SUBJECT CODE : 310242

As per Revised Syllabus of

SAVITRIBAI PHULE PUNE UNIVERSITY

Choice Based Credit System (CBCS)

T.E. (Computer Engineering) Semester - V

THEORY OF COMPUTATION

Mrs. Anuradha A. Puntambekar

M.E. (Computer)

Formerly Assistant Professor in

P.E.S. Modern College of Engineering, Pune

Dr. Santoshkumar V. Chobe

Ph.D. (Computer Engineering)

M.E.(CSE), B.E.(CSE),

Professor

Dr. D. Y. Patil Institute of Technology, Pimpri, Pune

Dr. Jyoti Rao

Ph. D. (Computer Engineering)

Professor in Computer Engineering Department

Dr. D.Y. Patil Institute of Technology,

Pimpri, Pune.

TECHNICAL PUBLICATIONS

SINCE 1993

An Up-Thrust for Knowledge

THEORY OF COMPUTATION

Subject Code : 310242

T.E. (Computer Engineering) Semester - V

© Copyright with A. A. Puntambekar

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 :

TECHNICAL PUBLICATIONS
SINCE 1993
An Up-Thrust for Knowledge

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

ISBN 978-93-91567-03-3

9 789391 567033

SPPU 19

PREFACE

The importance of **Theory of Computation** is well known in various engineering fields. Overwhelming response to our books on various subjects inspired us 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.

We wish to express our 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 our whole family. We 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.

Authors
Mrs. Anuradha A. Puntambekar
Dr. Santoshkumar V. Chobe
Dr. Jyoti Rao

Dedicated to God.

SYLLABUS

Theory of Computation - (310242)

Credit :	Examination Scheme :
03	Mid-Sem (TH) : 30 Marks
	End-Sem (TH) : 70 Marks

Unit I Formal Language Theory and Finite Automata

Finite Automata (FA) : An informal picture of FA, Finite State Machine (FSM), Language accepted by FA, Definition of Regular Language.

FA without output : Deterministic and Nondeterministic FA (DFA and NFA), epsilon - NFA and inter-conversion. Minimization of DFAs.

FA with output : Moore and Mealy machines -Definition, models, inter-conversion.

(Chapter - 1)

Unit II Regular Expressions (RE)

Introduction, Operators of RE, Precedence of operators, Algebraic laws for RE, Language to Regular Expressions, Equivalence of two REs. Conversions : RE to NFA, DFA, DFA to RE using Arden's theorem, Pumping Lemma for Regular languages, Closure and Decision properties of Regular languages. Myhill-Nerode theorem. **(Chapter - 2)**

Unit III Context Free Grammar (CFG) and Context Free Language (CFL)

Basic Elements of Grammar, Formal Definition of Context Free Grammar, Sentential form, Derivation and Derivation Tree / Parse Tree, Context Free Language (CFL), Ambiguous Grammar, writing grammar for language. **Simplification of CFG :** Eliminating ε - productions, unit productions, useless production, and useless symbols. **Normal Forms :** Chomsky Normal Form, Greibach Normal Form, Pumping Lemma for CFG, Closure properties of CFL, Decision properties of CFL, Chomsky Hierarchy, Cock-Younger-Kasami Algorithm. **(Chapter - 3)**

Unit IV Pushdown Automata (PDA)

Introduction, Formal definition of PDA, Equivalence of Acceptance by Final State and Empty stack, Non-deterministic PDA (NPDA), PDA and Context Free Language, Equivalence of PDA and CFG, PDA vs CFLs. Deterministic CFLs. **(Chapter - 4)**

Unit V Turing Machines (TM)

Turing Machine Model, Formal definition of Turing Machines, Language Acceptability by Turing Machines, Design of TM, Description of TM, Techniques for TM Construction, Computing function with Turing Machine, Variants of Turing Machines, Halting Problem of TM, Halting vs Looping, A Turing-unrecognizable language, Reducibility, Recursion Theorem. The Model of Linear Bounded Automata. **(Chapter - 5)**

Unit VI Computability and Complexity Theory

Computability Theory : Decidable Problems and Un-decidable Problems, Church-Turing Thesis.

Reducibility : Undecidable Problems that is recursively enumerable, A Simple Un-decidable problem.

Complexity Classes : Time and Space Measures, The Class P, Examples of problems in P, The Class NP, Examples of problems in NP, P Problem Versus NP Problem, NP-completeness and NP-hard Problems. **(Chapter - 6)**

TABLE OF CONTENTS

Unit - I

Chapter - 1 Formal Language Theory and Finite Automata **(1 - 1) to (1 - 102)**

Part I : Finite Automata (FA)

1.1 Introduction to Formal Language .....1 - 3

1.2 Alphabets and Languages .....1 - 3

1.2.1 Defining Languages . . . . . 1 - 3

1.2.2 Symbols / Alphabets . . . . . 1 - 3

1.2.3 String or Word . . . . . 1 - 4

1.3 An informal Picture of FA.....1 - 4

1.4 Representation .....1 - 5

1.5 Finite State Machine (FSM).....1 - 7

1.6 Language Accepted by FA .....1 - 7

1.7 Definition of Regular Language.....1 - 8

Part II : FA without Output

1.8 Types of Finite Automata.....1 - 9

1.9 Deterministic Finite Automata.....1 - 9

1.10 Non Deterministic Finite Automata .....1 - 32

1.11 Epsilon- NFA .....1 - 34

1.11.1 ε - Closure . . . . . 1 - 36

1.12 NFA with ε to NFA without ε .....1 - 37

1.13 NFA to DFA Conversion.....1 - 45

1.14 NFA with Epsilon to DFA Conversion .....1 - 57

1.15 Minimization of DFAs.....1 - 66

Part III : FA with Output

1.16 Moore and Mealy Machines .....1 - 69

1.17 Inter-Conversion .....1 - 83

1.18 Case Study.....1 - 98

1.19 Fill in the Blanks with Answers .....1 - 100

1.20 Multiple Choice Questions with Answers .....1 - 100

Unit - II

Chapter - 2 Regular Expressions (2 - 1) to (2 - 74)

2.1 Operators of RE.....2 - 2

2.2 Language to Regular Expressions.....2 - 3

2.3 Precedence of Operators.....2 - 13

2.4 Algebraic Laws for RE.....2 - 13

2.5 Equivalence of Two Regular Expressions.....2 - 14

2.6 RE to DFA .....2 - 23

2.7 DFA to RE using Arden's Theorem .....2 - 39

2.7.1 State / Loop Elimination . . . . . 2 - 40

2.7.2 Arden's Theorem . . . . . 2 - 44

2.8 Pumping Lemma for Regular Languages .....2 - 52

2.8.1 Applications of Pumping Lemma . . . . . 2 - 53

2.9 Closure Properties of Regular Languages .....2 - 63

2.10 Decision Properties of Regular Languages.....2 - 67

2.11 Limitation of FA.....2 - 68

2.12 Myhill - Nerode Theorem .....2 - 68

2.13 Case Study.....2 - 70

2.14 Fill in the Blanks with Answers .....2 - 72

2.15 Multiple Choice Questions with Answers.....2 - 72

Unit - III

Chapter - 3 Context Free Grammar (CFG) and Context Free Language (CFL) (3 - 1) to (3 - 100)

3.1 Basic Elements of Grammar ..... 3 - 2

3.2 Formal Definition of Context Free Grammar ..... 3 - 2

3.3 Sentential Form ..... 3 - 3

3.4 Context Free Language (CFL) ..... 3 - 4

3.5 Derivation ..... 3 - 23

3.6 Parse Tree ..... 3 - 26

3.7 Ambiguous Grammar ..... 3 - 28

3.7.1 Inherent Ambiguity ..... 3 - 31

3.7.2 Removal of Ambiguity ..... 3 - 32

3.8 Simplification of CFG ..... 3 - 39

3.8.1 Removal of Useless Symbols ..... 3 - 39

3.8.2 Elimination of ε Productions from Grammar ..... 3 - 44

3.8.3 Removing Unit Productions ..... 3 - 48

3.9 Normal Forms ..... 3 - 54

3.9.1 Chomsky Normal Form ..... 3 - 54

3.9.2 Greibach Normal Form ..... 3 - 64

3.10 Pumping Lemma for CFG ..... 3 - 71

3.11 Closure Properties of CFL ..... 3 - 74

3.12 Decision Properties of CFL ..... 3 - 79

3.13 Chomsky Hierarchy ..... 3 - 83

3.14 Cock-Younger-Kasami Algorithm ..... 3 - 84

3.15 Applications of CFG ..... 3 - 91

3.16 Case Studies ..... 3 - 95

3.17 Fill in the Blanks with Answers ..... 3 - 97

3.18 Multiple Choice Questions with Answers ..... 3 - 97

Unit - IV

Chapter - 4 Pushdown Automata (PDA) (4 - 1) to (4 - 36)

4.1 Formal Definition of PDA ..... 4 - 2

4.2 Instantaneous Description ..... 4 - 3

4.3 Acceptance by Final State ..... 4 - 18

4.4 Acceptance by Empty Stack ..... 4 - 19

4.5 Equivalence of Acceptance by Final State and Empty Stack ..... 4 - 19

4.6 Non-deterministic PDA (NPDA) ..... 4 - 21

4.7 PDA and Context Free Language ..... 4 - 22

4.7.1 PDA to CFG ..... 4 - 22

4.7.2 CFG to PDA ..... 4 - 29

4.8 Deterministic CFLs ..... 4 - 32

4.9 Fill in the Blanks with Answers ..... 4 - 34

4.10 Multiple Choice Questions with Answers ..... 4 - 34

Unit - V

Chapter - 5 Turing Machines (TM) (5 - 1) to (5 - 68)

5.1 Turing Machine Model ..... 5 - 2

5.2 Design of TM ..... 5 - 2

5.3 Instantaneous Description(ID) of TM ..... 5 - 2

5.4 Techniques for TM Construction ..... 5 - 3

5.4.1 Storage in Finite Control ..... 5 - 3

5.4.2 Multiple Tracks ..... 5 - 4

5.4.3 Checking Off Symbols ..... 5 - 4

5.4.4 Subroutine ..... 5 - 6

5.5 Language Acceptability by Turing Machines.....5 - 8

5.6 Computing Function with Turing Machine .....5 - 29

5.7 Variants of Turing Machines.....5 - 50

5.7.1 Deterministic Turing Machine (DTM) . . . . . 5 - 50

5.7.2 Non Deterministic Turing Machine (NTM) . . . . . 5 - 51

5.7.3 Universal Turing Machine . . . . . 5 - 51

5.7.4 Multi-tape Turing Machine . . . . . 5 - 52

5.8 Halting Problem of TM.....5 - 52

5.9 Halting Vs Looping .....5 - 54

5.10 A Turing - unrecognizable Language.....5 - 55

5.11 Reducibility .....5 - 56

5.12 Recursion Theorem.....5 - 57

5.12.1 Class of Recursive Functions . . . . . 5 - 59

5.13 The Model of Linear Bounded Automata .....5 - 62

5.14 Fill in the Blanks with Answers .....5 - 67

5.15 Multiple Choice Questions with Answers.....5 - 67

Unit - VI

Chapter - 6 Computability and Complexity Theory (6 - 1) to (6 - 32)

