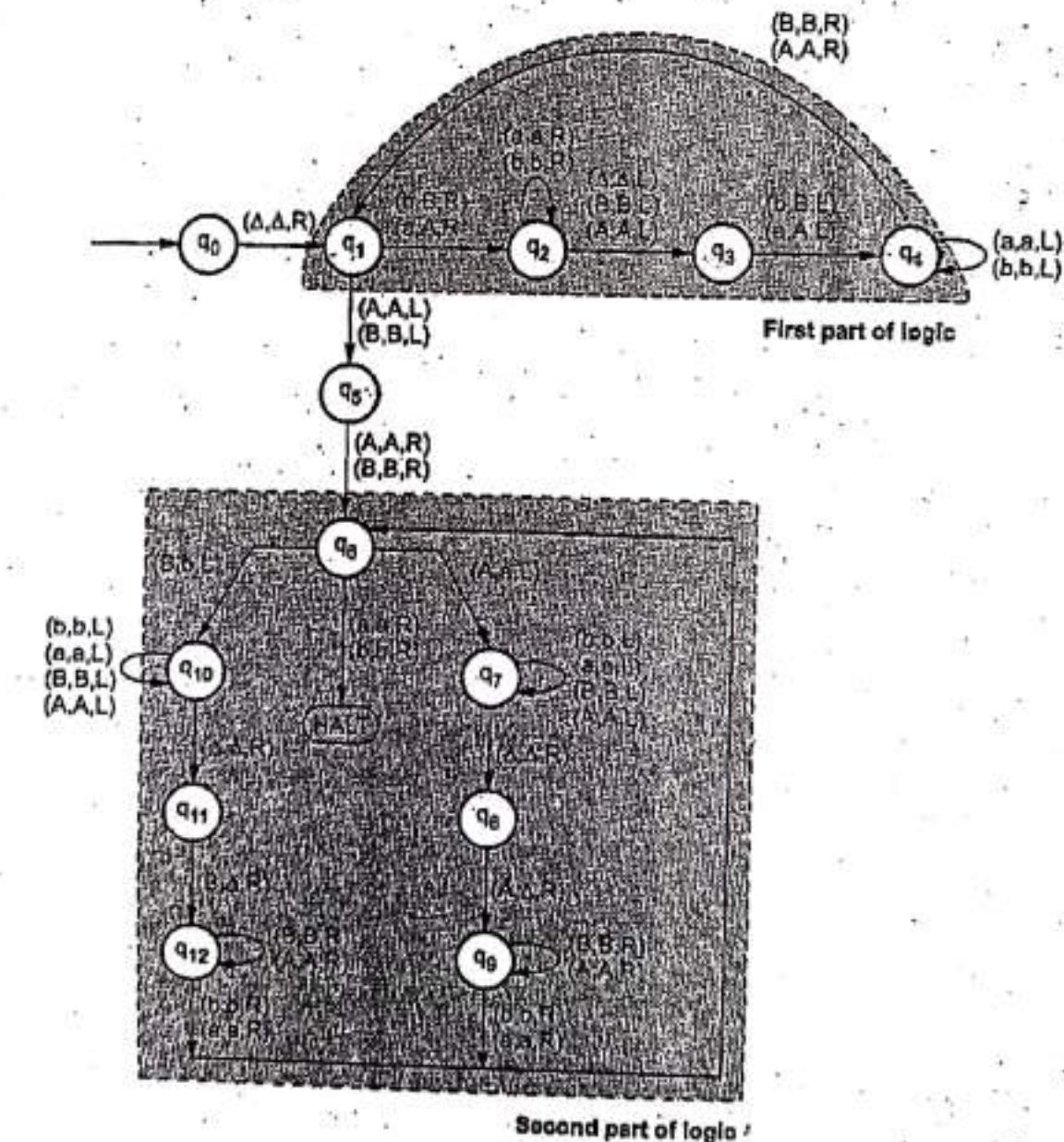


Fig. 5.6.8

Simulation of $\Delta b a b a \Delta$

Symbol read	Action to be taken
$\Delta b a b a \Delta$ ↑	Read Δ , keep it as it is and move right. state q_0
$\Delta b a b a \Delta$ ↑	Convert b to B and move right. q_1 state.
$\Delta B a b a \Delta$ ↑	Skip all a's and b's and move on right upto Δ . Refer q_2 .
$\Delta B a b a \Delta$ ↑	Move left.
$\Delta B a b a \Delta$ ↑	Convert a to A and move left. State q_3 .
$\Delta B a b A \Delta$ ↑	Skip all a's and b's and move left towards capital letter. Refer q_4 .
$\Delta B a b A \Delta$ ↑	Keep B as it is and move right. Refer q_5 .
$\Delta B a b A \Delta$ ↑	Convert a to A and move right. Refer q_6 .
$\Delta B A b A \Delta$ ↑	Skip b and move right upto first capital letter.
$\Delta B A B A \Delta$ ↑	Move left and then convert b to B and move left.
$\Delta B A B A \Delta$ ↑	First part is over. We have obtained mid. Read A and move right. State q_4 and q_1 .
$\Delta B A B A \Delta$ ↑	Read B and move left q_1 to q_5 .
$\Delta B A B A \Delta$ ↑	Read A and move right. q_5 to q_6 .
$\Delta B A B A \Delta$ ↑	Convert B to b and move left. q_6 to q_{10} .
$\Delta B A b A \Delta$ ↑	Now go on moving left upto Δ .
$\Delta B A b A \Delta$ ↑	Move just right. q_{10} to q_{11} .
$\Delta B A b A \Delta$ ↑	Convert B to Δ and move right. State q_{11} to q_{12} . Here we have checked match of second part symbol with first part symbol. The second part symbol is converted to lower letter while first part symbol is converted to Δ .
$\Delta \Delta A b A \Delta$ ↑	Just skip all capital letters and move right. State q_{12} .

$\Delta \Delta A b A \Delta$	Move right by keeping b as it is. State q_{12} to q_0 .
$\Delta \Delta A b A \Delta$	Convert A to a and move left.
$\Delta \Delta A b a \Delta$	Keep going left and convert A to Δ . Refer q_7 to q_{19} .
$\Delta \Delta \Delta b a \Delta$	Move right and HALT.

→ Example 5.6.9 : Design a Turing machine which accepts the language consisting string which contain aba as a substring over alphabets {a, b}. **GTU : Summer-19, Marks 7**

Solution :

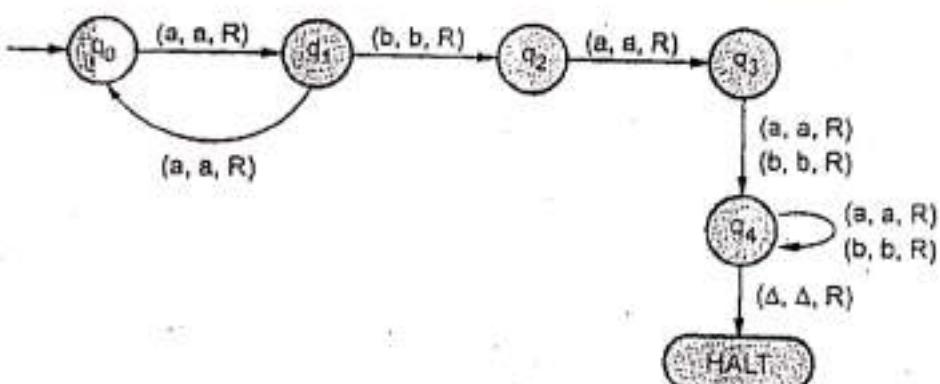


Fig. 5.6.9

→ Example 5.6.10 : Draw a Turing machine that accepts the language $\{xx \mid x \in \{a, b\}^*\}$. Also trace the TM on input string aa. **GTU : Winter-19, Marks 7**

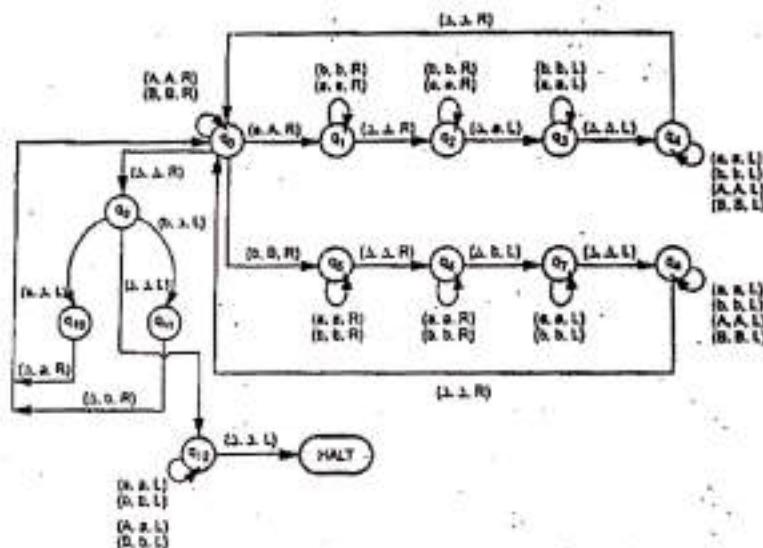
Solution : The logic will be read a and convert it to A. Move right at the end and copy a. Similarly if we read b convert it to B. Move right at the end and copy b. Finally we will get S Δ S. Now eliminate this Δ and shift S ahead so that SS Δ remains on the tape.

Suppose we have input aba on the input tape then, the tape will be $\Delta a b a \Delta \dots$

After some transitions

$\Delta A B A \Delta a b a \Delta$	Move right convert a to Δ
$\Delta A B A \Delta \Delta b a \Delta$	Move left convert Δ to a
$\Delta A B A a \Delta b a \Delta$	Move right
$\Delta A B A a \Delta b a \Delta$	Move right
$\Delta A B A a \Delta b a \Delta$	Convert b to Δ and move left
$\Delta A B A a \Delta \Delta a \Delta$	Move Δ to be and move right
$\Delta A B A a \Delta a \Delta a \Delta$	Move right convert a to Δ

$\Delta A B A a b \Delta \Delta \Delta$
 ↑ Move left convert Δ to a
 $\Delta A B A a b b \Delta \Delta$
 ↑ Move right
 $\Delta A B A a b b \Delta \Delta$
 ↑ Move right
 $\Delta A B A a b b \Delta \Delta$
 ↑ Move left and convert all A to a and B to b
 $\Delta a b a a b a \Delta$
 ↑ HALT



→ Example 5.6.11 : Draw a Turing machine that accepts the language $\{a^n b^n a^n \mid n \geq 0\}$ over $(a, b)^*$. Also trace the TM on input string aaabbbbaaa.

GTU : Winter-19, Marks 7

Solution : The TM for $\{L = a^n b^n a^n\}$ is

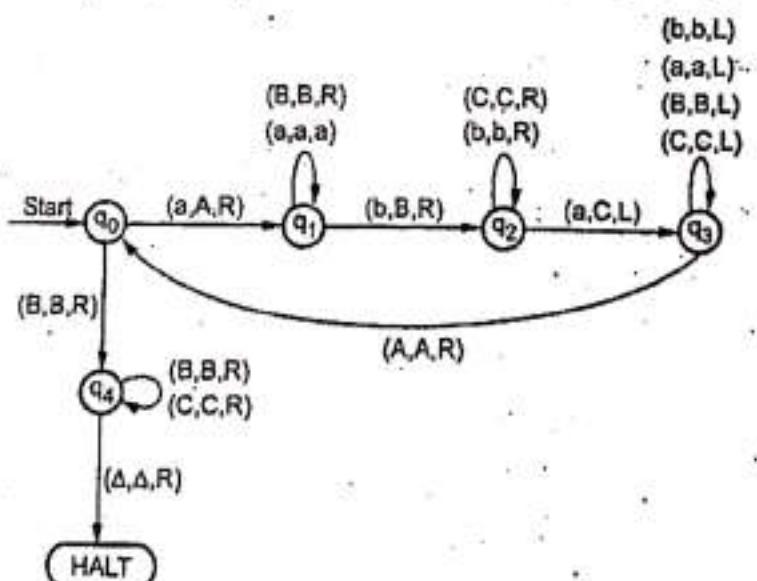


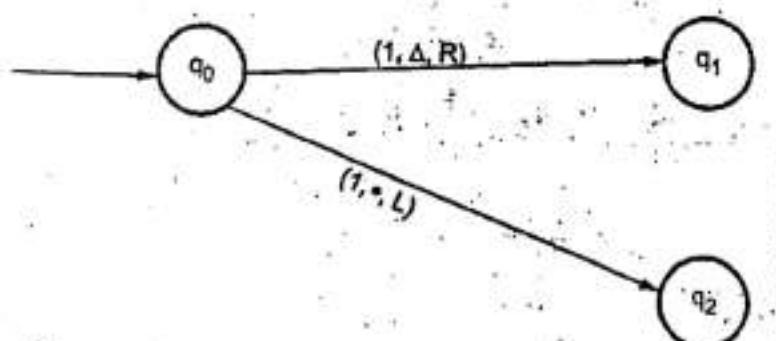
Fig. 5.6.10

Step	Input	Action
1.	aabbbaaaΔ	Read a, convert to A and keep moving right upto b.
2.	Aaa bbbbaaaΔ	Read b, convert it to B and keep moving right upto a:
3.	AaaBbbbaaaΔ	Read a, convert it to C and move left.
4.	AaaBbb b CaaΔ	Now keep moving left upto A.
5.	Aaa B bbCaaΔ	Move right.
6.	AaaBbbCaaΔ	Repeat step 1 to 5.
7.	AAABBBCCCCΔ	Move right.
8.	AAABBBCCCCΔ	Just keep moving right by skipping all B's and C's.
9.	AAABBBBCCCCΔ	Move right and HALT.