Part I : Computability Theory

6.1 Decidable Problems and Un-decidable Problems.....6 - 2

6.1.1 Decidability Problems Concerning Regular Languages . . . . . 6 - 2

6.1.2 Decidability Problems Concerning Context Free Languages . . . . . 6 - 3

6.2 Church-Turing Thesis .....6 - 5

Part II : Reducibility

6.3 Undecidable Problems that is Recursively Enumerable .....6 - 5

6.4 A Simple Un-decidable Problem .....6 - 11

6.4.1 Post Correspondence Problem . . . . . 6 - 11

Part III : Complexity Classes

6.5 Time and Space Measures .....6 - 12

6.5.1 Order of Growth . . . . . 6 - 14

6.6 The Class P and NP.....6 - 16

6.6.1 Difference between P Class and NP Class Problems . . . . . 6 - 17

6.7 Examples of Problems in P.....6 - 18

6.8 Examples of Problems in NP .....6 - 19

6.9 NP-completeness and NP-hard Problems.....6 - 20

6.10 Polynomial Time Reduction .....6 - 22

6.10.1 Proving NP Problems . . . . . 6 - 23

6.11 Satisfiability Problems .....6 - 26

6.11.1 The Satisfiability Problems. . . . . 6 - 26

6.11.2 Representing SAT Instances. . . . . 6 - 26

6.11.3 NP Completeness of SAT Problem. . . . . 6 - 26

6.12 Restricted Satisfiability Problem.....6 - 30

6.12.1 Normal Forms for Boolean Expression . . . . . 6 - 30

6.12.2 Converting Expressions to CNF . . . . . 6 - 30

Solved Model Question Papers **(M - 1) to (M - 4)**

Notes

UNIT - I

1 Formal Language Theory and Finite Automata

Syllabus

Finite Automata (FA) : *An informal picture of FA, Finite State Machine (FSM), Language accepted by FA, Definition of Regular Language.*
FA without output : *Deterministic and Nondeterministic FA (DFA and NFA), epsilon- NFA and inter-conversion. Minimization of DFAs.*
FA with output : *Moore and Mealy machines - Definition, models, inter-conversion.*

Contents

Part I : Finite Automata (FA)

1.1 Introduction to Formal Language
1.2 Alphabets and Languages . . . . . **Dec.-10,19,** · · · · · · · · · · · · Marks 4
1.3 An informal Picture of FA
1.4 Representation
1.5 Finite State Machine (FSM)
1.6 Language Accepted by FA
1.7 Definition of Regular Language . . . . . **Dec.-18,** · · · · · · · · · · · · · · Marks 3

Part II : FA without Output

1.8 Types of Finite Automata
1.9 Deterministic Finite Automata. **May-10,12,13,18,19,**
. . . . . **Dec.-14,16,17,18,**
. . . . . **Aug.-15,17, Oct.-16,** · · · · Marks 10
1.10 Non Deterministic Finite Automata **Dec.-12,13,14, May-12,14,**
. . . . . **Aug.-15,17,** · · · · · · · · · · · · Marks 6
1.11 Epsilon- NFA **Dec.-13,18,19, Aug.-17,**
. . . . . **May-11,** · · · · · · · · · · · · · · Marks 5

1.12 NFA with ε to NFA without ε **Dec.-14,15, Oct.-16,**
. . . . . **May-19,** · · · · · · · · · · · · · · · Marks 6
1.13 NFA to DFA Conversion **Dec.-05,09,13,17,**
. . . . . **May-09,10,12,**
. . . . . **Aug.-15,17,** · · · · · · · · · · · Marks 10
1.14 NFA with Epsilon to DFA Conversion . . . . . **Dec.-11,** · · · · · · · · · · · · · · Marks 6
1.15 Minimization of DFAs . . . . . **Dec.-12,** · · · · · · · · · · · · · · Marks 8

Part III : FA with Output

1.16 Moore and Mealy Machines **May-07,10,11,13,14,15,**
. . . . . **Dec.-06,11,13,14,19,**
. . . . . **Oct.-16,** · · · · · · · · · · · · · Marks 10
1.17 Inter - Conversion **Dec.-05,11,12,18,**
. . . . . **May-06,07,12,14,19,**
. . . . . **Aug.-15,17, Oct.-19**, · · · · · Marks 8
1.18 Case Study
1.19 Fill in the Blanks
1.20 Multiple Choice Questions

Part I : Finite Automata (FA)

1.1 Introduction to Formal Language

- Formal language is normally defined by using alphabets and formation rules.
- The **alphabet** of formal language is a set of symbols using which the language is built.
- The **formation rules** specify which strings of symbols are used in well-formed manner. The well-formed strings of symbols are also called **words**. The formation rules can be recursive.
- In theory of computations the formal languages are denoted by **regular expressions,regular languages, non regular languages, context free grammar** and so on.

1.2 Alphabets and Languages

SPPU : Dec.-10,19, Marks 4

1.2.1 Defining Languages

The language is a **collection of appropriate strings.**

The language is defined using an input set. The input set is typically denoted by letter $\sum$.

For example : $\sum = \{\varepsilon, 0, 00, 000, ...\}$

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

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

1.2.2 Symbols / Alphabets

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

The example of alphabet is,

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

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

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

Here 0 and 1 are alphabets, denoted by W.

Symbols : Members of an alphabet are called 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'.

1.2.3 String or Word

String is finite collection of symbols from alphabet.

For example, if $\sum$ = {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.

Review Questions

1. *Define the following terms with example - 1) Symbol 2) Alphabet.*

   SPPU : Dec.-10, Marks 4

2. *Define the following terms with example : i) Alphabet ii) String iii) Language.*

   SPPU : Dec.-19, End Sem, Marks 6

1.3 An informal Picture of FA

Finite automata is useful to solve real-world problems. To simulate any real-world problem, the events are captured by the states of finite automata. The interactions among the participants is then represented by transitions.

For example : Suppose a customer wants to purchase some books on-line. Then there are three participants involved in this scenario. Those are customer, book-store and bank. The events that would take place in such a system are -

1. The customer may decide to **pay.**

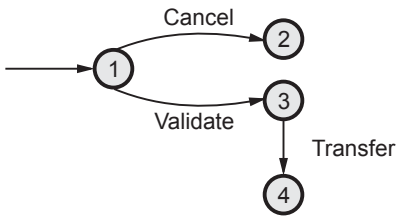
   The customer purchases the required book and sends the money to the store, by credit-card.

2. The customer may decide to **cancel** the purchase order. The money is sent to the bank with a message that the value of the money is to be added to the customer's bank account.

3. The book-store may **ship** books to the customer.

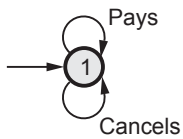
4. The store may get is **validated.** That means the credit card information will be validated by the bank. And it checked whether the customer's bank balance is greater than the purchase amount.

Then the finite automata for **bank** can be drawn as follows -



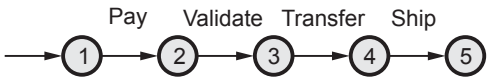
If customer cancels the purchase order then the money will not be transferred to the store. If customer purchases the book, then bank validates his credit card and transfers the money to the book-store.

The finite automata for **customer** can be



Customer performs only two actions either pays the money or cancel the order.

The finite automata for **book-store** can be -



The customer pays the money by credit card, the book store gets it validated from bank then the required amount of money gets transferred from customer's account to book-store's account. When the book-store gets the purchase amount then the books will be shipped to the customer.

1.4 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 -

A transition diagram or transition graph can be defined as collection of -

1) Finite set of states K.

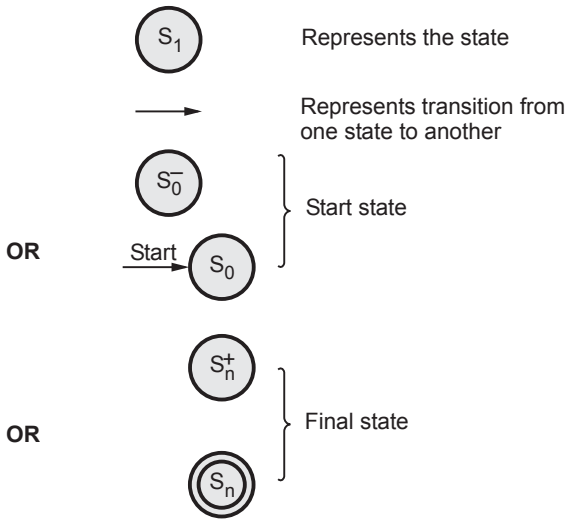
2) Finite set of symbols Σ.

3) A non empty set S of K. It is called **start state**.

4) A set $F \subseteq K$ of **final states**.

5) A transition function $K \times A \rightarrow K$ with K as state and A as input from Σ^*.

The notations used in transition diagram are -



For example :

The FA can be represented using transition graph. The machine initially is in start state S_0 then on receiving input 0 it changes to state S_1. From S_0 on receiving input 1 the machine changes its state to S_4. The state S_2 is a final state or accept state. When we trace the input for transition diagram and reach to a final state at end of input string then it is said that the given input is **accepted by** transition diagram.

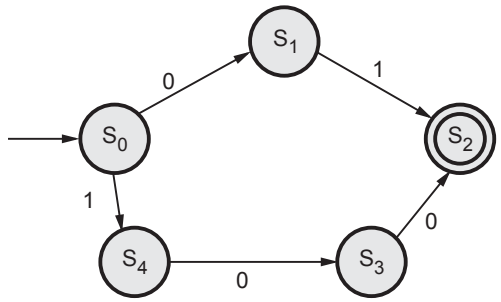
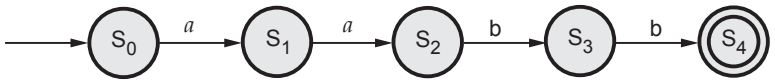


Fig. 1.4.1 Transition diagram

Another example :



We have drawn a transition diagram for the input aabb.

Note that the start state is S_0 and final state is S_4. The input set is $\Sigma = \{a, b\}$. The states S_1, S_2, S_3 are all intermediate states.

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
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.

1.5 Finite State Machine (FSM)

A finite state machine 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.

1.6 Language Accepted by FA

The finite automata can be represented as :

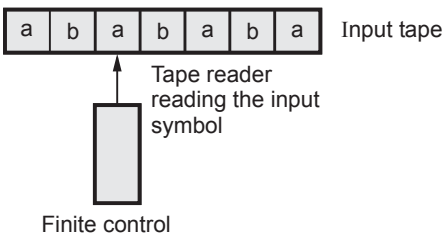


Fig. 1.6.1 Model for finite automata

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 $(w, s) \vdash^{*} (q, \varepsilon)$

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,

$$L = L(M).$$

1.7 Definition of Regular Language

SPPU : Dec.-18, Marks 3

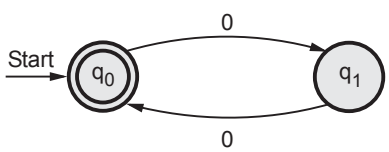
Regular languages are the sets which are accepted by finite automata.

For example : L = {ε, 00, 0000, 000000, ...}

i.e. the set of even number of zeros.

We can represent this set by a finite automata as shown in figure.

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



The given set L is said to be a regular languages because it is represented by finite automata.

Review Question

1. *Define the following terms with example : i) Alphabet ii) String iii) Regular language.*

SPPU : Dec.-18, In Sem, Marks 3

Part II : FA without Output

1.8 Types of Finite Automata

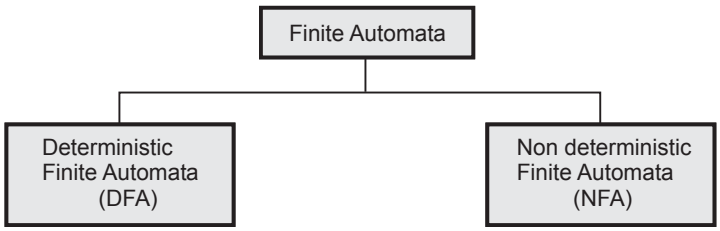


Fig. 1.8.1 DFA and NFA

There are two types of finite automata -

i) Deterministic Finite Automata.

ii) Non deterministic Finite Automata.

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.

1.9 Deterministic Finite Automata

SPPU : May-10,12,13,18,19, Dec.-14,16,17,18, Aug.-15,17, Oct.-16, Marks 10

Definition of DFA

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 $F \in Q$.
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.

Example

For example, $q_0 = \delta(q_0, a)$ means from current state q_0, with input a the next state transition is q_2.

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

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

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

It can be represented by following Fig. 1.9.1.

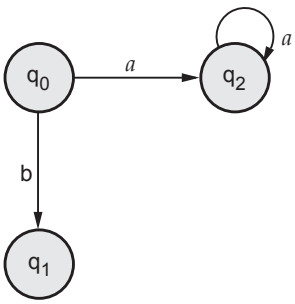


Fig. 1.9.1 Deterministic finite automata

Problems Based on DFA

Example 1.9.1 *Design a FA which checks whether the given binary number is even.*

Solution :

- **Logic Used**

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 a even number, it is equal to 4.

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

- **Design**

While designing FA we will assume one start state, one state ending in 0 and other state for 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.

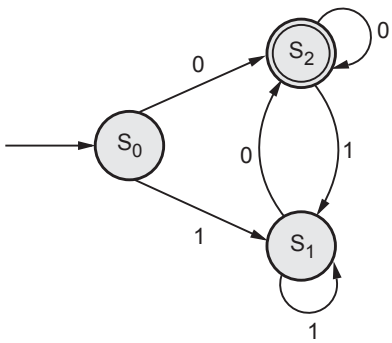


Fig. 1.9.2 DFA

- **Simulation**

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.

$$01000 \Rightarrow 0\underline{S_2}1000$$

$$01\underline{S_1}000$$

$$010\underline{S_2}00$$

$$0100\underline{S_2}0$$

$$01000S_2$$

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

$$1011 \Rightarrow S_0 \vdash 1011$$

$$1S_1 \vdash 011$$

$$10S_2 \vdash 11$$

$$101S_1 \vdash 1$$

$$1011S_1 \vdash$$

Now we are at the state S_1 which is a non final state. Here the symbol "$\vdash$" denotes the symbols to be scanned. On LHS of $\vdash$ the already read symbols are given and at the R.H.S. of "$\vdash$" the symbols to be scanned are given.

- **Representation of DFA using Transition Table**

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

States \ Input	0	1
$\rightarrow S_0$	S_2	S_1
S_1	S_2	S_1
S_2	S_2	S_1

Table 1.9.1 Transition table

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

2) 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**. Sometimes $*$ is also used to indicate final state.

Example 1.9.2 *Design FA which accepts odd number of 1's and any number of 0's.*

Solution :

- **Logic**

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.

- **Design**

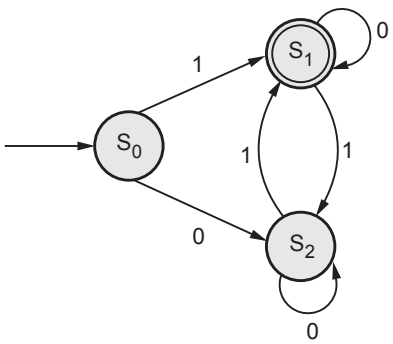


Fig. 1.9.3

Example 1.9.3 *Design FA which checks whether the given unary number is divisible by 3.*

Solution :

- **Logic**

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.

- **Design**

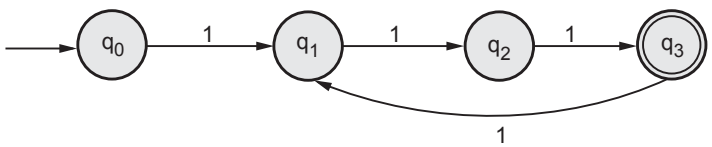


Fig. 1.9.4

Input \ State	1
$\rightarrow q_0$	q_1
q_1	q_2
q_2	q_3
q_3	q_1

- **Simulation**

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

$1\underline{q_1}11111$

$11\underline{q_2}1111$

$111q_3111$

$1111\underline{q_1}11$

$11111\underline{q_2}1$

$111111q_3$ $\rightarrow$ Now we are in final state.

Example 1.9.4 *Design FA to check whether given decimal number is divisible by three.*

Solution :

- **Logic**

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)

- **Design**

We will 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.

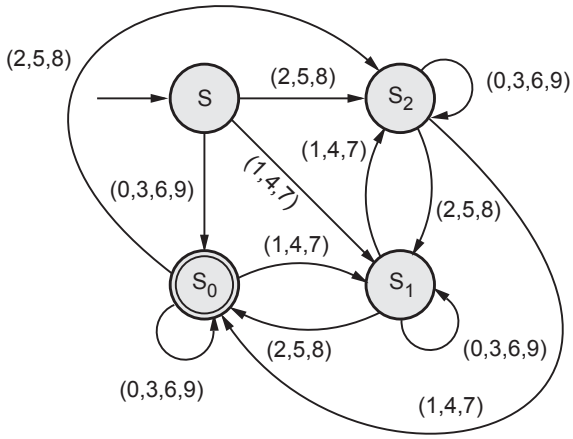


Fig. 1.9.5

- **Simulation**

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

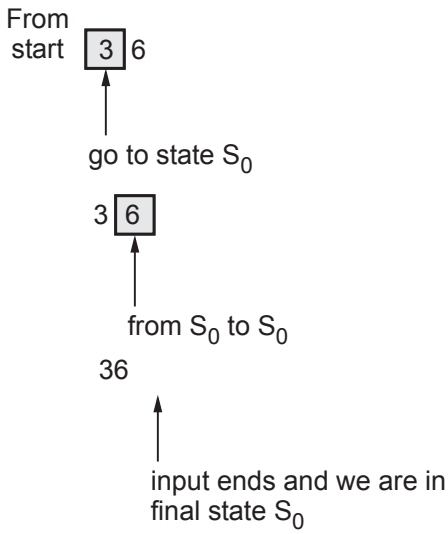


Fig. 1.9.6

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 $\underline{S_1}$ 21

12 $\underline{S_0}$ 1

121 S_1 which is remainder 1 state.

Example 1.9.5 *Design FA which checks whether a given binary number is divisible by three.*

SPPU : May-18, End Sem, Marks 6

Solution :

- **Design**

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 S_0, remainder 1 by S_1 and remainder 2 by S_2.

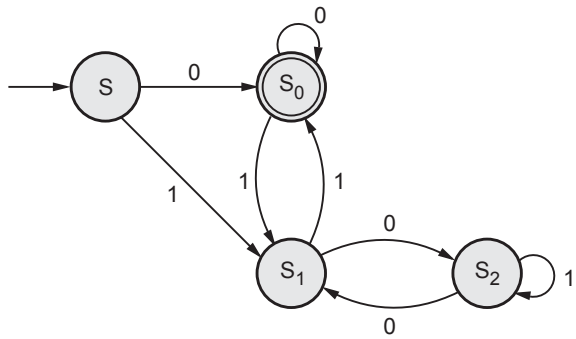


Fig. 1.9.7

- **Simulation**

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

$$\begin{array}{ccc} 0 & 1 & 0 \\ \uparrow & \uparrow & \uparrow \\ S_0 & S_1 & S_2 \end{array}$$

Then we will reach to state S_2 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.

$$\begin{array}{cccc} 1 & 0 & 0 & 1 \\ \uparrow & \uparrow & \uparrow & \uparrow \\ S_1 & S_2 & S_1 & \boxed{S_0} \end{array}$$

Thus the number is really divisible by 3.

Example 1.9.6 *Design FA which accepts even number of 0's and even number of 1's.*

Solution :

- **Logic**

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.

- **Design**

Let us try to design the machine :

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 :

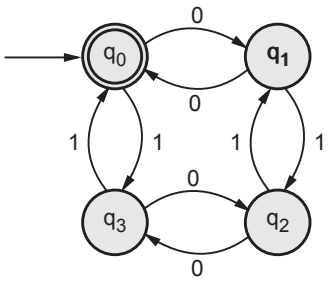


Fig. 1.9.8

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.

The transition table can be as follows -

	0	1
→ q_0	q_1	q_3
q_1	q_0	q_2
q_2	q_3	q_1
q_3	q_2	q_0

Example 1.9.7 *Design FA to accept the string that always ends with 00.*

Solution :

If the input is 01001100 it will be processed as :

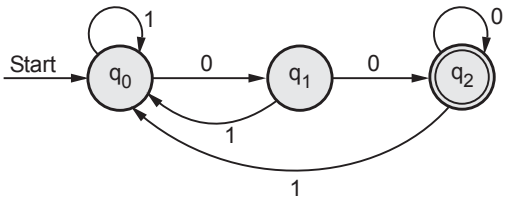


Fig. 1.9.9

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

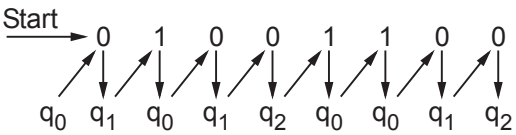


Fig. 1.9.10

Example 1.9.8 *Construct the transition graph for a FA which accepts a language L over $\Sigma\{0, 1\}$ in which every string start with 0 and ends with 1.*

Solution :

In the above transition graph we have handled the case as if the input starts with 1 then it will be in q_3 state which is a dead state and never lead to final state. Thus this machine strictly handles the strings starting with 0 and ending with 1.

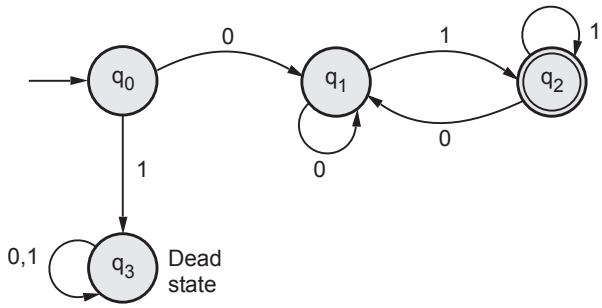


Fig. 1.9.11

Example 1.9.9 *Design FA to accept L, where L = {Strings in which a always appears trippled} over the set Σ = {a, b}.* **SPPU : Aug.-17, In Sem, Marks 3**

Solution :

- **Logic**

For this particular language the valid strings are *aaa*b, b*aaaaaa*, bb*aaa*b and so on. The *a* always appears in a clump of 3. The TG (transition graph) will look like this -

- **Design**

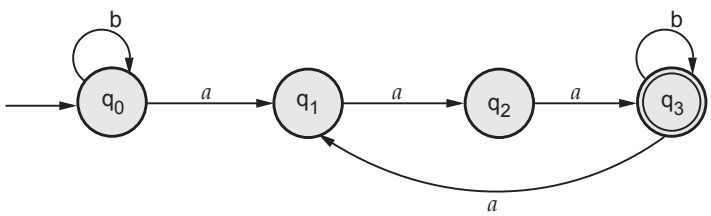


Fig. 1.9.12

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

Example 1.9.10 *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 :

- **Logic**

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.