5.7 Non Deterministic TM

The non-deterministic TM is a machine which prescribes more than one different actions for the same input.

For example



That means if there are two different actions specified such as being "in state q_0 for input 1, change it to Δ and go to right" and "in state q_0 , for input 1, change it to * and go to left".

The non-deterministic turing machine is a collection of following components

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \Delta)$$

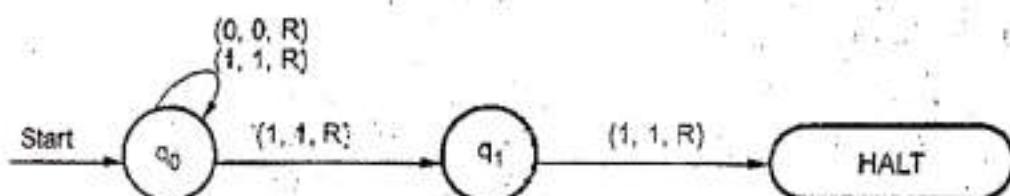
where

- 1) Q is a finite set of states

- 2) Γ is finite set of external symbols.
- 3) Σ is a finite set of input symbols.
- 4) δ is a transition function or mapping function. There can be more than one different transition functions for same input.
- 5) q_0 is initial state
- 6) Δ is blank character.

For example

Following is a turing machine which accepts the language in which each string ends with 11.



Any language which is accepted by Deterministic Turing Machine is also accepted by Non-deterministic Turing Machine. Hence NTM is as powerful as DTM.

5.8 Universal TM

The universal language L_u is a set of binary strings which can be modeled by a turing machine. The universal language can be represented by a pair (M, w) where M is a TM that accepts this languages and w is a binary string in $(0 + 1)^*$ such that w belongs to $L(M)$. Thus we can say that any binary string belongs to a universal language. From this the concept of universal Turing machine came up. The universal language can be represented by $L_u = L(U)$ where U is a universal turing Machine. In fact U is a binary string. This binary string represents various codes of many turing machines. Thus the universal turing machine is a turing machine which accepts many turing machines. The TM U is a multitape Turing machine which can be as shown below.

- The universal turing machine U accepts the TM M .
- The transitions of M are stored initially on first tape along with the string w .
- On the second tape the simulated tape of M is placed. Here the tape symbol x_i of M will be represented by 0's and tape symbols are separated by single 1's.
- On the third tape various states of machine M are stored. The state q_i is represented by i C's.

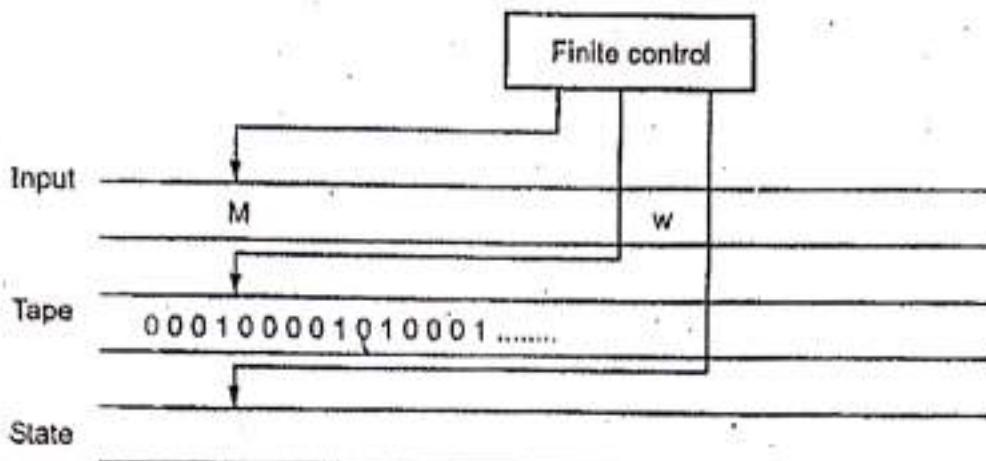


Fig. 5.8.1

Operations on Turing Machine

1. The first tape is observed carefully to check whether the code for M is valid or not. If the code is not valid then U halts without accepting. As invalid codes are assumed to represent the TM with no moves, such TM accepts no inputs and halts.
2. Initialize the second tape for placing the tape values of M in encoded form that means for each 0 of the input w place 10 on the second tape and for each 1 place 100 there. The blank letters of tape of machine M are represented by 1000. But actually blank appears at the end of input w for the machine U. However the blanks of M are simulated by 1000.
3. As the third tape stores states of machine M, place 0 there as the start state of machine M. The head of third tape will now position at start state (i.e. at 0) and the head of third tape will now position at start state (i.e. at 0) and the head for the tape 2 will position at the first simulated cell of second tape.
4. Now the move of M can be simulated. As on the first tape transitions are stored U first searches the tape 1 to know the transitions which are always given in the form $0^i 10^j 10^k 10^l 10^m$; as we have already seen that such a turing machine code consists of

0^i means current state,

0^j means current input tape symbol,

0^k means changed state,

0^l means changed input tape symbol,

0^m means direction in which the head is to be moved.

Now to simulate the move of M change the content of tape 3 to 0^k that is, simulate the state change of M. Then replace 0^i on tape 2 by 0^l , that is simulate the change in input. Then move the tape head on tape to the position of next 1. The tape head moves to left or right depending upon the value of m. That means if $m = 1$ then move left and if $m = 2$ move right. Thus U simulates the move of M to left or to the right.

5. If M has no transition then no transition is performed. Therefore M halts in the simulated configuration and hence U halts.
6. If M enters in accept state, then U accepts.

Thus the TM U simulates M on binary input string w where a pair (M, w) is also coded in binary form.

Review Question

1. Explain Universal Turing Machine.

GTU : Summer-16, Marks 4; Winter-17, 18, Summer-19, Marks 7

5.9 Recursive and Recursively Enumerable Languages

The recursively enumerable language has two categories. The first category consists of all the languages that has some algorithm and an algorithm for language L can be modeled by a turing machine. Naturally such turing machine always halts on valid input and enters in accept state, but on invalid input (the input that is not belonging to language L) it halts without entering in halt state. Thus languages of this category possess a property of definiteness. The languages of this category are particularly called as recursive languages.

In the second category, the languages can be modeled by turing machines but there is no guarantee that the TM will eventually halt. In the sense that we cannot predict that turing machine will halt or will enter in an infinite loop for certain input. Such type of languages can be denoted by pair (M, w) where M is a turing machine and w is an input string. These languages of this category are typically called as Recursively Enumerable languages (RE).

5.9.1 Properties of Recursive and Recursively Languages

A language is L actually denoted by $L(M)$ called recursive if it is accepted by some turing machine such that

1. If string w is in L , turing machine M accepts it and then halts.
2. If w is not in L , then M eventually halts although it never enters in accepting states.

We can also call recursive languages as the definite languages or the languages which can be represented by some algorithm and such an algorithm helps in construction of Turing machine for that language.

Decidable and Undecidable Languages - If a language is recursive then it is called decidable languages and if the language is not recursive then such a language is called undecidable language.

Hence broadly there are three categories of the languages.

1. Recursive language for which the algorithm exists.
2. Recursively enumerable language for which it is not sure that on which input the TM will ever halt. Such languages are not recursive.
3. The non-recursively enumerable languages for which there is no Turing machine at all.

Fig. 5.9.1 shows the relationship between these languages

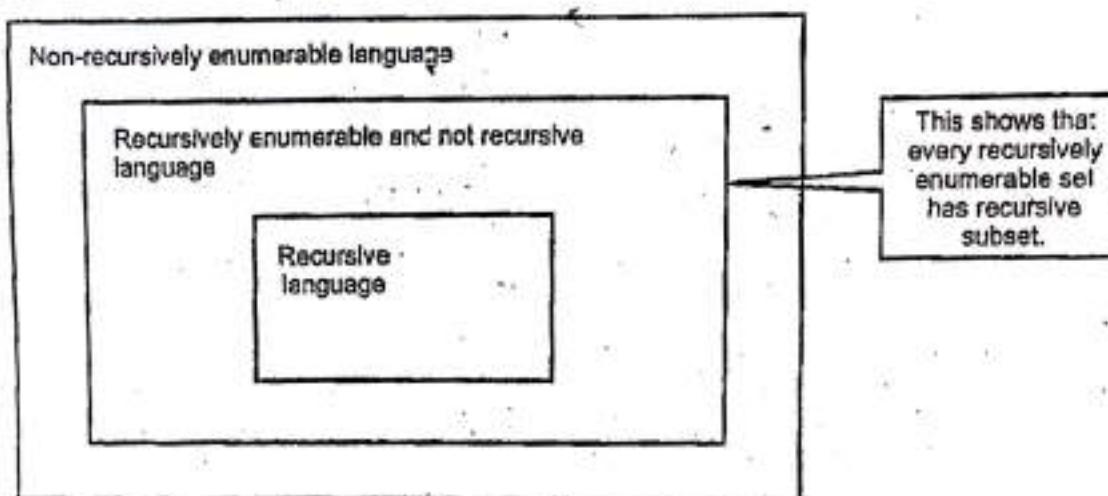


Fig. 5.9.1 Relationship between languages

Review Questions

1. Describe recursive languages and recursively enumerable languages. **GTU : Winter-18, Marks 4**
2. What is decidability? How to prove that the given language is unpredictable? List some undecidable problems. **GTU : Summer-19, Marks 3**

5.10 Context Sensitive Languages

The context sensitive languages are the languages which are defined by context sensitive grammar. In this grammar more than one terminal or non-terminal symbol may appear on the left hand side of the production rule. Along with it, the context sensitive grammar follows following rules -

- i) The number of symbols on the left hand side must not exceed number of symbols on the right hand side.
- ii) The rule of the form $A \rightarrow \epsilon$ is not allowed unless A is a start symbol. It does not occur on the right hand side of any rule.

The classic example of context sensitive language is $L = \{a^n b^n c^n | n \geq 1\}$. The context sensitive grammar can be written as -

$$\begin{aligned}S &\rightarrow aBC \\S &\rightarrow SABC\end{aligned}$$

$CA \rightarrow AC$ $BA \rightarrow AB$ $CB \rightarrow BC$ $aA \rightarrow aa$ $aB \rightarrow ab$ $bB \rightarrow bb$ $bC \rightarrow bc$ $cC \rightarrow cc$

Now to derive the string aabbcc we will start from start symbol -

S	rule $S \rightarrow SABC$
<u>SABC</u>	rule $S \rightarrow aBC$
<u>aBC<u>A</u>BC</u>	rule $CA \rightarrow AC$
<u>aBAC<u>B</u>C</u>	rule $CB \rightarrow BC$
<u>a<u>B</u>ABCC</u>	rule $BA \rightarrow AB$
<u>aAB<u>B</u>CC</u>	rule $aA \rightarrow aa$
<u>aa<u>B</u>BBCC</u>	rule $aB \rightarrow ab$
<u>aab<u>B</u>CC</u>	rule $bB \rightarrow bb$
<u>aabb<u>C</u>CC</u>	rule $bC \rightarrow bc$
<u>aabb<u>c</u>C</u>	rule $cC \rightarrow cc$
aabbcc	

Note that The language $a^n b^r c^n$ where $n \geq 1$ is represented by context sensitive grammar but it can not be represented by context free grammar.