- **Design**

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

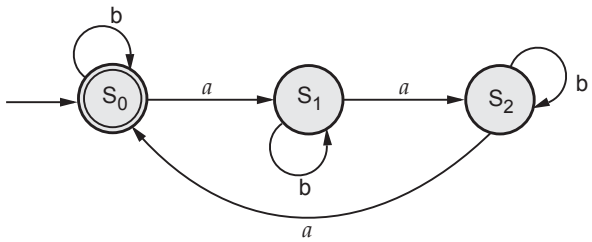


Fig. 1.9.13

Example 1.9.11 *Design a FA that reads strings made up of letters in the word. CHARIOT and recognize those strings that contain the word 'CAT' as a substring.*

Solution :

- **Logic**

To design this FA we will first consider the input set which will be Σ = {C, H, A, R, I, O, T}. From this input set we have to recognize the word 'CAT'. We can do that as follows -

Step 1 :

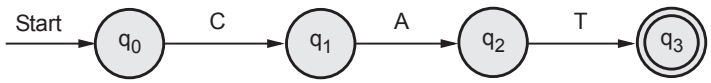


Fig. 1.9.14

Step 2 :

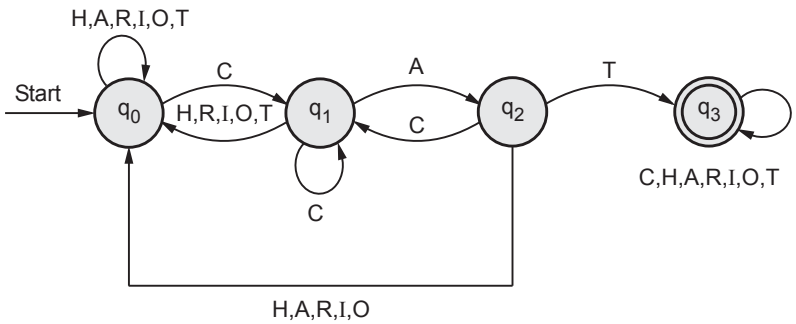


Fig. 1.9.15 Required finite automata

From Fig. 1.9.15 it is clear that on recognizing C initially from Σ = {C, H, A, R, I, O, T} we are moving to state q_1, other than C all the characters are remaining in state q_0 only. From q_1 the character A will be identified and from q_2 character T will be identifier. The transition table can be drawn as follows :

- **Design**

State \ Input	C	H	A	R	I	O	T
$\rightarrow q_0$	q_1	q_0	q_0	q_0	q_0	q_0	q_0
q_1	q_1	q_0	q_2	q_0	q_0	q_0	q_0
q_2	q_1	q_0	q_0	q_0	q_0	q_0	q_3
q_3	q_3	q_3	q_3	q_3	q_3	q_3	q_3

The substring may appear anywhere in the input string for any number of times. For example, 'HACCATHA'. From this 'CAT' can be recognized and we should reach to q_3 state finally.

Example 1.9.12 *Design DFA to accept odd and even numbers represented using binary notation.*

Solution :

- **Logic**

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 -

- **Design**

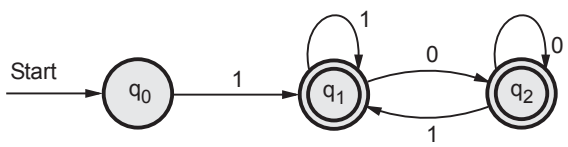


Fig. 1.9.16

Example 1.9.13 *Write a DFA to accept the language L = {L : |W| mod 5 ≠ 0}.*

Solution :

- **Logic**

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

- **Design**

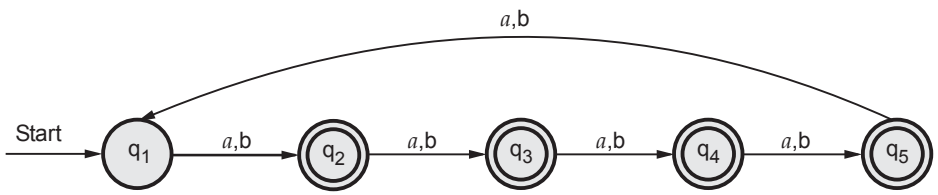


Fig. 1.9.17

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 "*a*bbb". This string is a valid string as it is not divisible by 5 i.e *a*bbb mod 5 ≠ 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 :

- **Simulation**

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

where q_5 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.

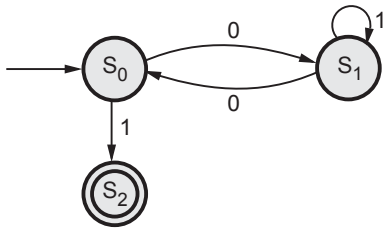
As q_1 is a start we will start from state q_1.

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

Here q_1 is a non-final state. That means "*ababa*" is invalid string for the given DFA.

Example 1.9.14 *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

We now will reach to final state S_2 and the input ends here. Hence 000101 is a valid input string. The transition table for above drawn DFA can be represented as

States \ Input	0	1
$\rightarrow S_0$	S_1	S_2
S_1	S_0	S_1
S_2	-	-

Table 1.9.2 Transition table

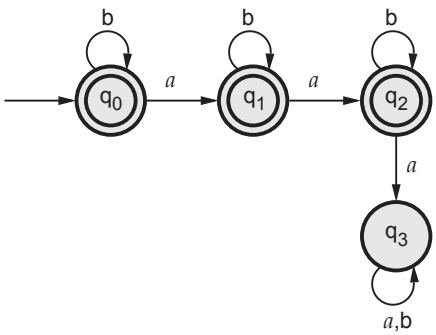
Example 1.9.15 *Design DFA which accepts all the strings not having more than two a's over* $\Sigma = \{a, b\}$.

Solution :

- **Logic**

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.

- **Design**



The transition table can be as shown -

Input / States	*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

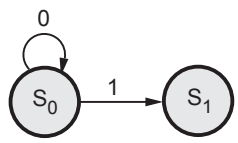
Table 1.9.3 Transition table

Example 1.9.16 *Design a DFA for accepting all the strings of* $\{L = 0^m 1^n \mid m \geq 0 \text{ and } n \geq 1\}$.

Solution :

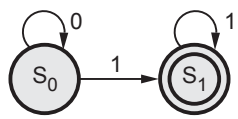
- **Logic**

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 -



- **Design**

We have 1^n in the language. Hence the DFA can be



The transition table can be

State \ Input	0	1
S_0	S_0	S_1
S_1	–	S_1

- **Simulation**

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

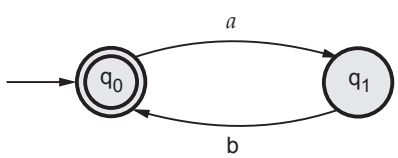
$$\begin{aligned}\delta(S_0, 0\,0\,1\,1\,1) &\vdash (S_0, 0111)\\ &\vdash (S_0, 111)\\ &\vdash (S_1, 11)\\ &\vdash (S_1, 1)\\ &\vdash (S_1, \varepsilon)\end{aligned}$$

Thus we reach to final state S_1 on acceptance of 00111.

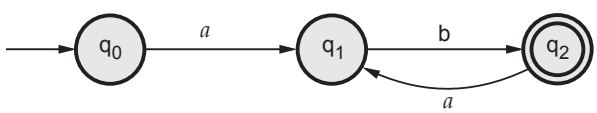
Example 1.9.17 *Design DFA over* $\sum = \{a, b\}$ *for i)* $(ab)^n$ *with* $n \geq 0$ *ii)* $(ab)^n$ *with* $n \geq 1$.

Solution : This is a language of ab in a pair in which i) Condition ε is accepted and in ii) Condition ε is not accepted.

The DFA for i) can be



ii) The DFA will be -



The string accepted by

i) $\delta(q_0, ab) \vdash (q_1, b)$

$\vdash (q_0, \varepsilon)$

Here q_0 is a final state.

ii) $\delta(q_0, ab) \vdash (q_1, b)$
$\vdash (q_2, \varepsilon)$

Here q_2 is final state.

Example 1.9.18 *Design DFA for accepting the set of integers.*

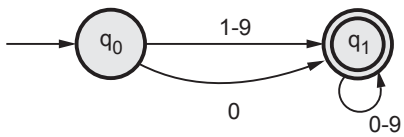
Solution :

- **Logic**

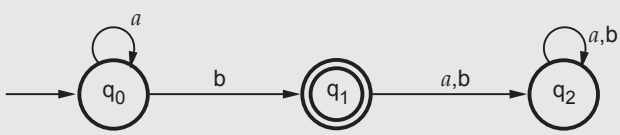
For representing any integer the set {0, 1, ... 9} is used. We can represent any integer value by taking appropriate combinations of symbols from {0, ... 9}.

- **Design**

DFA will be



Example 1.9.19 *Recognize the language given by following DFA.*



Solution : In the given DFA, at q_0 state a self loop for symbol *a* is given. That means any number of *a*'s are allowed. Then if single 'b' comes then it leads to final state. But after this b if anything i.e. '*a*' or 'b' comes then we reach to a non final state q_2 and in this state whatever may come, we will be in q_2 state only. Basically state q_2 is called a dead state. Thus it is a language in which any number of *a*'s should be followed by single b only. Hence this language is denoted as

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

Example 1.9.20 *Design DFA for a language of string 0 and 1 that*

I) Ending with 10 *II) Ending with 11*

III) Ending with 1. **SPPU : May-10, Marks 10**

Solution :

I) DFA which ends with 10

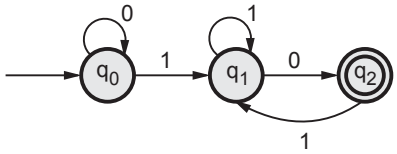


Fig. 1.9.18

II) DFA which ends with 11

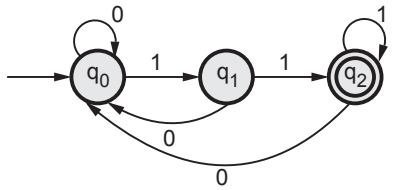


Fig. 1.9.19

III) DFA which ends with 1

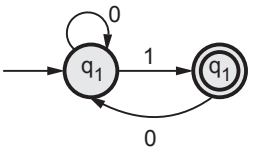


Fig. 1.9.20

Example 1.9.21 *Construct DFA for checking "whether a string over alphabet {a, b} contains a substring abb".* **SPPU : May-12, Marks 8**

Solution : The DFA will be -

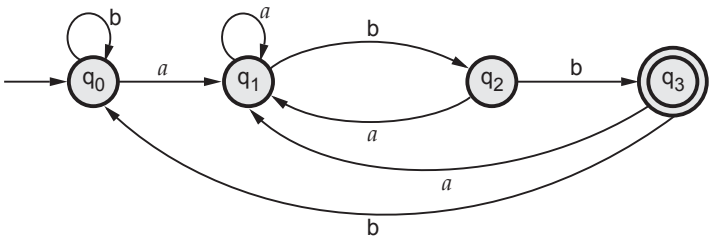


Fig. 1.9.21

Example 1.9.22 *Design a finite automata that reads strings made up of letters in the word 'CHARIOT' and recognize those strings that contain the word 'CAT' as a substring.* **SPPU : May-13, Marks 8**

Solution :

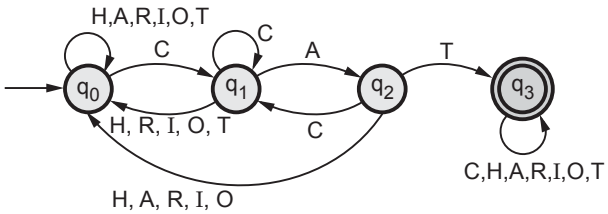


Fig. 1.9.22

State \ Input	C	H	A	R	I	O	T
→ q_0	q_1	q_0	q_0	q_0	q_0	q_0	q_0
q_1	q_1	q_0	q_2	q_0	q_0	q_0	q_0
q_2	q_1	q_0	q_0	q_0	q_0	q_0	q_3
q_3	q_3	q_3	q_3	q_3	q_3	q_3	q_3

$M = (\{q_0, q_1, q_2, q_3\}, \{C, H, A, R, I, O, T\}, \delta, q_0, \{q_3\})$

Example 1.9.23 *Design FSM over {a, b} accepting the string where number of a's are divisible by 2 and number of b's are divisible by 3.* **SPPU : Dec.-14, End Sem, Marks 6**

Solution :

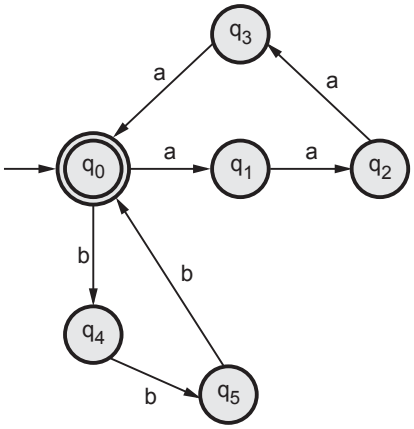
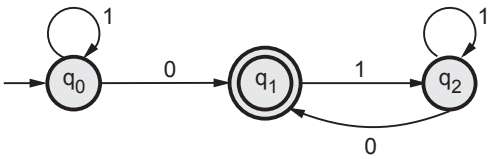


Fig. 1.9.23

Example 1.9.24 *Draw FA for strings which do not contain '00' as substring in it over alphabet* $\sum = \{0,1\}^*$ **SPPU : Aug.-15, In Sem, Marks 2**

Solution :



Example 1.9.25 *Construct a DFA over the alphabets {0, 1} for accepting the strings having number of 1's as multiple of 3.* **SPPU : Dec.-16, End Sem, Marks 6**

Solution :

- **Logic**

  Here the condition is on number of 1's and not on number of 0's.

- **Design**

  Hence DFA is as follows :

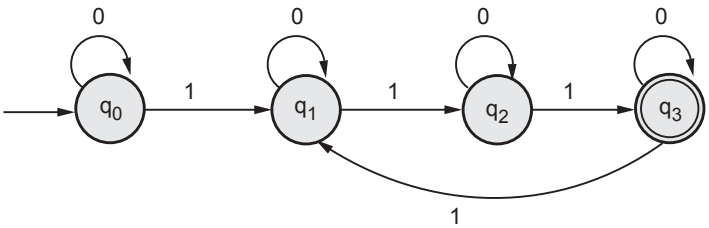


Fig. 1.9.24

- **Simulation :**

Consider string 0100110

$$q_0 \vdash 0100110$$
$$0q_0 \vdash 100110$$
$$01q_1 \vdash 00110$$
$$010q_1 \vdash 0110$$
$$0100q_0 \vdash 110$$
$$01001q_2 \vdash 10$$
$$010011q_3 \vdash 0$$
$$0100110q_3$$

ACCEPT

Example 1.9.26 *Design Finite Automata (FA) for accepting strings, over Σ = {0, 1}, with even number of 0's and odd number of 1's.* **SPPU : Oct.-16, In Sem, Marks 4**

Solution : The finite automata will be as follows -

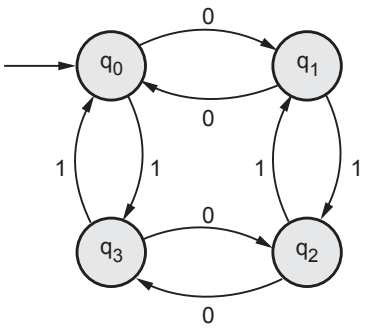


Fig. 1.9.25

Example 1.9.27 *Construct the DFA for the language of all strings that begin and end with same symbol over the alphabet Σ = {0, 1}.* **SPPU : Aug.-17, In Sem, Marks 4**

Solution :

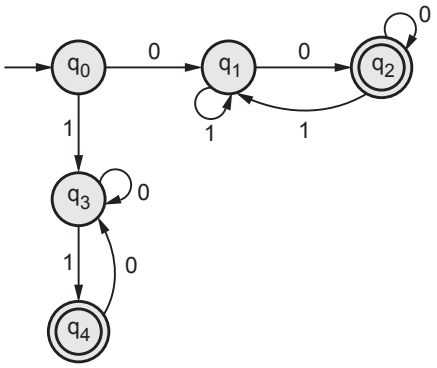


Fig. 1.9.26

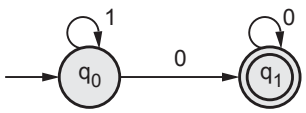
Example 1.9.28 *Construct DFA for language defined by Σ = {0, 1} where*

S = {Strings ending with 0 always}

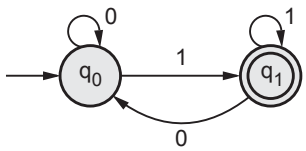
S = {strings representing odd binary numbers}

S = {strings over Σ with total number of 0's even}* **SPPU : Dec.-17, End Sem, Marks 6**

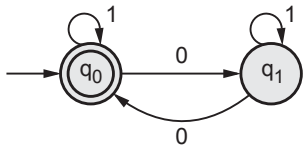
Solution : i)



ii)



iii)



Example 1.9.29 *Design a FA which checks the divisibility by 4 for a decimal number.*

SPPU : Dec.-17, End Sem, Marks 6

Solution : For checking divisibility by 4 for a decimal number we have to consider remainder 0, 1, 2 and 3 groups from input set Σ = {0, 1, 2, 3, 4, 5, 6, 7, 8}. We will name these groups as q_0, q_1, q_2 and q_3 states.

q_0 = {0, 4, 8}

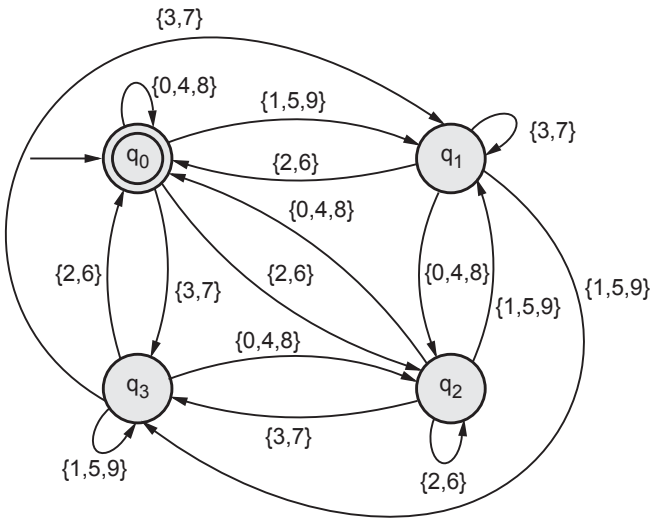
q_1 = {1, 5, 9}

q_2 = {2, 6}

q_3 = {3, 7}

Here q_0 is a start state as well as final state. The transition table as well as transition diagram is as given below.

State \ i/p	{0, 4, 8}	{1, 5, 9}	{2, 6}	{3, 7}
q_0	q_0	q_1	q_2	q_3
q_1	q_2	q_3	q_0	q_1
q_2	q_0	q_1	q_2	q_3
q_3	q_2	q_3	q_0	q_1



Example 1.9.30 *Explain the application of regular expressions in text search and replace.*

SPPU : May-18, End Sem, Marks 6

Solution : 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] +

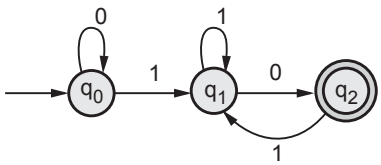
Thus use of regular expressions for identifying patterns from given text makes the job of compiler more simplified.

In web applications also the pattern matching is done using regular expressions.

Example 1.9.31 *Construct a DFA to accept strings ending with 10.*

SPPU : Oct.-18, In Sem, Marks 4

Solution :



Example 1.9.32 *Design a DFA which accepts a ternary number divisible by 4.*

SPPU : Dec.-18, End Sem, Marks 6

Solution : The ternary number is denoted by $\sum = \{0, 1, 2\}$.

For divisibility by 4 there are four cases -

Remainder 0, Remainder 1, Remainder 2 and Remainder 3.

Now we will consider separate states for each of these cases.

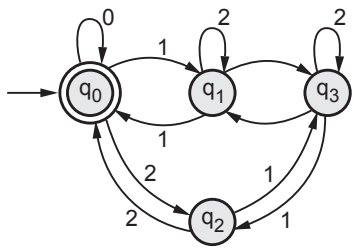
q_0 - Remainder 0 state

q_1 - Remainder 1 state

q_2 - Remainder 2 state

q_3 - Remainder 3 state

Here q_0 is start state as well as final state. The FA is,



For example

$(121)_3 = (16)_{10}$ which is divisible by 4.

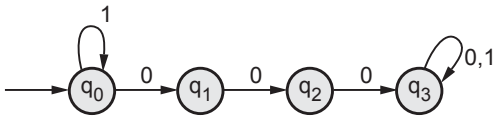
Example 1.9.33 *Design FA accepting the following language over the alphabet {0, 1}*

i) Set of all strings having at least three consecutive zeros.

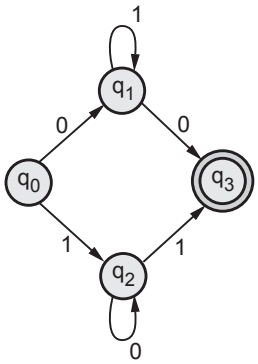
ii) Set of all strings that begin and end with same symbol.

SPPU : Dec.-18, End Sem, Marks 8

Solution : i)



ii)



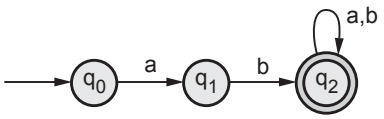
Example 1.9.34 *Construct a DFA accepting the following languages over the alphabet {a, b}*

i) Set of all strings that begin with the substring ab.

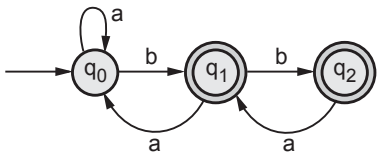
ii) Set of all strings with at most two consecutive b's.

SPPU : Dec.-18, End Sem, Marks 8

Solution : i)



ii)



Example 1.9.35 *Construct FA for the following language L.*

L = [w/w is a binary word of length 4i, i > = 1 such that each consecutive block 4 bits contains at least 2 0's]

SPPU : May-19, End Sem, Marks 8

Solution :

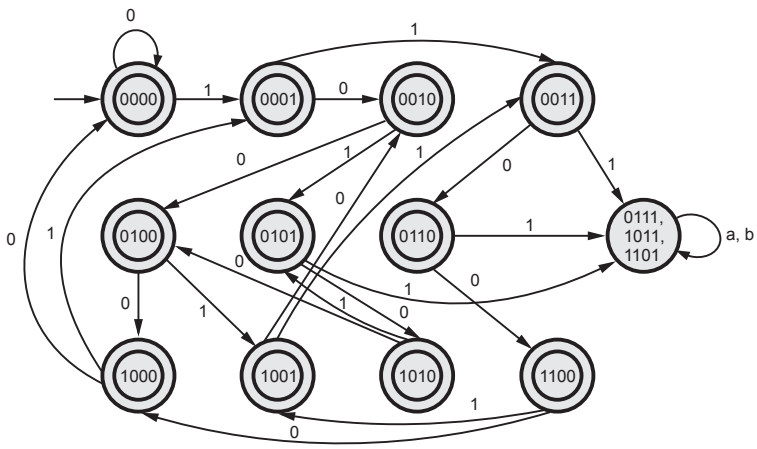


Fig. 1.9.27

Review Question

1. *Give formal definition of - Deterministic Finite Automata.*

SPPU : May-19, End Sem, Marks 2

1.10 Non Deterministic Finite Automata

SPPU : Dec.-12,13,14, May-12,14, Aug.-15,17, Marks 6

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.