→ Example 5.10.1 : Differentiate regular grammars and context sensitive grammars.

GTU : Summer-14, Marks 7

Solution : Regular grammar - The rules of regular grammar are written either in left linear or right linear form. They can be represented using finite automata or push down automata. They are simple to represent and are less powerful.

Context sensitive grammars -

On the other hand rules of context sensitive languages are of the following form -

$\alpha A \beta \rightarrow \alpha \gamma \beta$

where A is non terminal symbol, α and β are any number of terminals and non terminals and γ is one or more number of terminals or non terminals. The context sensitive grammar can be represented by linear bounded automata.

Example 5.10.2 : Define context-sensitive grammar. What is the language of following context-sensitive grammar ?

$$S \rightarrow aTb \mid ab$$

$$aT \rightarrow aaTb \mid ac.$$

GTU : Winter-18, Marks 3

Solution : Context sensitive grammar : Refer section 5.10.

Consider given c.s. grammar. We will form various derivations in order to recognize the language.

S	S	S
<u>aTb</u>	<u>aTb</u>	ab
aa <u>Tbb</u>	aa <u>Tbb</u>	This derivation denotes ab
aaa <u>Tbbb</u>	aaa <u>Tbbb</u>	
aaaa <u>bbb</u>	This derivation denotes $a^n c b^n$	
This derivation denotes $a^n c b^n$		

Hence the Language specified by given context sensitive grammar is,

$$L(G) = \{ab\} \cup \{a^n c b^n \mid n > 0\}$$

Example 5.10.3 : Define context sensitive language and context sensitive grammar. Write CSG for $L = \{a^n b^n c^n \mid n \geq 1\}$.

GTU : Winter-19, Marks 3

Solution : Context Sensitive Languages

The context sensitive languages are the languages which are defined by context sensitive grammar. In this grammar more than one terminal or non terminal symbol may appear on the left hand side of the production rule. Along with it, the context sensitive grammar follows following rules -

- i) The number of symbols on the left hand side must not exceed number of symbols on the right hand side.
- ii) The rule of the form $A \rightarrow \epsilon$ is not allowed unless A is a start symbol. It does not occur on the right hand side of any rule.

The classic example of context sensitive language is
 $L = \{a^n b^n c^n \mid n \geq 1\}$. The context sensitive grammar can be written as -

$$S \rightarrow aBC$$

$$S \rightarrow SABC$$

$CA \rightarrow AC$
 $BA \rightarrow AB$
 $CB \rightarrow BC$
 $aA \rightarrow aa$
 $aB \rightarrow ab$
 $bB \rightarrow bb$
 $bC \rightarrow bc$
 $cC \rightarrow cc$

Now to derive the string aabbcc we will start from start symbol -

S	rule $S \rightarrow SABC$
<u>SABC</u>	rule $S \rightarrow aBC$
<u>aBCABC</u>	rule $CA \rightarrow AC$
<u>aBACBC</u>	rule $CB \rightarrow BC$
<u>aBABCC</u>	rule $BA \rightarrow AB$
<u>aABBCC</u>	rule $aA \rightarrow aa$
<u>aaBBCC</u>	rule $aB \rightarrow ab$
<u>aabBBC</u>	rule $bB \rightarrow bb$
<u>aabbCC</u>	rule $bC \rightarrow bc$
<u>aabbcC</u>	rule $cC \rightarrow cc$
aabbcc	

Note that The language $a^n b^n c^n$ where $n \geq 1$ is represented by context sensitive grammar but it can not be represented by context free grammar.

5.11 Chomsky Hierarchy

The Chomsky's Hierarchy represents the class of languages that are accepted by different machine. The category of languages in Chomsky's Hierarchy is as given below -

Language class	Language	Grammar	Machine	One example
Type 3	Regular	Regular grammar	FSM i.e. NFA or DFA	$a^* b^*$
Type 2	Context free	Context free grammar	PDA	$a^n b^n$
Type 1	Decidable languages	Context sensitive grammar	Linear bounded automata	$a^n b^n c^n$
Type 0	Computable languages	Unrestricted grammar	Turing machine	nil

This is a hierarchy therefore every language of type 3 is also of type 2, 1 and 0. Similarly every language of type 2 is also of type 1 and 0 etc.

Type 3 - Regular languages

Regular languages are those languages which can be described using regular expressions. These languages can be modelled by NFA or DFA.

Type 2 - Context free languages

The context free languages are the languages which can be represented by Context Free Grammar (CFG). The production rule is of the form

$$A \rightarrow \alpha$$



Fig. 5.11.1 Chomsky hierarchy

where A is any single non-terminal and α is any combination of terminals and non-terminals.

A NFA or DFA cannot recognize strings of this language because these automata are not having "stack" to memorize. Instead of it the Push Down Automata can be used to represent these languages.

Type 1 - Context sensitive languages

The context sensitive grammars are used to represent context sensitive languages. The context sensitive grammar follows the following rules -

1. The context sensitive grammar may have more than one symbol on the left hand side of their production rules.
2. The number of symbols on the left hand side must not exceed the number of symbols on the right hand side.
3. The rule of the form $A \rightarrow \epsilon$ is not allowed unless A is a start symbol. It does not occur on the right hand side of any rule.

The automaton which recognizes context sensitive languages is called linear bounded automaton. While deriving using context sensitive grammar the sentential form must always increase in length every time a production rule is applied. Thus the size of a sentential form is bounded by a length of the sentence we are deriving.

Type 0 - Unrestricted languages

There is no restriction on the grammar rules of these type of languages. These languages can be effectively modeled by Turing machines.

Example 5.11.1 : Give the formal definition of Turing machine. Also compare the power of DFA, NFA, DPDA, NDPA and TM. GTU : Summer-19, Marks 3.

Solution : Powers of DFA, NFA, NPDA, PDA and TM :

- 1) The finite automata is of two types : Deterministic Finite Automata (DFA) and Non-Deterministic Finite Automata (NDFA or NFA).
Both DFA and NFA accept regular language only. Hence both the machines have equal power.
- 2) The push down automata is of two types : Deterministic pushdown automata (DPDA) and non-deterministic pushdown automata (NDPDA).
The power of DPDA is more than NFA or DFA as it contains stack memory. Similarly NPDA accepts larger class than DPDA. Hence NPDA is more powerful than PDA.
- 3) The TM can be programmed. Hence, it accepts larger class of languages than NPDA. Hence TM is most powerful model.
 $\therefore \text{TM} > \text{NPDA} > \text{PDA} > \text{NFA or DFA}$.

Review Question

1. Define grammar and Chomsky hierarchy.

GTU : Winter-18, Marks 3

5.12 Short Questions and Answers

Q.1 Who invented Turing machine ?

Ans. : Alan Turing

Q.2 Turing machine is more powerful than _____.

- a finite automata
 b push down automata
 c both a and b

[Ans. : c]

Q.3 What are the ways of representing Turing machine ?

Ans. : a. Transition table b. Transition diagram c. Instantaneous description

- Q.4** A PDA behaves like Turing machine when the number of auxiliary memory it has is _____.
 a zero b one or more
 c two or more d none of these [Ans. : c]
- Q.5** Turing Machine (TM) is more powerful than FMS (Finite State Machine) because _____.
 a tape movement is confined to one direction
 b it has no finite state
 c it has the capability to remember arbitrarily long sequences of input symbols
 d none of these [Ans. : c]
- Q.6** What is the purpose of instantaneous description for Turing machine ?
Ans. : The Instantaneous description of Turing machine represents the present state, the next state, the input symbol to be processed and the output to be written on the tape.
- Q.7** A turing machine that is able to simulate other turing machines :
 a Nested turing machines b Universal turing machine
 c Counter machine d None of the mentioned [Ans. : b]
- Q.8** Give any one reason that makes Turing machine more powerful than FSM.
Ans. : Turing machine has head movement in both left and write directions and it can write the output symbol on the input tape.
- Q.9** Universal TM influenced the concept of _____.
 a stored program computers
 b interpretative implementation of programming language
 c computability d all of these [Ans. : d]
- Q.10** The number of external states of Universal Turing Machine should be at least _____.
 a 1 b 2
 c 3 d none of these [Ans. : b]
- Q.11** If Turing machine accepts all the words of language L and rejects or loops for other words which are not in L then L is said to be _____.
Ans. : recursively enumerable language.
- Q.12** We can think of Turing machine's transition function as _____.
 a computer system b software
 c hardware d none of these [Ans. : b]



Computable Functions

6.1 Partial, Total and Constant Functions

The problem of finding out whether a given problem is solvable or not is based on evaluation of function on the set of natural number or by a given alphabet.

Let us start by defining three important types of functions.

1. Partial function

A partial function f from A to B is a rule which assigns every element of A to at the most one element of B . (As shown Fig. 6.1.1)

2. Total function

A total function f from A to B is a rule which assigns to every element of A a unique element of B . (As shown Fig. 6.1.2)

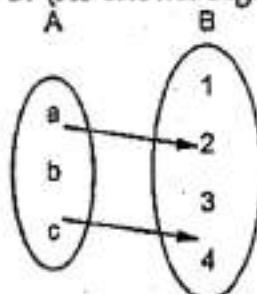


Fig. 6.1.1 Partial function

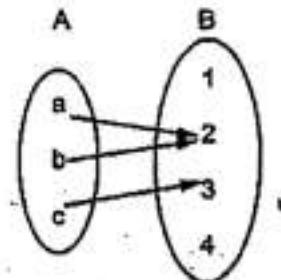


Fig. 6.1.2 Total function

3. Constant function

A constant function $f(x)$ over n and every k (natural numbers) using recursion can be defined as

$$f(0) = k$$

$$f(n + 1) = g(n, f(n))$$

For example

- 1) $z(x) = 0$ zero function
- 2) $f(x) = x$ identity function

6.2 Primitive Recursive Functions

Recursive function is class of functions those are turing computable. The theory of recursive functions is just converse to Church's hypothesis. Recursive function theory begins with some very elementary functions that are intuitively effective. Then it provides a few methods for building more complicated functions from simpler functions. That means the computability of given function can be proved using the initial function and some building operations which are few in number.

Initial function : The initial functions are the elementary functions whose values are independent of their smaller arguments. The following functions comprise the class of recursive functions.

The zero function : $z(x) = 0$

The successor function : $s(x) = \text{successor of } x$ (roughly, " $x+1$ ")

The identity function : $\text{id}(x) = x$

The zero function returns zero regardless of its argument.

The successor function returns the successor of its argument. Since successorship is a more primitive notion.

The zero and successor functions take only one argument each. But the identity function is designed to take any number of arguments. When it takes one argument (as above) it returns its argument as its value. When it takes more than one argument, it returns one of them. That means,

$$\text{id}(x,y) = x$$

$$\text{id}(x,y) = y$$

Building operations : We will build more complex functions from the initial set by using only three methods that are,

- i) Composition
- ii) Primitive recursion
- iii) Minimization.

I) Composition

We will start with the successor function,

$$s(x) = x + 1$$

Then we may replace its argument, x , with a function. If we replace the argument, x , with the zero function,

$$z(x)$$

then the result is the successor of zero,

$$s(z(x)) = 1$$

$$s(s(z(x))) = 2 \text{ and so on.}$$

In this way, with the help of the initial functions we can describe the natural numbers. This building operations is called "composition". It should be clear that when composition is applied to computable functions, only computable functions will result.

ii) Primitive recursion

The second building operation is called primitive recursion. Primitive recursion is a method of defining new functions from old function. The function h is defined through functions f and g by primitive recursion when

$$h(x, 0) = f(x)$$

$$h(x, s(y)) = g(x, h(x, y))$$

where f and g are known computable functions. There are two equations. When h 's second argument is zero, the first equation applies; when it is not zero, we use the second. Use of successor function in second equation enforces the condition that the argument be greater than zero. Hence, the first equation applies in the minimal case and the second applied in every other case.