Example

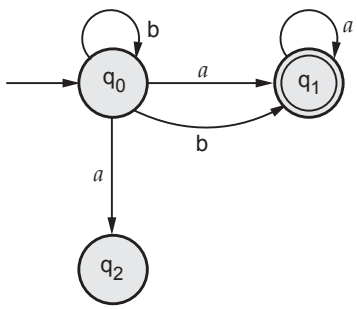


Fig. 1.10.1 Non deterministic finite automata

Difference in NFA and DFA

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

Problems Based on NFA

Example 1.10.1 *An NFA with states 1-5 and input alphabet {a, b} has following transition table :*

q	δ(q, a)	δ(q, b)
1	{1, 2}	{1}
2	{3}	{3}
3	{4}	{4}
4	{5}	ϕ
5	ϕ	{5}

i) Draw transition diagram ii) Calculate δ (1, ab) iii) Claculate δ* (1, abaab)*

SPPU : May-14, Marks 6

Solution :

i) Transition diagram

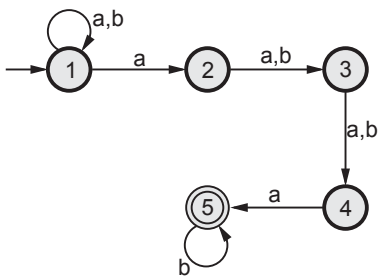


Fig. 1.10.2

ii) δ* (1, ab) = 1 or 3

iii) δ* (1, abaab) = 1 or 3 or 4 or 5

Example 1.10.2 *Give non-deterministic finite automata to accept the following language over {0, 1}*. The set of all strings containing either 101 or 110 as a substring.*

SPPU : Dec.-13, Marks 4

Solution :

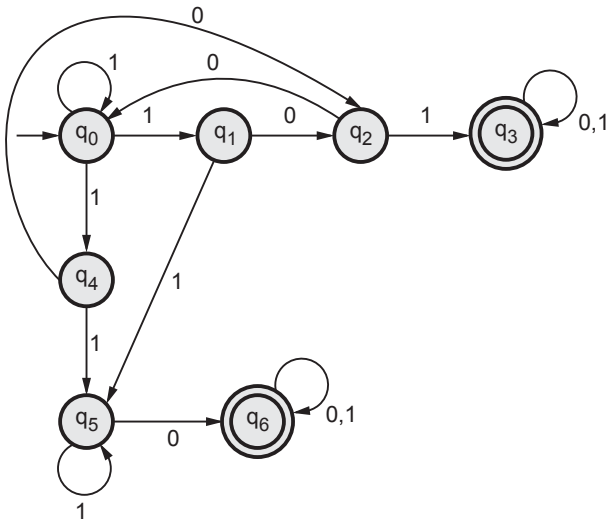


Fig. 1.10.3

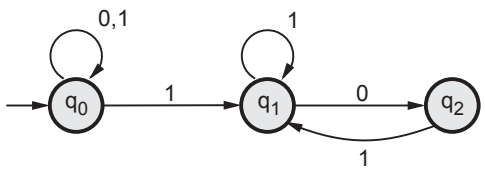
Example 1.10.3 *Is NFA more powerful than DFA ? Justify* **SPPU : Dec.-14, In Sem, Marks 2**

Solution : The NFA is not more powerful than DFA. In fact NFA can be converted to equivalent DFA. Thus any regular language can be represented by both NFA and DFA.

Example 1.10.4 *Draw NFA for strings ending with '10' over alphabet* $\sum = \{0,1\}^*$.

SPPU : Aug.-15, In Sem, Marks 2

Solution :



Review Questions

1. *Differentiate between NFA and DFA.*
   SPPU : May-12, Marks 2, May-14, Marks 4, Aug.17, In Sem, Marks 3
2. *Define formally with example - Non deterministic finite automata.*
   SPPU : Dec.-12, May-14, Marks 2

1.11 Epsilon- NFA

SPPU : Dec.-13,18,19, Aug.-17, May-11, Marks 5

The ε-transitions in NFA are given in order to move from one state to another without having any symbol from input set $\sum$.

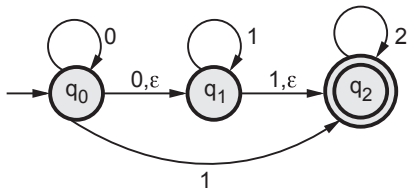


Fig. 1.11.1

- In this NFA with ε, q_0 is a **start state** with input 0.
- Initially with **input 0**, we can go to q_0 or q_1.
- Similarly with **input** 1, we can go to state q_1 or q_2.
- Thus it is not definite that on input 1 whether we will be in state q_1 or q_2. Hence it is called **non-deterministic Finite Automata** (NFA) and since there are some ε moves to simply change the states, it is called **NFA with** ε .

Definition

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 $Q \times \{\Sigma \cup \varepsilon\}$ to 2^Q.

q_0 is a start state.

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

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 1.11.1 *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.*

SPPU : Dec.-19, End Sem, Marks 3

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 -

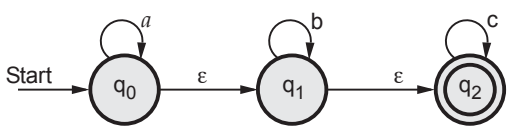


Fig. 1.11.2

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

δ	Σ			
State \ Input	*a*	b	c	ε
q_0	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
q_2	ϕ	ϕ	q_2	ϕ

We can parse the string *aa*bbcc as follows -

$$
\begin{aligned}
\delta(q_0, aabbcc) &\vdash \delta(q_0, abbcc) \\
&\vdash \delta(q_0, bbcc) \\
&\vdash \delta(q_0, \varepsilon bbcc) \\
&\vdash \delta(q_1, bbcc) \\
&\vdash \delta(q_1, bcc) \\
&\vdash \delta(q_1, cc) \\
&\vdash \delta(q_1, \varepsilon cc) \\
&\vdash \delta(q_2, cc) \\
&\vdash \delta(q_2, c) \\
&\vdash \delta(q_2, \varepsilon)
\end{aligned}
$$

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

Example 1.11.2 *Compare NFA and NFA - ε .* **SPPU : Aug.-17, In Sem, Marks 3**

Solution :

- The NFA and NFA ε both are non-deterministic in nature.
- There are no ε (empty) transitions in NFA whereas there are empty transitions present in NFA ε .
- NFA occupies less space whereas NFA ε occupies more space.

1.11.1 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 $\delta(q, \varepsilon) = r$ then ε - closure (p) = {q, r}.

Example 1.11.3 *Find ε - closure for the following NFA with ε .*

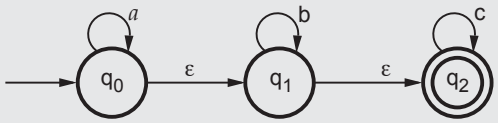


Fig. 1.11.3

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 $(q_2) = \{q_2\}$

Review Questions

1. *Describe - NFA with null moves.* **SPPU : May-11, Marks 2**
2. *Give the formal definitions of following with suitable examples : Epsilon closure, non-deterministic finite automata and deterministic finite automata.* **SPPU : Dec.-13, Marks 5**
3. *Explain the extended transition function NFA with epsilon.* **SPPU : Dec.-18, In Sem, Marks 3**

1.12 NFA with ε to NFA without ε **SPPU : Dec.-14,15, Oct.-16, May-19, Marks 6**

Steps for Conversion

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.

$$\delta'(q, a) = \varepsilon \text{ - closure}\left(\delta\left(\hat{\delta}(q,\varepsilon), a\right)\right)$$

where $$\hat{\delta}(q,\varepsilon) = \varepsilon \text{ - closure}(q)$$

Example 1.12.1 *Convert the given NFA with ε to NFA without ε .*

SPPU : Dec.-14, Oct.-16, In Sem, Marks 4, Dec.-15, End Sem, Marks 6

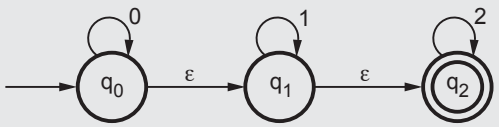


Fig. 1.12.1

Solution : We will first obtain ε - closure of each state i.e. we will find out ε - reachable states from current state.

Hence

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

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.

$$\begin{aligned}
\delta'(q_0, 0) &= \varepsilon \text{ - closure } \left(\delta\left(\hat{\delta}(q_0, \varepsilon), 0\right)\right) \\
&= \varepsilon \text{ - closure } \left(\delta\left(\varepsilon \text{ - closure}(q_0), 0\right)\right) \\
&= \varepsilon \text{ - closure } \left(\delta(q_0, q_1, q_2), 0\right) \\
&= \varepsilon \text{ - closure } \left(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)\right) \\
&= \varepsilon \text{ - closure } (q_0 \cup \phi \cup \phi) \\
&= \varepsilon \text{ - closure } (q_0) = \{q_0, q_1, q_2\} \\
\delta'(q_0, 1) &= \varepsilon \text{ - closure } \left(\delta\left(\hat{\delta}(q_0, \varepsilon), 1\right)\right) \\
&= \varepsilon \text{ - closure } \left(\delta(q_0, q_1, q_2), 1\right) \\
&= \varepsilon \text{ - closure } \left(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)\right) \\
&= \varepsilon \text{ - closure } (\phi \cup q_1 \cup \phi) \\
&= \varepsilon \text{ - closure } (q_1) \\
\delta'(q_0, 1) &= \{q_1, q_2\} \\
\delta'(q_1, 0) &= \varepsilon \text{ - closure } \left(\delta\left(\hat{\delta}(q_1, \varepsilon), 0\right)\right) \\
&= \varepsilon \text{ - closure } \left(\delta\left(\varepsilon \text{ - closure}(q_1), 0\right)\right) \\
&= \varepsilon \text{ - closure } \left(\delta(q_1, q_2), 0\right)
\end{aligned}$$

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

$$\delta'(q_1, 1) = \varepsilon\text{ - closure }\left(\delta\left(\hat{\delta}(q_1, \varepsilon), 1\right)\right)$$
$$= \varepsilon\text{ - closure }\left(\delta\left(\varepsilon\text{ - closure}(q_1), 1\right)\right)$$
$$= \varepsilon\text{ - closure }\left(\delta(q_1, q_2), 1\right)$$
$$= \varepsilon\text{ - closure }\left(\delta(q_1, 1) \cup \delta(q_2, 1)\right)$$
$$= \varepsilon\text{ - closure }(q_1 \cup \phi)$$
$$= \varepsilon\text{ - closure }(q_1)$$
$$= \{q_1, q_2\}$$

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

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

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

$$\delta'(q_0, 2) = \varepsilon\text{ - closure }\left(\delta\left(\hat{\delta}(q_0, \varepsilon), 2\right)\right)$$
$$= \varepsilon\text{ - closure }\left(\delta\left(\varepsilon\text{ - closure}(q_0), 2\right)\right)$$
$$= \varepsilon\text{ - closure }\left(\delta(q_0, q_1, q_2), 2\right)$$
$$= \varepsilon\text{ - closure }\left(\delta(q_0, 2) \cup \delta(q_1, 2) \cup \delta(q_2, 2)\right)$$
$$= \varepsilon\text{ - closure }(\phi \cup \phi \cup q_2)$$
$$= \varepsilon\text{ - closure }(q_2)$$
$$= \{q_2\}$$

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

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

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

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

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

$$= \{q_2\}$$

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

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

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

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

$$= \{q_2\}$$

Now we will summarize all the computed δ' transitions -

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

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

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

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

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

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

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

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

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

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

The NFA will be as shown in Fig. 1.12.2.

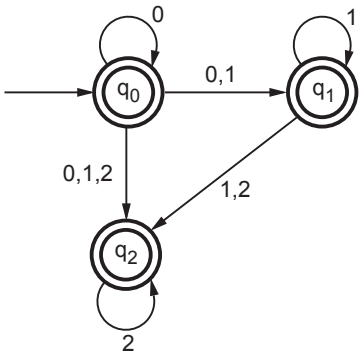


Fig. 1.12.2

Here q_0, q_1 and q_2 is a final state because ε - closure (q_0), ε - closure (q_1) and ε - closure (q_2) contains final state q_2.

Example 1.12.2 *Convert the following NFA with ε to NFA without ε .*

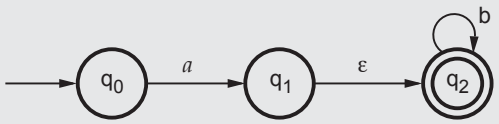


Fig. 1.12.3

Solution : We will first obtain ε - closures of q_0, q_1 and q_2 as follows.

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

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

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

Now the δ′ transitions on each input symbol is obtained as

$$\begin{aligned}
\delta'(q_0, a) &= \varepsilon\text{ - closure}\left(\delta\left(\hat{\delta}(q_0, \varepsilon), a\right)\right) \\
&= \varepsilon\text{ - closure}\left(\delta\left(\varepsilon\text{ - closure}(q_0), a\right)\right) \\
&= \varepsilon\text{ - closure}\left(\delta(q_0, a)\right) \\
&= \varepsilon\text{ - closure}(q_1) \\
&= \{q_1, q_2\} \\
\delta'(q_0, b) &= \varepsilon\text{ - closure}\left(\delta\left(\hat{\delta}(q_0, \varepsilon), b\right)\right) \\
&= \varepsilon\text{ - closure}\left(\delta\left(\varepsilon\text{ - closure}(q_0), b\right)\right)
\end{aligned}$$

$$= \varepsilon\text{ - closure}\left(\delta\left(q_0, b\right)\right)$$

$$= \phi$$

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

$$= \varepsilon\text{ - closure}\left(\delta\left(\varepsilon\text{ - closure}(q_1), a\right)\right)$$

$$= \varepsilon\text{ - closure}\left(\delta(q_1, q_2), a\right)$$

$$= \varepsilon\text{ - closure}\left(\delta(q_1, a) \cup \delta(q_2, a)\right)$$

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

$$= \phi$$

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

$$= \varepsilon\text{ - closure}\left(\delta\left(\varepsilon\text{ - closure}(q_1), b\right)\right)$$

$$= \varepsilon\text{ - closure}\left(\delta(q_1, q_2), b\right)$$

$$= \varepsilon\text{ - closure}\left(\delta(q_1, b) \cup \delta(q_2, b)\right)$$

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

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

$$= \{q_2\}$$

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

$$= \varepsilon\text{ - closure}\left(\delta\left(\varepsilon\text{ - closure}(q_2), a\right)\right)$$

$$= \varepsilon\text{ - closure}\left(\delta(q_2, a)\right)$$

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

$$= \phi$$

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

$$= \varepsilon\text{ - closure}\left(\delta\left(\varepsilon\text{ - closure}(q_2), b\right)\right)$$

$$= \varepsilon\text{ - closure}\left(\delta(q_2, b)\right)$$

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

$$= \{q_2\}$$

The transition table can be -

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

States q_1 and q_2 becomes the final as ε - closures of q_1 and q_2 contains the final state q_2. The NFA can be shown by following transition diagram -

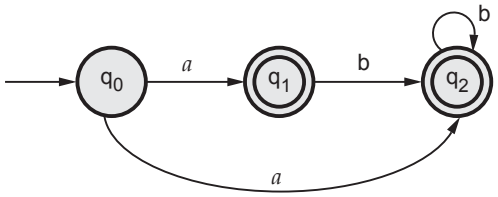


Fig. 1.12.4

Example 1.12.3 *Convert the following NFA with ε-moves into NFA without ε moves.*

SPPU : May-19, End Sem, Marks 6

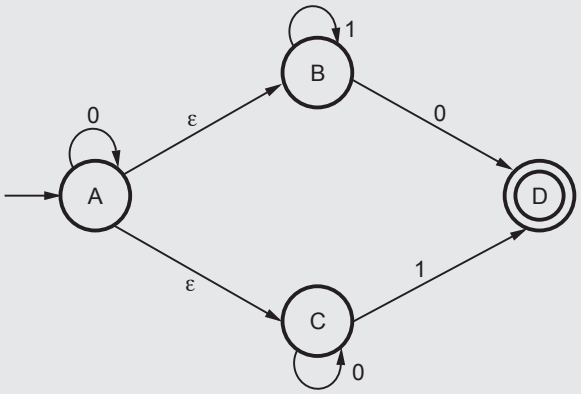


Fig. 1.12.5

Solution : We will compute ε closures of A, B, C, D

ε - closure (A) = {A, B, C}

ε - closure (B) = {B}

ε - closure (C) = {C}

ε - closure (D) = {D}

The δ′ transitions on each input symbol is

δ' (A,O) = ε - closure (δ (ε - closure (A),O))

= ε - closure (δ (A, B, C), O)

= ε - closure (δ (A,O) ∪δ (B,O) ∪δ (C,O))

= ε - closure (A, D, C)

= {A,B,C} ∪ {D}, {C}

δ' **(A,O) = {A,B,C,D}**

δ' (A,1) = ε - closure (δ (ε - closure (A),1))

= ε - closure (δ (A,B,C),1)

= ε - closure (δ (A,1) ∪ δ (B, 1) ∪ δ (C, 1))

= ε - closure (ϕ ∪ B ∪ D)

= ε - closure (B ∪ D)

δ' **(A,1) = {B, D}**

δ' (B,0) = ε - closure (δ (ε - closure (B), 0))

= ε - closure (δ (B, 0))

= ε - closure (D)

δ' **(B,0) = D**

δ' (B,1) = ε - closure (δ (ε - closure (B),1))

= ε - closure (δ (B, 1))

= ε - closure (B)

δ' **(B,1) = B**

δ' (C,0) = ε - closure (δ (ε - closure (C),0))

= ε - closure (δ (C, 0))

= ε - closure (C)

δ' **(C,0) = C**

δ' (C,1) = ε - closure (δ (ε - closure (C),1))

= ε - closure (δ (C, 1))

= ε - closure (D)

δ' **(C,1) = D**

δ' (D,1) = ε - closure (δ (ε - closure (D),0))

$= \varepsilon$ - closure (δ (D, 0))

$= \varepsilon$ - closure (ϕ)

δ' **(D,0)** = ϕ

δ' (D,1) = ε - closure (δ (ε - closure (D),1))

$= \varepsilon$ - closure (ϕ)

δ' **(D,1)** = ϕ

The transition table for NFA without ε is

State \ i/p	0	1
A	{A, B, C, D}	{B, D}
B	{D}	{B}
C	{C}	{D}
D	ϕ	ϕ

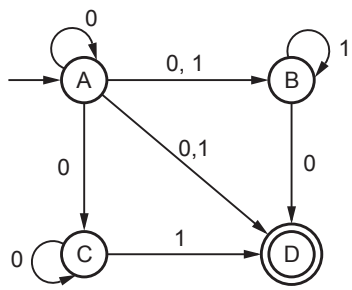


Fig. 1.12.6

1.13 NFA to DFA Conversion

SPPU : Dec.-05,09,13,17, May-09,10,12, Aug.-15,17, Marks 10

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 $[q_1, q_2, q_3, \ldots\ldots\ldots q_i]$ in Q′ the transitions for each input symbol Σ can be obtained as,

   i) $\delta'([q_1, q_2, \ldots\ldots\ldots q_i], a) = \delta(q_1, a) \cup \delta(q_2, a) \cup \ldots\ldots\ldots \delta(q_i, a)$

   $= [q_1, q_2, \ldots\ldots q_k]$ may be some state.

   ii) Add the state $[q_1, q_2, \ldots\ldots q_k]$ to DFA if it is not already added in Q′.

   iii) Then find the transitions for every input symbol from Σ for state $[q_1, q_2, \ldots\ldots q_k]$. If we get some state $[q_1, q_2, \ldots\ldots q_n]$ 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 $[q_1, q_2, \ldots\ldots q_n] \in Q'$ of DFA if any one state q_i is a final state of NFA then $[q_1, q_2, \ldots\ldots q_n]$ becomes a final state. Thus the set of all the final states $\in F'$ of DFA.

Example 1.13.1 *Convert the given NFA to DFA.*

State \ Input	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	q_0
q_1	q_2	q_1
q_2	q_3	q_3
q_3	ϕ	ϕ

Solution : As we know, first we will compute δ' function.

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

Hence $\delta'([q_0], 0) = [q_0, q_1]$

Similarly,

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

Hence $\delta'([q_0], 1) = [q_0]$

Thus we have got a new state $[q_0, q_1]$.

Let us check how it behaves on input 0 and 1.

So,

$$\begin{aligned}\delta'([q_0, q_1], 0) &= \delta([q_0], 0) \cup \delta([q_1], 0)\\ &= \{q_0, q_1\} \cup \{q_2\}\\ &= \{q_0, q_1, q_2\}\end{aligned}$$

Hence a new state is generated i.e. $[q_0, q_1, q_2]$

Similarly,

$$\begin{aligned}\delta'([q_0, q_1], 1) &= \delta([q_0], 1) \cup \delta([q_1], 1)\\ &= \{q_0\} \cup \{q_1\}\\ &= \{q_0, q_1\}\end{aligned}$$

No new state is generated here.

Again δ' function will be computed for $[q_0, q_1, q_2]$, the new state being generated.

State \ Input	0	1
$[q_0]$	$[q_0, q_1]$	$[q_0]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_1]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_3]$

As, you have observed in the above table for a new state $[q_0, q_1, q_2]$ the input 0 will give a new state $[q_0, q_1, q_2, q_3]$ and input 1 will give a new state $[q_0, q_1, q_3]$, because

$$\begin{aligned}\delta'([q_0, q_1, q_2], 0) &= \delta'([q_0], 0) \cup \delta'([q_1], 0) \cup \delta'([q_2], 0)\\ &= \{q_0, q_1\} \cup \{q_2\} \cup \{q_3\}\\ &= \{q_0, q_1, q_2, q_3\}\\ &= [q_0, q_1, q_2, q_3]\end{aligned}$$

Same procedure for input 1. Thus the final DFA is as given below.

State	0	1
$\rightarrow [q_0]$	$[q_0, q_1]$	$[q_0]$
$[q_1]$	$[q_2]$	$[q_1]$
$[q_2]$	$[q_3]$	$[q_3]$
$[q_3]$	ϕ	$[q_2]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_1]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_3]$
$[q_0, q_1, q_3]$	$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$
$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$

DFA for example 1.13.1

Example 1.13.2 *Construct DFA equivalent to the given NFA.* **SPPU : Dec.-05, Marks 10**

State \ Input	0	1
→ 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
→ [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
→ [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, r]
[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.