To solve the function by primitive recursion

- 1) When the second argument of h is zero, the function is equivalent to some known function f and we compute it;
- 2) Otherwise it is equivalent to some known function g and we compute it.

Thus the function obtained will be computable in nature. For example, we can calculate the factorial function using recursion as :

Initially $1! = 1$ and to calculate $n!$, if we multiply n by $(n - 1)$ then it will generate a nonrecursive series. Instead of that we will multiply n by $(n - 1)!$ We can express the calculation of factorial function by following two equations -

$$1! = 1$$

$$n! = n \cdot (n - 1)!$$

A strict definition of the factorial function, $f(n)$ then, consists of these two equations :

$$f(n) = 1 \quad \text{when } n = 1 \quad \dots(1)$$

$$f(n) = n \cdot f(n-1) \quad \text{when } n > 1 \quad \dots(2)$$

Consider $n = 5$ then using equation (2) we will get,

$$f(5) = 5 \cdot f(4)$$

$$f(4) = 4 \cdot f(3)$$

$$f(3) = 3 \cdot f(2)$$

$$f(2) = 2 \cdot f(1)$$

By putting the value of equation (1) for calculating $f(2)$ we will get,

$$f(2) = 2 \cdot 1 = 2$$

Then

$$f(3) = 3 \cdot f(2)$$

$$= 3 \cdot 2$$

$$= 6$$

Then

$$f(4) = 4 \cdot f(3)$$

$$= 4 \cdot 6$$

$$= 24$$

Then

$$f(5) = 5 \cdot f(4)$$

$$= 5 \cdot 24$$

$$= 120$$

Primitive recursion is like mathematical induction. The first equation defines the basis, and the second defines the induction step.

III) Minimization

If $g(x)$ is a function that computes the least x such that $f(x) = 0$, then we know that g is computable. And then we can say that g is produced from f by minimization. But we can build g by minimization only if $f(x)$ is already known to be computable.

For example : Suppose we want to obtain least x which makes $f(x) = 0$ then we will try the natural numbers 0, 1, 2, until we reach the first value that gives $f(x) = 0$. Now if such search for x never gets terminated then it is called unbounded minimization. And if we obtain such x within some tests then it is called bounded minimization. While unbounded minimization has the disadvantages of a partial function which may never terminate, bounded minimization has the disadvantage of sometimes failing to minimize.

6.2.1 Class of Recursive Functions

The classification of recursive functions is as shown in Fig. 6.2.1.
(See Fig. 6.2.1 on next page).

1. Partial Recursive Function

The function is called partial recursive function if it can be obtained by applying composition, primitive recursion and minimization as building operations.

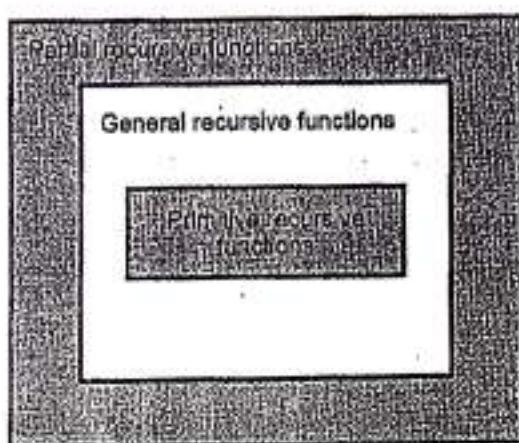


Fig. 6.2.1 Classification of recursive functions

2. General Recursive Function

The function is called general recursive if it is obtained by applying composition, primitive recursive and an unbounded minimization that happen to terminate. The general recursive function is a larger class than partial recursive functions.

3. Primitive Recursive Function

The function is called primitive recursive if it is obtained by applying composition, primitive recursion and unbounded minimization that does not terminate. The set of general recursive function is the same as the set of turing computable functions. The example of general recursive function is an Ackermann's function. The Ackermann's function can be defined as follows -

$$A(0, y) = y + 1$$

$$A(x+1, 0) = A(x, 1)$$

$$A(x+1, y+1) = A(x, A(x+1, y))$$

Then we can calculate $A(x, y)$ for every pair of (x, y) .

→ **Example 6.2.1 :** Compute $A(1, 1)$ $A(1, 2)$, $A(2, 1)$, $A(2, 2)$ using Ackermann's function.

Solution : Let

$$A(0, y) = y + 1 \quad \dots (1)$$

$$A(x+1, 0) = A(x, 1) \quad \dots (2)$$

$$A(x+1, y+1) = A(x, A(x+1, y)) \quad \dots (3)$$

Be the Ackermann's function,

To compute $A(1, 1)$ put $x = 0, y = 0$, then

$$\begin{aligned}
 A(1, 1) &= A(0+1, 0+1) && \dots \text{Using equation (3)} \\
 &= A(0, A(0+1, 0)) \\
 &= A(0, A(0, 1)) && \dots \text{Using equation (2)} \\
 &= A(A(0, 1)) && \dots \text{Using equation (1)} \\
 &= A(0, 2) && \dots \text{Using equation (1)}
 \end{aligned}$$

Hence $A(1, 1) = 3$.

To compute $A(1, 2)$ put $x = 0$ and $y = 1$

$$\begin{aligned}
 A(1, 2) &= A(0+1, 1+1) && \dots \text{Using equation (3)} \\
 &= A(0, A(1, 1)) \\
 &= A(0, 3) && \dots \text{Using equation (1)} \\
 &= 4
 \end{aligned}$$

To compute $A(2, 1)$ put $x = 1$ and $y = 0$

$$\begin{aligned}
 A(2, 1) &= A(1+1, 0+1) \\
 &= A(1, A(2, 0)) \\
 &= A(1, A(1, 1)) \\
 &= A(1, 3) \\
 &= A(0+1, 2+1) \\
 &= A(0, A(1, 2)) \\
 &= A(0, 4) \\
 &= 5
 \end{aligned}$$

To compute $A(2, 2)$ put $x = 1$ and $y = 1$

$$\begin{aligned}
 A(2, 2) &= A(1+1, 1+1) \\
 &= A(1, A(2, 1)) \\
 &= A(1, 5)
 \end{aligned}$$

Now we will compute $A(1, 5)$ where in $x = 0$ and $y = 4$.

$$A(1, 5) = A(0+1, 4+1)$$

$$\begin{aligned}
 &= A(0, A(1, 4)) \\
 &= A(0, (A(0+1, 3+1))) \\
 &= A(0, (A(0, A(1, 3)))) \\
 &= A(0, (A(0, A(0+1, (2+1)))) \\
 &= A(0, (A(0, A(0, A(1, 2)))) \\
 &= A(0, (A(0, A(0, (4)))) \\
 &= A(0, (A(0, 5))) \\
 &= A(0, 6) \\
 A(1, 5) &= 7
 \end{aligned}$$

Hence $A(2, 2) = A(1, 5)$
 $A(2, 2) = 7$

→ **Example 6.2.2 :** Define functions by Primitive Recursion. Show that the function $f(x, y) = x + y$ is primitive recursive.

GTU : Winter-16, Marks 7

Solution : Refer section 6.2.1 (3)

We will write

$$f(x, 0) = x \quad \dots(1)$$

$$f(x, y+1) = f(x, y) + 1 \quad \dots(2)$$

For example consider $x = 2, y = 3$

$$\begin{aligned}
 f(2, 3) &= f(2, 2) + 1 \\
 &= f(2, 1) + 1 + 1 \\
 &= f(2, 0) + 1 + 1 + 1 \\
 &= 2 + 1 + 1 + 1 \quad \therefore f(2, 0) = 2 \text{ from equation 1} \\
 &= 5
 \end{aligned}$$

This shows that

$$f(x, y) = x + y \text{ is primitive recursive}$$

→ **Example 6.2.3 :** Show that the function $f(x, y) = x + y$ is primitive recursive.

GTU : Summer-18, Marks 3

Solution : The primitive recursion can be defined as :

If g and h are primitive recursive and

and $f(x, 0) = h(x)$

$$f(x, y) = g(x, y, f(x, y))$$

Now to prove $f(x, y) = x + y$ as primitive recursive, we define f using the recursion rule and g, h :

$$\begin{aligned} f(x, 0) &= g(x) = \Pi_1(x) = x - \text{Projection rule.} \\ f(x, n+1) &= h(x, n, f(x, n)) \\ &= S(\Pi_3(x, n, f(x, n))) \\ F(x, n+1) &= F(x, n) + 1 \end{aligned}$$

This proves that, $F(x, y) = x + y$ is primitive recursive.

Review Question

1. Write Short note on Following : Primitive Recursive Function. **GTU : Summer-16, Marks 7**

6.3 Recursive Predicate and Bounded Minimization

Any function $f(X)$ can be defined as

$$f(X) = \begin{cases} f_1(X) & \text{when } P_1(X) \text{ is true} \\ f_2(X) & \text{when } P_2(X) \text{ is true} \\ \dots \\ f_k(X) & \text{when } P_k(X) \text{ is true} \end{cases}$$

Where $f_i(X)$ represents some property of the function and $P_i(X)$ represents some condition. In other words any function $f(X)$ can be represented by its property $f_i(X)$ when the condition $P_i(X)$ is true.

The condition $P_i(X)$ has only two states. It can be either true or false. Hence $P_i(X)$ is also called as predicates. This is more precisely called as n -place predicate, because X can be at any place n in the domain N .

The recursive predicate can be defined as

$$f_P(X) = \begin{cases} 1 & \text{if } P(X) \text{ is true} \\ 0 & \text{if } P(X) \text{ is false.} \end{cases}$$

All these predicates can be written by other primitive recursive functions and initial functions, hence they are primitive recursive predicates.

The recursive predicates can make use of logical operators such as AND (\wedge) OR (\vee) and NOT (\neg)

Theorem : If P_1 and P_2 are recursive predicates then there exists $P_1 \wedge P_2$, $P_1 \vee P_2$ and $\neg P_1$

Proof : The given expressions of logical operators can be represented by simple addition, multiplication and subtraction operations. This representation will be

$$P_1 \wedge P_2 = P_1 \cdot P_2$$

$$P_1 \vee P_2 = P_1 + P_2 - P_1 \wedge P_2$$

$$\neg P_1 = 1 - P_1$$

► **Example 6.3.1 :** Give complete primitive recursive derivation for

$$f(n) = n!$$

Solution : The function $f(n)$ can be given as below

$$f(0) = 1$$

$$f(t+1) = g(t, f(t))$$

$$\text{Where } g(x_1, x_2) = S(x_1) \cdot x_2$$

$$\text{For instance } f(0) = 1$$

$$f(1) = g(0, f(0))$$

$$= f(0) \cdot 1$$

$$= 1 \cdot 1$$

$$f(1) = 1.$$

$$f(2) = f(1) \cdot 2$$

$$= 1 \cdot 2$$

$$f(2) = 2$$

$$f(3) = f(2) \cdot 3$$

$$= 2 \cdot 3$$

$$f(3) = 6$$

► **Example 6.3.2 :** If $f(x, y) = x + y$ and $f(5, 0) = 5$ then find $f(5, 3)$.

Solution : $f(x, y)$ is a partial recursive function. We will first find $f(5, 1)$ $f(5, 2)$ and finally $f(5, 3)$. Each functional value can be computed by adding 1 to it.

$$f(5, 1) = f(5, 0) + 1$$

$$= 5 + 1$$

$$= 6$$

$$f(5, 2) = f(5, 1) + 1$$

$$= 6 + 1$$

$$f(5, 2) = 7$$

$$f(5, 3) = f(5, 2) + 1$$

$$= 7 + 1$$

$$f(5, 3) = 8$$

6.3.1 Bounded Quantifications

Bounded quantifications are the quantifications added in the list of standard quantifiers \forall (for every) and \exists (there exists).

Let, P is a predicate then bounded existential quantification of P is denoted by E_P . The E_P is defined as

$$E_P(X, t) = \text{there exists some } y \text{ with } 0 \leq y \leq t \text{ such that } P(X, y) \text{ is true.}$$

The bounded universal quantification of predicate P is denoted by A_P

$$A_P(X, t) = \text{for every } y \text{ with } 0 \leq y \leq t \text{ such that } P(X, y) \text{ is true.}$$

Theorem : If the predicate P is primitive recursive then $A_P(X, y)$ and $E_P(X, y)$ both are primitive recursive then $A_P(X, y)$ and $E_P(X, y)$ both are primitive recursive.

⇒ Example 6.3.3 : Show that $y|x$ is primitive recursive.

Solution : y is divisor of x and it can be represented by bounded quantifications as

$$E_P(y|x) = \text{there exists some } t \text{ such that } y \cdot t = x.$$

Thus using bounded existential quantifier we can show that given function is primitive recursive.

⇒ Example 6.3.4 : Show that Prime(x) is primitive recursive.

Solution : Let x is a prime predicates that prime(x) is primitive recursive.

Prime(x) can be defined using bounded universal quantification as

$$A_P(X, t) = \text{every } y \text{ with } 1 \leq y \leq t \text{ such that } P(y) \text{ is prime.}$$

6.3.2 Bounded Minimization

Minimization is the maximum minimum value which satisfies the function.

The bounded minimization of predicate P is denoted by B_P . The B_P is defined as

$$B_P(X, t) = \begin{cases} \min \{y \mid 0 \leq y \leq t \text{ and } P(X, y) \text{ is true}\} \\ t+1 \quad \text{otherwise} \end{cases}$$

The μ is the operator used for minimization operator. The bounded minimization operator. The bounded minimization is also denoted as $\mu^t y [P(X, y)]$.

For example : If we want to find prime value smaller than 19 then we try 2, 3, 5, 7, 11, 13, 17 then largest minimum value 17 that satisfies the condition (smaller than 19). Hence we select 17 as largest minimum value of bounding function (prime number smaller than 19). This is bounded minimization.

Theorem : If predicated P is primitive recursive then its bounded minimization B_P is also primitive recursive.

6.4 Regular Function

Let $f(x_1, x_2, \dots, x_n, y)$ be a total function over N (natural numbers). The function f is a regular function if there exists some natural number y_0 such that $f(x_1, x_2, \dots, x_n, y_0) = 0$ for all values of x_1, x_2, \dots, x_n in N .

6.5 μ -Recursive Functions

The μ -recursive functions or partial μ -recursive functions are defined as follows -

- 1) Every initial function is a μ -recursive function.
- 2) Every function obtained by composition or by primitive recursion is a μ -recursive function.
- 3) Every total function defined by μ -operator is a μ -recursive function.

$$f(X) = \mu y [f(X, y) = 0]$$

The set of μ -recursive functions is denoted by M.

Theorem : All μ -recursive partial functions are computable.

Proof : In the set of μ -recursive partial function there are initial functions, partial functions obtained by either composition or by primitive recursion. These all type of functions are computable. Hence we can further state that if a function f is a computable partial function then its unbounded minimization M_f is also computable partial function.

To prove this, we can construct a Turing machine T_f which computes M_f (unbounded minimization). Such TM simply computes $f(X, 0), f(X, 1), f(X, 2) \dots$ until $f(X, i) = 0$ is obtained.

If such an i is obtained then the TM accepts and halts. But if no such i is present then the computation of TM continues forever.

This proves that all μ -recursive partial functions are computable.

Review Questions

1. Define the terms partial, total and constant functions

2. Define primitive recursive functions

GTU : Winter 2013, Summer 2014, 3 Marks

3. Explain primitive recursive operations and functions

GTU : Summer 2013, 4 Marks

4. Explain μ -Recursive Functions

GTU : Summer 2013, 3 Marks, Winter-17, 7 Marks

6.6 Short Questions and Answers

Q.1 What is Partial Function ?

Ans. : A partial function f from A to B is a rule which assigns every element of A to at the most one element of B .

Q.2 State the identity function.

Ans. : $f(x) = x$

Q.3 What is primitive recursion ?

Ans. : The primitive recursion is a method of defining new functions from old function. The function h is defined through functions f and g by primitive recursion when,

$$h(x,0) = f(x)$$

$$h(x,s(y)) = g(x, h(x,y))$$

Where f and g are known computable functions

Q.4 State the Ackerman's function

Ans. : The Ackerman's function can be defined as follows :

$$A(0,y) = y+1$$

$$A(x+1,0) = A(x,1)$$

$$A(x+1,y+1) = A(x, A(x+1,y))$$

Q.5 State bounded existential quantification ?

Ans. : Let, P is a predicate then bounded existential quantification of P is denoted by \exists_p . The \exists_p is defined as

$E_p(X, t) = \text{there exists some } y \text{ with } 0 \leq y \leq t \text{ such that } P(X, y) \text{ is true.}$

Q.6 State bounded universal Quantification ?

Ans. : Let, P is a predicate then bounded existential quantification of P is denoted by A_p , the A_p is defined as

$A_p(X, t) \doteq$ for every y with $0 \leq y \leq t$ such that $P(X,y)$ is true.

9.7 What is minimization?

Ans. : Minimalization is the maximum minimum value which satisfies the function. The bounded minimalization of predicate P is denoted by B_P . The B_P is defined as

$$B_P(X, t) = \begin{cases} \min\{y | 0 \leq y \leq t\} & \text{and } P(X, y) \\ t+1 & \text{otherwise} \end{cases}$$

The μ operator is used for minimization operator. The bounded minimization is denoted as, $\mu^b y [P(X, y)]$

Q.8 All μ -recursive functions are computable. (True/False)

Ans. i. True.



Undecidability

7.1 A Language that can't be Accepted and a Problem that can't be Decided

Decidable language : A language is decidable if there is a Turing Machine(TM) which accepts and halts on every input string w . In other words, there exists some algorithm for solving the problem that belongs to decidable language.

Every decidable language is also called as Turing Acceptable Language or recursive languages

For a decidable language, for each input string, the TM halts either at the accept or the reject state.

For example - Is number n is even ?

Solution : Let $S = \{2, 4, 6, 8, 10, \dots\}$ is a set of even numbers. Divide the number ' n ' by 2, if the remainder is zero then the number is accepted as 'Even Number' otherwise it is rejected.

Undecidable language : A language for which there exists no Turing machine which accepts the language is called undecidable language.

A decision problem P is called "undecidable" if the language L of all yes instances to P is not decidable. Undecidable languages are not recursive languages, but sometimes, they may be recursively enumerable languages.

Examples of undecidable languages are -

1. Halting problem
2. Post correspondence problem

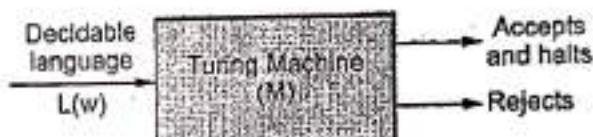


Fig. 7.1.1 Decidability

7.2 Non Recursive Enumerable (RE) Language

A language is called recursively enumerable if and only if the language is accepted by some Turing machine M . In other words L is said to be RE if $L = L(M)$ for some TM M .

There are certain language is not recursively enumerable. Consider a language consisting a pair (M, w) where -

1. M , a Turing machine with input set $\{0, 1\}$.
2. w a binary string consisting of 0's and 1's.
3. M accepts w .

For example :

The universal language L_u is a set of binary strings which can be modeled by a turing machine. The universal language can be represented by a pair (M, w) where M is a TM that accepts this languages and w is a binary string in $(0 + 1)^*$ such that w belongs to $L(M)$.

- The universal turing machine U accepts the TM M .
- The transitions of M are stored initially on first tape along with the string w .
- On the second tape the simulated tape of M is placed. Here the tape symbol X_i of M will be represented by 0ⁱ and tape symbols are separated by single 1's.
- On the third tape various states of machine M are stored. The state q_i is represented by i 0's.

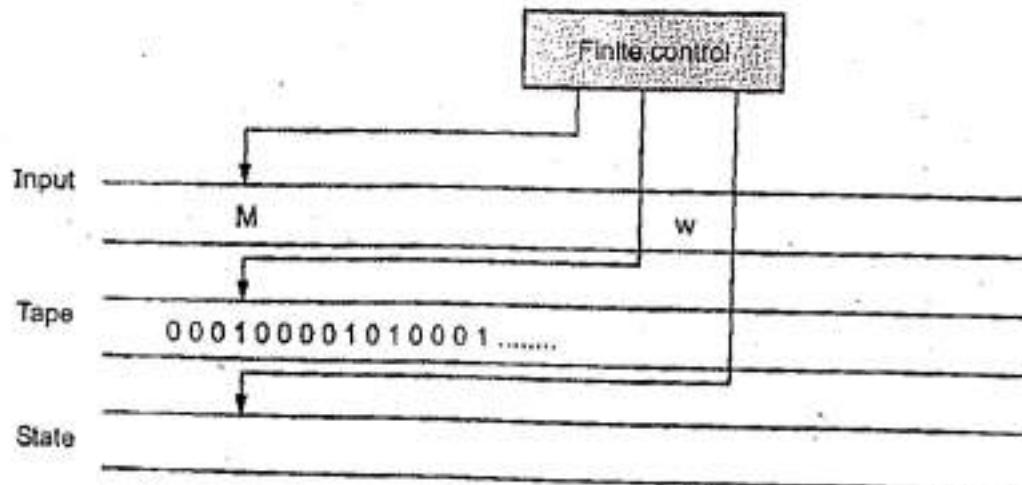


Fig. 7.2.1

7.3 Undecidable Problem with RE

The recursively enumerable (RE) languages are categorized into two classes -

1. The class of languages that has Turing machine. This Turing machine decides whether the input string belongs to that language or not. Such a Turing machine always halts, whether or not it reaches to accept state.
2. The second class of languages consists of those RE languages that are not accepted by any Turing machine with the guarantee of halting.

A language is L actually denoted by $L(M)$ called recursive if it is accepted by some Turing machine such that

1. If string w is in L , turing machine M accepts it and then halts.
2. If w is not in L , then M eventually halts although it never enters in accepting states.

We can also call recursive languages as the **definite languages** or the languages which can be represented by some algorithm and such an algorithm helps in construction of Turing machine for that language.

Decidable and Undecidable Languages -

If a language is recursive then it is called **decidable languages** and if the language is not recursive then such a language is called **undecidable language**.

Hence broadly there are three categories of the languages.

1. Recursive language for which the algorithm exists.
2. Recursively enumerable language for which it is not sure that on which input the TM will ever halt. Such languages are not recursive.
3. The non-recursively enumerable languages for which there is no Turing machine at all.

Fig. 7.3.1 shows the relationship between these languages.

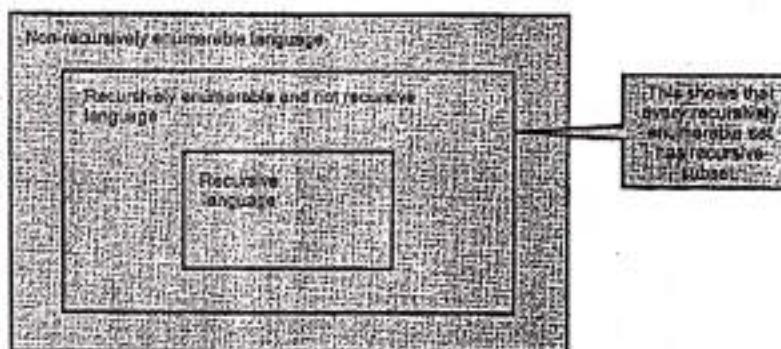


Fig. 7.3.1 Relationship between languages

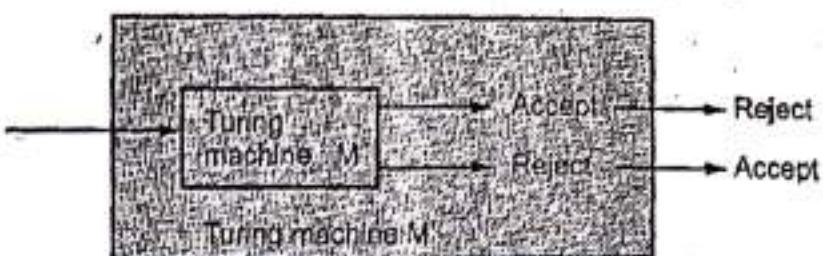
The important fact about recursive and recursively enumerable languages can be seen with the help of following theorem.

Theorem 1 : If L is recursive language then L' is also a recursive language.

Proof : Let there will be some L that can be accepted by turing machine M . Hence we can denote language L by $L(M)$. On acceptance of $L(M)$ the machine M always halts. Now, we construct a TM M' such that $L' = (M')$. for construction of M' following steps are followed -

1. The accepting steps of M are made non-accepting states of M' and there is no transition from M' . That means we have created the states such that M' will halt without accepting.
2. Now create a new accepting state for M' say r and there is no transition from r .

3. In machine M , for each of the transition with combination of nonaccepting state and input tape symbol, make the same transition having the combination of accepting state and input tape symbol for machine M' .



Since M is guaranteed to halt M' is also guaranteed to halt. In fact, M' accepts exactly those strings that M does not accept. Thus we can say that M' accepts L' .

Fig. 7.3.2 Construction of M' accepting L'

Theorem 2 : If a language L and its complement L' both are RE then L is a recursive language.

Proof : Consider a turing machine M made up of two turing machines M_1 and M_2 . The machine M_2 is complement of machine M_1 . We can also denote that $L(M) = L(M_1)$ and $L(M_2)$. Both M_1 and M_2 are simulated in parallel by machine M . Machine M is a two tape TM, which can be made one tape TM for the ease of simulation. This One tape then will consist of tape of machine M_1 and machine M_2 . The states of M consists of all the states of machine M_1 and all the states of machine M_2 . The machine M made up of M_1 and M_2 is as shown in Fig. 7.3.3.

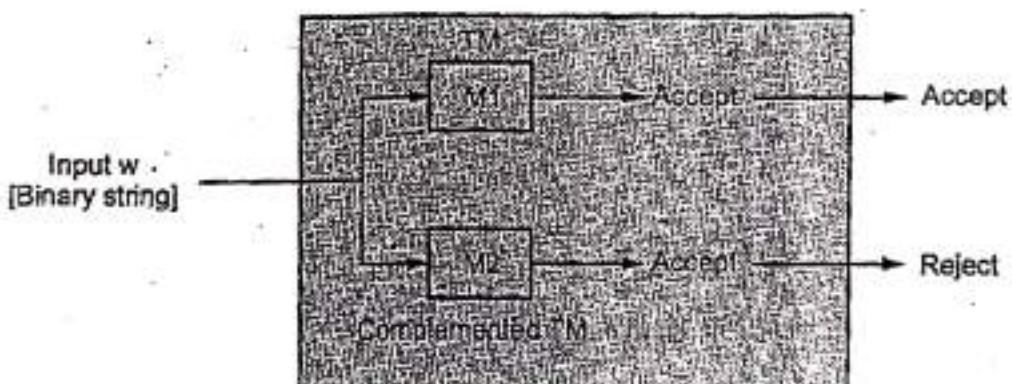


Fig. 7.3.3

If the input w of language L is given to M then M_1 will eventually accept and therefore M will accept L and halt. If w is not in L , then it is in L' . So M_2 will accept and therefore M will halt without accepting. Thus on all inputs, M halts. Thus $L(M)$ is exactly L . Since M always halts we can conclude that $L(M)$ mean \bar{L} is a recursive language. Thus a recursive language can be recursively enumerable but a recursively enumerable language is not necessarily be recursive.

Theorem 3 : Show that if L_1 and L_2 are recursive languages then $L_1 \cup L_2$ and $L_1 \cap L_2$ also recursive.

Proof : Let

L_1 is a recursive language.

L_2 is a recursive language.

As L_1 and L_2 are recursive languages there exists a machine M_1 that accepts L_1 as well as machine M_2 that accepts L_2 . Now, we have to simulate a machine (algorithm) M that accepts the language L such that $L = L_1 \cup L_2$. Then construct machine M which accepts if M_1 accepts. The construction of M is as shown in following Fig. 7.3.4

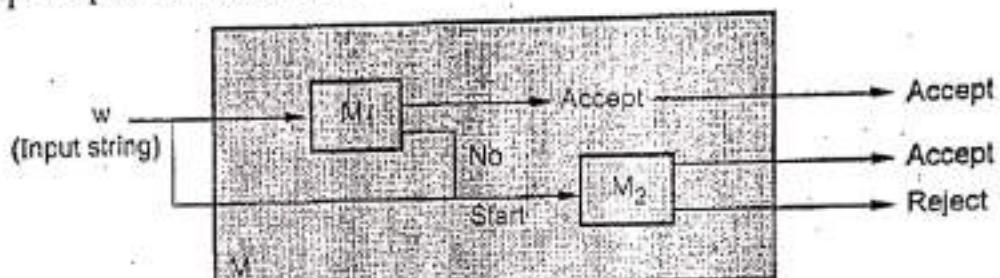
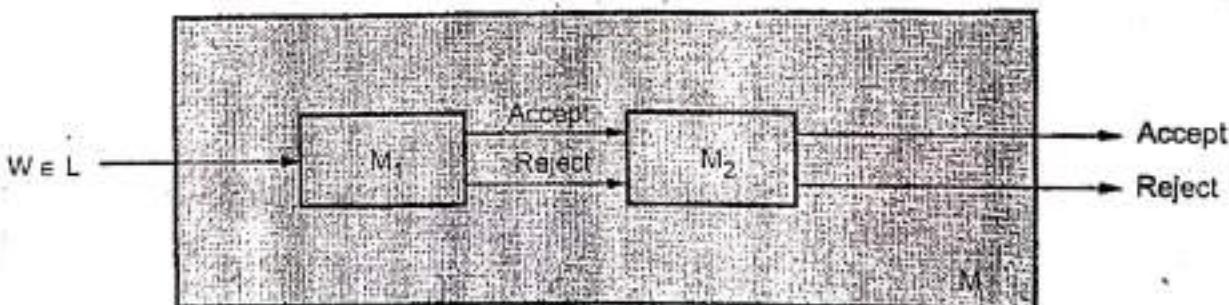


Fig. 7.3.4

If machine M_1 does not accept then M_2 simulates M . That means if M_2 accepts then M accepts, if M_2 rejects M also rejects. Thus M accepts the language $L = L_1 \cup L_2$ which is recursive.

To prove $L_1 \cap L_2$ as a recursive language consider machine M that accepts the language L such that $L = L_1 \cap L_2$.



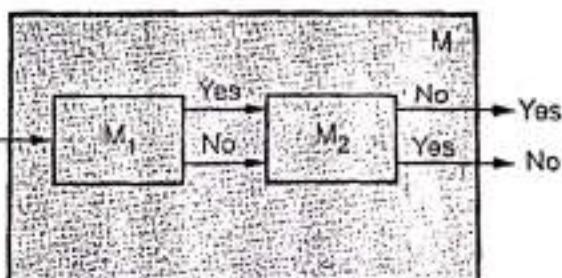
M_1 is a TM which accepts L_1 and M_2 is a TM which accepts L_2 . The TM M is simulated which halts on accepting $w \in L$ such that $L = L_1 \cap L_2$. Hence intersection of two recursive languages is also recursive.

Theorem 4 : If L_1 and L_2 are two recursive languages and if L is defined as : $L = \{w | w \text{ is in } L_1 \text{ and not in } L_2 \text{ and not in } L_1\}$. Prove or disprove that L is recursive.

Proof : Let L_1 is a recursive language which is accepted by M_1 and L_2 is a recursive language which is accepted by M_2 . Languages L_1 and L_2 are such that if $w \in L_1$ then $w \notin L_2$. Similarly if $w \notin L_1$ then $w \in L_2$.

Find $L = L_1 \cup L_2$. We can then design a TM M which accepts the language L . Such a TM can be drawn as follows -

As the language L is accepted by TM M , we can say that the language L is recursive language.



Theorem 5 : Show that the set of languages L over $\{0, 1\}$ so that neither L nor L' is recursively enumerable is uncountable.

Proof : If there is any language L which is recursively enumerable then there exists a TM which semidecides it (either accepts and halts or loops for ever).

Every TM has description of finite length. Hence number of TM and number of recursively enumerable languages is countably infinite, because power set of every countable set is countably infinite. Now, for the language L its complement L' is RE. Then for L' also there exists TM which semidecides it. But there are uncountable number of languages. Hence there may be language L and L' which are not recursively enumerable. And there may be uncountable number of such languages. Hence neither L nor L' is RE in uncountable.

7.4 Undecidable Problems Involving Context-Free Languages

In this section we will discuss some more undecidable problems.

1. It is undecidable whether a function performs some arithmetic operation or print some message or calls some another function.
2. It is undecidable whether a CFG is ambiguous.
3. If G_1 and G_2 are context free grammars and R be a regular expression then following are undecidable -
 - a. Is intersection of two context free languages empty ?
 - b. Is language denoted by G_1 and G_2 both are equal ?
 - c. Is language denoted by G_1 is a subset of the language denoted by G_2 ?
 - d. Does particular string T belongs to language denoted by G_1 ?
 - e. Is language denoted by regular expression R is a subset of the language denoted by G_1 ?

Theorem : Let G be a CFG and r be a regular expression. Show that the problem
 1) $L(G) = L(r)$ 2) $L(r) \in L(G)$ are undecidable

Proof :

1) Let, $L(G)$ is a context free language and $L(r)$ represents a regular language having regular expression. The post's correspondence problem PCP has some instance which becomes a solution for given language $L(r)$ can be accepted by the CFG for $L(G)$. It is also possible that there is no solution instance of PCP. Such an instance is not accepted by CFG. Thus there may exist a CFG for $L(r)$ or there may not be a CFG for $L(r)$. Hence we can state that $L(G) = L(r)$ is undecidable.

2) Let $L(r)$ represents a regular language for some regular expression r . The $L(G)$ denotes the context free language for a CFG G . There may be some strings of language $L(r)$ that are acceptable by G . Then we can state $L(r) \in L(G)$.

But since these are arbitrary strings, there may be some strings of $L(r)$ which are not acceptable by $L(G)$. Hence $L(r) \notin L(G)$.

Thus it is not possible to determine whether $L(r) \in L(G)$. Hence it is undecidable

Theorem : It is undecidable whether $L(G_1) \cap L(G_2) = \emptyset$

Proof : Let, $A = (x_1, x_2, \dots, x_k)$ and $B = (y_1, y_2, y_3 \dots, y_k)$ are instance of PCP.

(Post correspondence problem).

Let G_1 and G_2 are two context free grammars. The production rules for these grammars are as follows -

$$S_1 \rightarrow a_i S_1 x_i \quad \text{for } i = 1, 2, 3, \dots, k$$

$$S_1 \rightarrow \epsilon$$

$$S_2 \rightarrow a_i S_2 y_i \quad \text{for } i = 1, 2, 3, \dots, k$$

$$S_2 \rightarrow \epsilon$$

The grammar G_1 generates the sentence of the form

$$a_{i_1} a_{i_2} \dots a_{i_n} x_{i_1} x_{i_2} \dots x_{i_n}$$

The grammar G_2 generates the sentence of the form

$$a_{i_1} a_{i_2} \dots a_{i_n} y_{i_1} y_{i_2} \dots y_{i_n}$$

If i_1, i_2, \dots, i_n is a solution then the general string is common to G_1 and G_2 . But having solution to PCP is itself undecidable. Hence whether $L(G_1) \cap L(G_2) = \emptyset$ is undecidable.

Theorem : It is undecidable whether a CFG is ambiguous.

Proof : Let $A = (x_1, x_2, \dots, x_k)$ and $B = (y_1, y_2, y_3 \dots, y_k)$ are two strings.

The G_1 and G_2 are two context free grammars. The production rules for these grammars are as follows -

$$S_1 \rightarrow a_i S_1 x_i \quad \text{for } i = 1, 2, 3, \dots, k$$

$$S_1 \rightarrow a_i x_i$$

$$S_2 \rightarrow a_i S_2 y_i \quad \text{for } i = 1, 2, 3, \dots k$$

$$S_2 \rightarrow a_i y_i$$

The grammar G_1 generates the sentence of the form

$$a_{in} a_{in-1} \dots a_{i1} x_{i1} x_{i2} \dots x_{in}$$

The grammar G_2 generates the sentence of the form

$$a_{in} a_{in-1} \dots a_{i1} y_{i1} y_{i2} \dots y_{in}$$

We can combine the languages from both the grammar. The new production which is added here is -

$$S \rightarrow S_1 \mid S_2$$

If grammar G_{12} is ambiguous then PCP with pair (A, B) has a solution.

Similarly if G_{12} is unambiguous then PCP with pair (A, B) has no solution.

But since having solution to PCP is undecidable and there is algorithm for decidability of PCP, there is no algorithm deciding whether given CFG is ambiguous or not. Hence whether CFG is ambiguous or not is undecidable.

7.5 Undecidable Problems about TM

7.5.1 Halting Problem

- 1) Halting Problem can be stated as follows - Given any functional matrix, input data tape and initial configuration, then it is decide whether the process will run forever or will eventually stop."
- 2) The halting problem is unsolvable.
- 3) Proof : Let, there exists a TM M_1 which decides whether or not any computation by a TM T will ever halt when a description d_T of T and tape t of T is given [That means the input to machine M_1 will be (machine, tape) pair]. Then for every input (t, d_T) to M_1 if T halt for input t, M_1 also halts which is called accept halt. Similarly if T does not halt for input t then the M_1 will halt which is called reject halt. This is shown in Fig. 7.5.1.

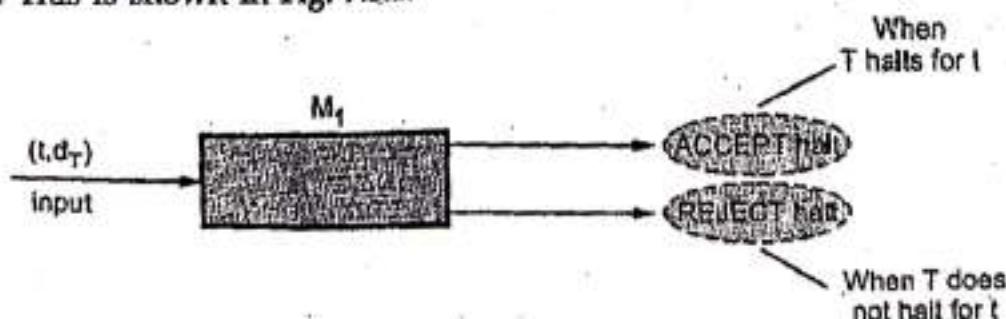


Fig. 7.5.1

Now we will consider another Turing Machine M_2 which takes an input d_T . It first copies d_T and duplicates d_T on its tape and then this duplicated tape information is given as input to machine M_1 . But machine M_1 is a modified machine with the modification that whenever M_1 is supposed to reach an accept halt, M_2 loops forever. Hence behavior of M_2 is as given. It loops if T halts for input $t = d_T$ and halts if T does not halt for $t = d_T$. The T is any arbitrary turing machine.

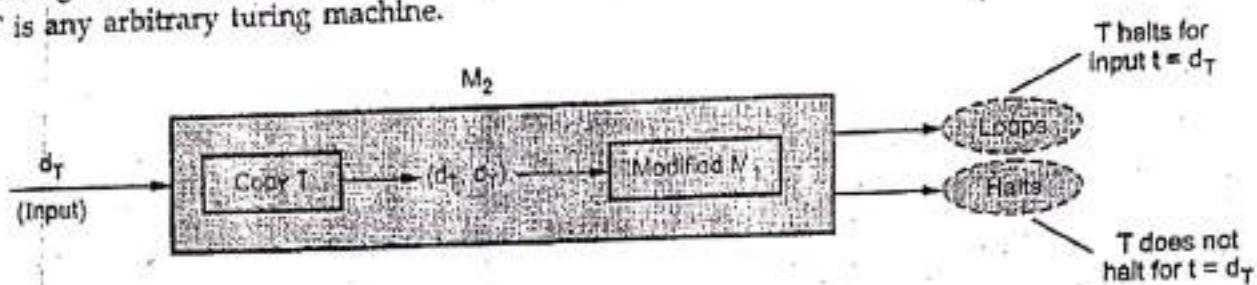


Fig. 7.5.2

As M_2 itself is one turing machine we will take $M_2 = T$. That means we will replace T by M_2 from above given machine.

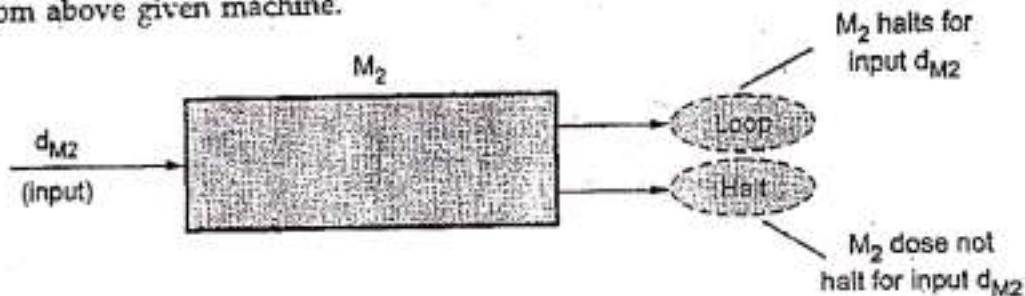


Fig. 7.5.3

Thus machine M_2 halts for input d_{M2} if M_2 does not halt for input d_{M2} . This is a contradiction. That means a machine M_1 which can tell whether any other TM will halt on particular input does not exist. Hence halting problem is unsolvable.

→ Example 7.5.1 : Compare and contrast push down automata and Turing machine.

GTU : Summer-19, Marks 3

Solution :

- 1) Both PDA and TM accept regular as well as context free languages. Both the languages can remember already read input.
- 2) The TM can be programmed and hence it accepts larger class of languages than PDA does.
- 3) We can write an output on the input tape in TM. But in case of simple DPDA it is not possible to produce output.

→ Example 7.5.2 : Enlist limitations of Turing machines. **GTU : Summer-19, Marks 4**

Solution : Following types of problems cannot be solved by TM

- 1) Determining if program will ever halt on given input.

- 2) Determining whether two programs will produce the same output.
- 3) Determining whether program will accept the given input.
- 4) Determining behaviour of other TM.

7.5.2 Post's Correspondence Problem

In this section, we will discuss the undecidability of strings and not of Turing machines. The undecidability of strings is determined with the help of Post's Correspondence Problem (PCP). Let us define the PCP.

The post's correspondence problem consists of two lists of strings that are of equal length over the input Σ . The two lists are $A = w_1, w_2, w_3, \dots, w_n$ and $B = x_1, x_2, x_3, \dots, x_n$, then there exists a non empty set of integers $i_1, i_2, i_3, \dots, i_n$ such that $w_{i_1}w_{i_2}w_{i_3}\dots w_{i_n} = x_{i_1}x_{i_2}x_{i_3}\dots x_{i_n}$.

To solve the post correspondence problem we try all the combinations of $i_1, i_2, i_3, \dots, i_n$ to find the $w_i = x_i$ then we say that PCP has a solution.

Example 7.5.3 : Consider the correspondence system as given below -
 $A = (1; 0; 010; 11)$ and $B = (10; 10; 01; 1)$. The input set is
 $\Sigma = \{0, 1\}$. Find the solution.

Solution : A solution is 1; 2; 1; 3; 3; 4. That means $w_1w_2w_1w_3w_3w_4 = x_1x_2x_1x_3x_3x_4$.
The constructed string from both lists is 10101001011.

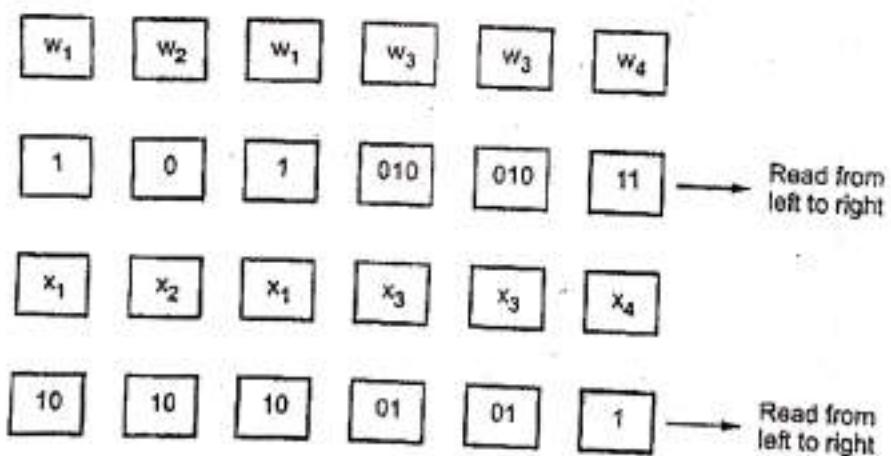


Fig. 7.5.4

Example 7.5.4 : Obtain the solution for the following correspondence system.
 $A = \{ba, ab, a, baa, b\}$, $B = \{bab, baa, ba, a, aba\}$. The input set $\{a, b\}$.

Solution : To obtain the corresponding system the one sequence can be chosen. Hence we get

$w_1w_5w_2w_3w_4w_4w_4w_3w_4 = x_1x_5x_2x_3x_4x_4w_3w_4$. This solution gives a unique string babababaabaaabaa = babababaabaaabaa. Hence solution is 15234434.

→ **Example 7.5.5 :** Obtain the solution for the following system of posts correspondence problem.

$$A = \{100, 0, 1\}, B = \{1, 100, 00\}$$

Solution : The solution is 1 3 1 1 3 2 2. The string is

$$\begin{aligned} A1A3A1A1A1A3A2A2 &= 100 + 1 + 100 + 100 + 1 + 0 + 0 \\ &= 1001100100100 \end{aligned}$$

$$\begin{aligned} B1B3B1B1B3B2B2 &= 1 + 00 + 1 + 1 + 00 + 100 + 100 \\ &= 1001100100100 \end{aligned}$$

→ **Example 7.5.6 :** Obtain the solution for the following system of posts correspondence problem

$$A = \{ba, abb, bab\}, B = \{bab, bb, abb\}$$

Solution : Now to consider 1, 3, 2 the string babababb from set A and bababbbb from set B thus the two strings obtained are not equal. As we can try various combinations from both the sets to find the unique sequence but we could not get such a sequence. Hence there is no solution for this system.

→ **Example 7.5.7 :** Does PCP with two lists $x = (b, bab^3, ba)$ and $y = (b^3, ba, a)$ have a solution?

Solution : Now we have to find out such a sequence that strings formed by x and y are identical. Such a sequence is 2, 1, 1, 3. Hence from x and y list

$$\begin{array}{ccccccccc} 2 & 1 & 1 & 3 & & 2 & 1 & 1 & 3 \\ bab^3 & b & b & ba & = & ba & b^3 & b^3 & a \end{array}$$

which forms bab^3b^3a . Thus PCP has a solution.

Review Questions

1. Write short note on - Halting problem.
2. What is post correspondence problem ? Explain with example.
3. Write short note on - Post correspondence problem.

GTU : Summer-16, 19, Marks 3

7.6 The Class P and NP

There are two groups in which a problem can be classified. The first group consists of the problems that can be solved in polynomial time. For example : searching of an element from the list $O(\log n)$, sorting of elements $O(n \log n)$.

The second group consists of problems that can be solved in non-deterministic polynomial time. For example : Knapsack problem $O(2^{n/2})$ and Travelling Salesperson problem ($O(n^2 2^n)$).

- Any problem for which answer is either yes or no is called decision problem. The algorithm for decision problem is called decision algorithm.
- Any problem that involves the identification of optimal cost (minimum or maximum) is called optimization problem. The algorithm for optimization problem is called optimization algorithm.
- Definition of P - Problems that can be solved in polynomial time. ("P" stands for polynomial). The polynomial time is nothing but the time expressed in terms of polynomial.
Examples - Searching of key element, Sorting of elements, All pair shortest path.
- Definition of NP - It stands for "non-deterministic polynomial time". Note that NP does not stand for "non-polynomial".
- Examples - Travelling Salesperson problem, Graph coloring problem, Knapsack problem, Hamiltonian circuit problems.
- The NP class problems can be further categorized into NP-complete and NP hard problems.

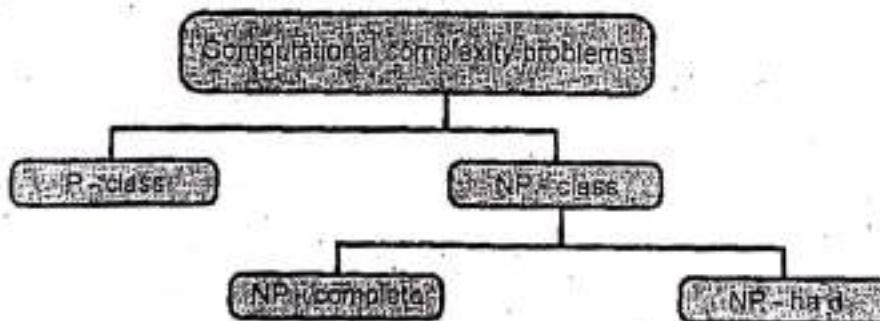


Fig. 7.6.1

- A problem D is called NP-complete if -
 - i) It belongs to class NP
 - ii) Every problem in NP can also be solved in polynomial time.
- If an NP-hard problem can be solved in polynomial time then all NP-complete problems can also be solved in polynomial time.
- All NP-complete problems are NP-hard but all NP-hard problems cannot be NP-complete.

- The NP class problems are the decision problems that can be solved by non-deterministic polynomial algorithms.
- Computational complexity** - The computational problems is an infinite collection of instances with a solution for every instance.

In computational complexity the solution to the problem can be "yes" or "no" type. Such type of problems are called decision problem. The computational problems can be function problem. The function problem is a computational problem where single output is expected for every input. The output of such type of problems is more complex than the decision problem.

The computational problems also consists of a class of problems, whose output can be obtained in polynomial time.

- Complexity classes** - The complexity classes is a set of problems of related complexity. It includes function problems, P classes, NP classes, optimization problem.
- Intractability** - Problems that can be solved by taking a long time for their solutions is known as intractable problems.

If NP is not same as P then NP complete problems are called intractable problems.

Difference between P class and NP class problems

P Class	NP Class Problem
Problems that can be solved in polynomial time are called P-class problems.	Problems that can be solved in non-deterministically Polynomial time are called NP class problem.
These are simple to solve.	These are complex to solve.
Examples : Searching an element from unordered list, sorting the elements.	Examples : Traveling salesperson problem, knapsack problem.

7.6.1 Example of P Class Problem

Kruskal's algorithm : In Kruskal's algorithm the minimum weight is obtained. In this algorithm also the circuit should not be formed. Each time the edge of minimum weight has to be selected, from the graph. It is not necessary in this algorithm to have edges of minimum weights to be adjacent. Let us solve one example by Kruskal's algorithm.

Example :

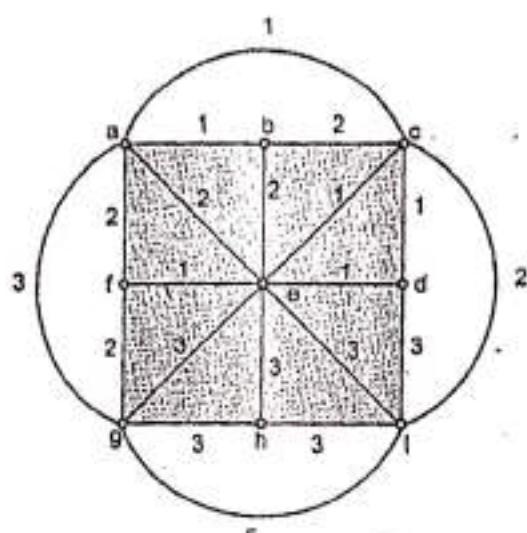


Fig. 7.6.2

Find the minimum spanning tree for the following figure using Kruskal's algorithm.

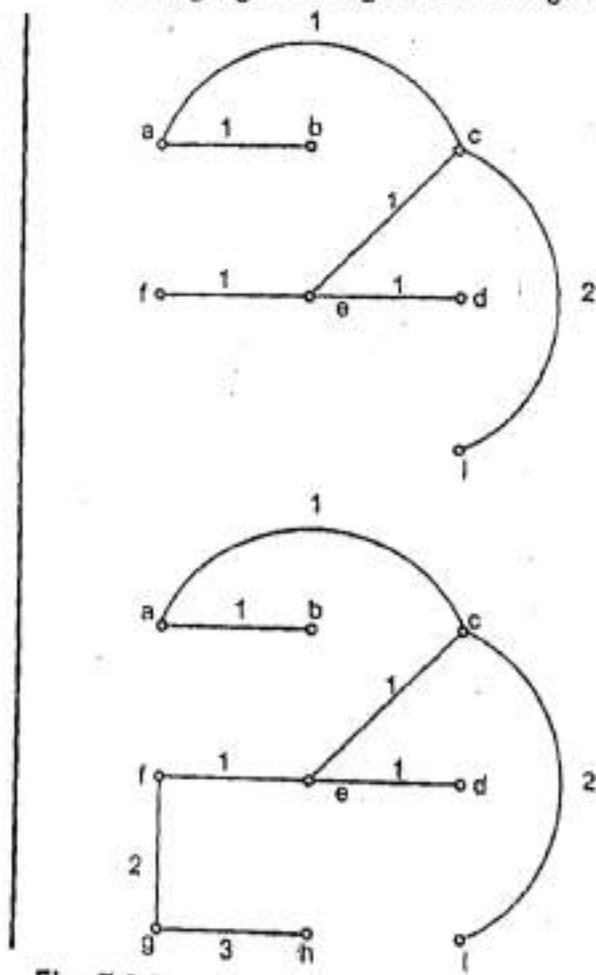
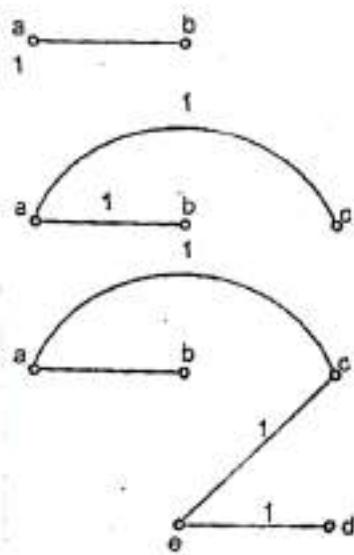


Fig. 7.6.3

In Kruskal's algorithm, we will start with some vertex and will cover all the vertices with minimum weight. The vertices need not be adjacent.

7.6.2 Example of NP Class Problem

Travelling Salesman's Problem (TSP) : This problem can be stated as " Given a set of cities and cost to travel between each pair of cities, determine whether there is a path that visits every city once and returns to the first city. Such that the cost travelled is less".

For example :

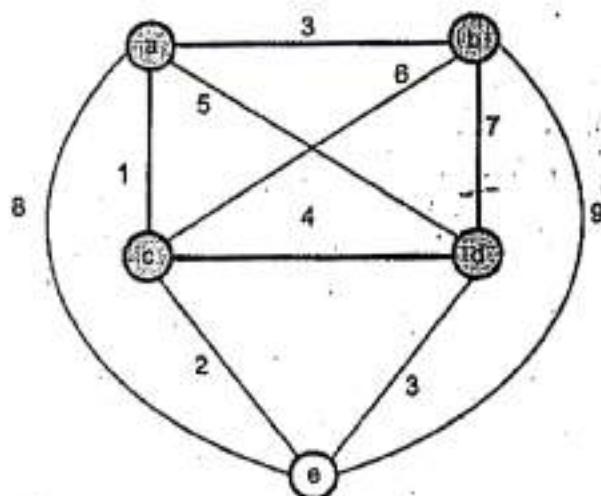


Fig. 7.6.4

The tour path will be a-b-d-e-c-a and total cost of tour will be 16.

This problem is NP problem as there may exist some path with shortest distance between the cities. If you get the solution by applying certain algorithm then Travelling Salesman problem is NPComplete Problem. If we get no solution at all by applying an algorithm then the travelling salesman problem belongs to NP hard class.

Review Questions

1. Explain with example - computational complexity.
2. Differentiate between P class and NP class problems.



SOLVED MODEL QUESTION PAPER

(As Per New Syllabus)

Theory of Computation

Semester - VI (CE / CSE)

Time : $2\frac{1}{2}$ Hours]

[Total Marks : 70]

Instructions

- 1) Attempt all questions.
 - 2) Make suitable assumptions wherever necessary.
 - 3) Figures to the right indicate full marks.
- Q.1** a) Define the term : ϵ -closure. (Refer section 2.8) [4]
b) Find a regular expression corresponding to each of the following subsets of $\{0, 1\}^*$.
1. The language of all strings that begin or end with 00 or 11.
2. The language of all strings containing both 11 and 010 as substrings.
(Refer example 2.1.27) [3]
c) Define one-to-one, onto and bijection function.
Check whether the function $f : R+ \rightarrow R$, $f(x) = x^2$ is one to one and onto.
(Refer example 1.3.4) [7]
- Q.2** a) Explain the concept of NFA with Epsilonn. (Refer section 2.8) [3]
b) Write regular expression over the alphabets {a, b} consisting strings :
• Second last character as 'a'
• Starting with 'a' and ending with 'b'. (Refer example 2.1.29) [4]
c) Write a DFA to accept the language $L = \{L : |W| \text{ mod } 5 \neq 0\}$.
(Refer example 2.6.11) [7]

OR

- c) Convert the given NFA into its equivalent DFA -

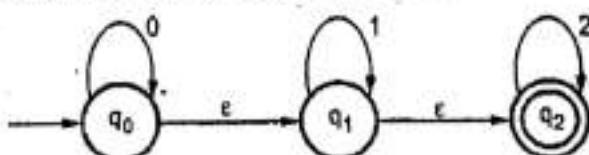


Fig. 1

(Refer example 2.10.4) [7]

- Q.3** a) Give the left linear grammar for $RE(10)^*1$. (Refer example 3.3.3)
(M - 1) [3]

- b) Define CFG. When is a CFG called an ambiguous grammar ?
 (Refer section 3.6) [4]

- c) What is CNF ? Convert the following CFG into CNF.
 $S \rightarrow ASA \mid aB$
 $A \rightarrow B \mid S,$
 $A \rightarrow b \mid \epsilon$ (Refer example 3.10.10) [7]

OR

- Q.3 a) Explain the concept of leftmost and rightmost derivation with example.
 (Refer section 3.4) [3]

- b) Consider following grammar :
 $S \rightarrow ASB \mid \lambda$
 $A \rightarrow aAS \mid a$
 $B \rightarrow Sbs \mid A \mid bb$
 i) Eliminate useless symbols, if any.
 ii) Eliminate λ productions. (Refer example 3.9.4) [4]

- c) Generate the context free grammar that give the following languages.
 i) {W | W contains at least three 1's}
 ii) {W | W starts and ends with same symbol} (Refer example 3.2.22) [7]

- Q.4 a) State and enlist the closure properties of context free languages.
 (Refer section 4.5) [3]

- b) What is Turing machine ? Write advantages of TM over FSM.
 (Refer example 5.1.1) [4]

- c) Design and draw a deterministic PDA accepting "Balanced strings of Brackets" which are accepted by following CFG.

$S \rightarrow SS \mid [S] \mid \{S\} \mid \lambda$ (Refer example 4.2.6) [7]

OR

- Q.4 a) Construct an NPDA that accept the language generated by the grammar
 $S \rightarrow aSbb \mid ab$. (Refer example 4.3.3) [3]

- b) Show that $L = \{a^n b^n c^n | n \geq 0\}$ is not a context free language.
 (Refer example 4.4.1) [4]

- c) Construct a TM for $L = \{a^n b^n c^n | n \geq 1\}$. (Refer example 5.4.5) [7]

- Q.5 a) Explain the P and NP class problems with example. (Refer section 7.6) [3]

- b) Show that the function $f(x, y) = x + y$ is primitive recursive.
 (Refer example 6.2.3) [4]

- c) Construct TM for reversing a binary string on the input tape.
(Refer example 5.6.4)

[7]

OR

- Q.5 a) Write short note on - Halting Problem. (Refer section 7.5) [4]
- b) Explain μ - recursive functions. (Refer section 6.5) [3]
- c) Obtain the solution for the following system of posts correspondence problem.
 $A = \{100, 0, 1\}$, $B = \{1, 100, 00\}$. (Refer example 7.5.5) [7]

□□□

LALJI

Mo. 9925644326
7990243510

CHAMUNDA XEROX

G-46,Offira Business Hub, Near Bhagwan Mahavir college, Bharthana Vesu, Surat.

**2 Rs 5 Copy
Back To Back**

**Color Xerox & Color Print
B & W Xerox & Print (Urgent)
Project Binding, Spiral Binding
Lamination**

NAME : _____

COLLEGE : _____

SUBJECT : _____

ROLL NO. : _____