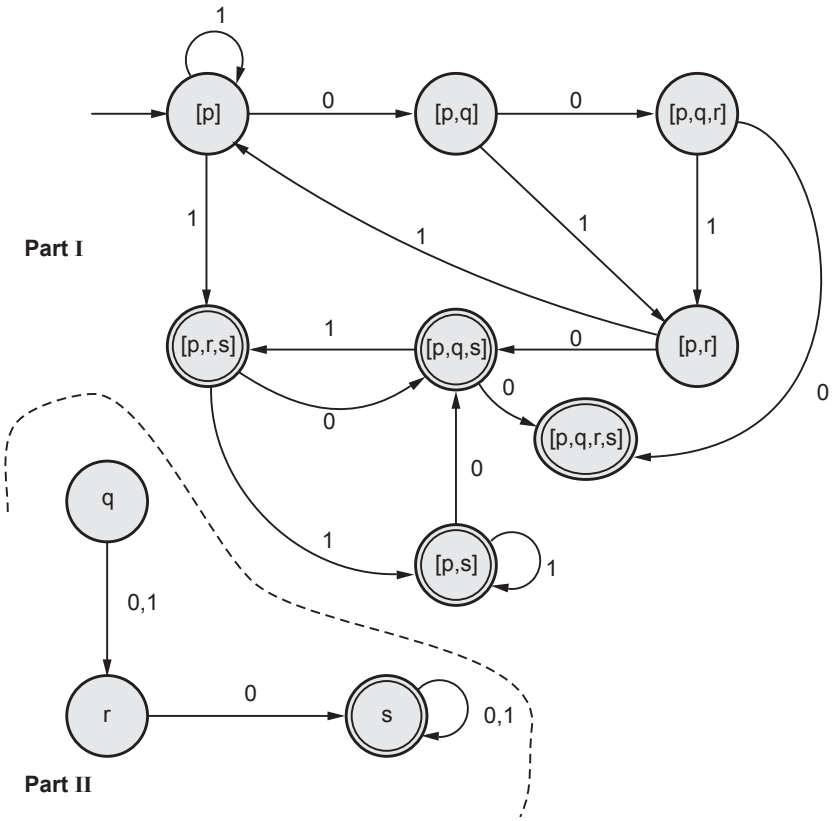


Fig. 1.13.1

Example 1.13.3 *Convert the given NFA to DFA.*

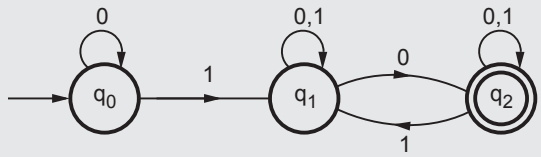


Fig. 1.13.2

Solution : For the given transition diagram in problem statement we will first construct the transition table.

State / Σ	0	1
→ q_0	q_0	q_1
q_1	$\{q_1, q_2\}$	q_1
q_2	$\{q_2\}$	$\{q_1, q_2\}$

Now we will obtain δ′ transitions for each state.

$$\delta'([q_0], 0) = \{q_0\}$$
$$= [q_0]$$
$$\delta'([q_0], 1) = \{q_1\}$$
$$= [q_1]$$
$$\delta'([q_1], 0) = \{q_1, q_2\}$$
$$= [q_1, q_2] \rightarrow \text{new state generated.}$$
$$\delta'([q_1], 1) = \{q_1\} = [q_1]$$
$$\delta'([q_2], 0) = \{q_2\}$$
$$= [q_2]$$
$$\delta'([q_2], 1) = \{q_1, q_2\}$$
$$= [q_1, q_2]$$

Now we will obtain δ′ transitions on $[q_1, q_2]$.

$$\delta'([q_1, q_2], 0) = \delta([q_1], 0) \cup \delta([q_2], 0)$$
$$= \{q_1, q_2\} \cup \{q_2\}$$
$$= \{q_1, q_2\}$$
$$= [q_1, q_2]$$
$$\delta'([q_1, q_2], 1) = \delta([q_1], 1) \cup \delta([q_2], 1)$$
$$= \{q_1\} \cup \{q_1, q_2\}$$
$$= [q_1, q_2]$$

The state $[q_1, q_2]$ is final state as well because it contains a final state q_2. The transition table for the constructed DFA will be -

State / Σ	0	1
→ $[q_0]$	$[q_0]$	$[q_1]$
$[q_1]$	$[q_1, q_2]$	$[q_1]$
$[q_2]$	$[q_2]$	$[q_1, q_2]$
$[q_1, q_2]$	$[q_1, q_2]$	$[q_1, q_2]$

The transition diagram will be -

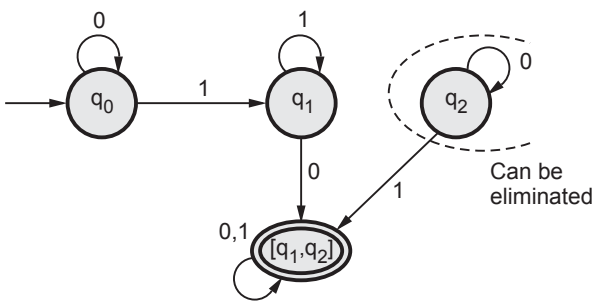


Fig. 1.13.3

Consider a string 0010 we will get -

Using NFA

$q_0\ 0010 \vdash 0q_0 0110$

$\vdash 00q_0 110$

$\vdash 001q_1 10$

$\vdash 0011q_1 0$

$\vdash 00110q_2$ or $\vdash 00110q_1$

Accept non-accept.

Using DFA

$q_0 00110 \vdash 0q_0 0110$

$\vdash 00q_0 110$

$\vdash 001q_1 10$

$\vdash$ $0011q_1 0$

$\vdash$ $00110\ [q_1, q_2]$

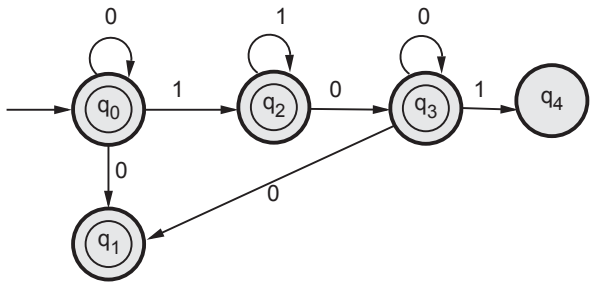
Accept state.

Thus both the NFA and DFA accept the same string 00110.

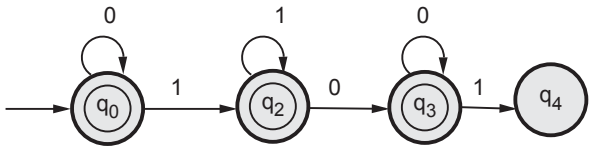
Example 1.13.4 *Construct a NFA and then equivalent DFA accepting strings over {0,1}, for accepting all possible strings of zeroes and ones not containing 101 as a substring.*

SPPU : May-09; Dec.-09, Marks 10

Solution : The NFA representing the strings that do not contain 101 as a substring is



The equivalent DFA will be

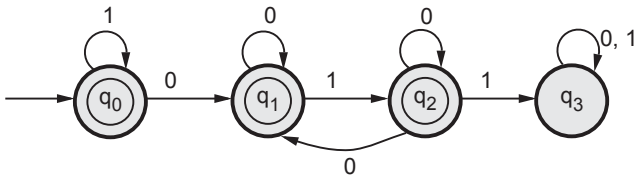


Example 1.13.5 *Construct a NFA and then equivalent DFA accepting string over {0, 1}, for accepting all possible strings of zeroes and ones which does not contain 011 a a substring.*

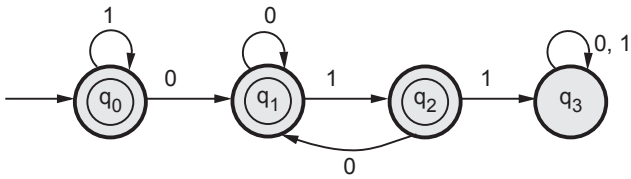
SPPU : May-09, 10, 12, Dec.-09, Marks 10

Solution :

NFA will be



DFA will be



Example 1.13.6 *Convert given non-deterministic finite automata (NFA) to deterministic finite automata (DFA). Informally describe the language it accepts.*

SPPU : Dec.-13, Marks 6

δ	0	1
p	{p, q}	{p}
q	ϕ	{r}
*r	{p, r}	{q}

Solution :

$\delta(p, 0) = \{p, q\} \rightarrow$ **[p, q]** new state

$\delta(p, 1) = \{p\} = [p]$ state

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

$\delta(q, 1) = \{r\} = [r]$ state

$\delta(r, 0) = \{p, r\} =$ **[p, r] new state**

$\delta(r, 1) = \{q\} = [q]$ state

Now, we will find input transitions on newly generated states.

$\delta([p, q], 0) = [p, q]$

$\delta([p, q], 1) = [p, r]$

$\delta([p, r], 0) = \{p, q, r\} =$ **[p, q, r] new state**

$\delta([p, r], 1) = \{p, q\} =$ **[p, q]** new state

$\delta([p, q, r], 0) = [p, q, r]$

$\delta([p, q, r], 1) = [p, q, r]$

The transition table for NFA is -

		0	1
→	[p]	[p, q]	[p]
	[q]	ϕ	[r]
*	[r]	[p, r]	[q]
	[p, q]	[p, q]	[p, r]
*	[p, r]	[p, q, r]	[p, q]
*	[p, q, r]	[p, q, r]	[p, q, r]

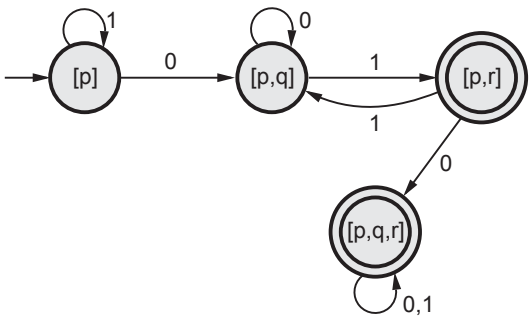


Fig. 1.13.4

Example 1.13.7 *Convert following NFA into its equivalent DFA.*

SPPU : Aug.-15, In Sem, Marks 4

Q \ Σ –>	0	1
–> P	P, Q	R
Q	R	R
R	S	Q
* S	S	S

Solution :

$\delta(P, 0) = \{P, Q\} =$ **[P, Q]** New state

$\delta(P, 1) = R$

$\delta(Q, 0) = R$

$\delta(Q, 1) = R$

$\delta(R, 0) = S$

$\delta(R, 1) = Q$

$\delta(S, 0) = S$

$\delta(S, 1) = S$

The δ transitions can be applied on new state [P, Q]

$\therefore \quad \delta([P, Q], 0) = \{P, Q, R\} =$ **[P, Q, R]** New state

$\delta([P, Q], 1) = R$

$\delta([P, Q, R], 0) = \{P, Q, R, S\} =$ **[P, Q, R, S]** New state

$\delta([P, Q, R), 1) = \{R, Q\} =$ **[R, Q]** New state

$\delta([P, Q, R, S], 0)$ = [P, Q, R, S]

$\delta([P, Q, R, S), 1)$ = {R, Q, S} = **[R, Q, S]** New state

$\delta([R, Q], 0)$ = {R, S} = **[R, S]** New state

$\delta([R, Q), 1)$ = [R, Q]

$\delta([R, Q, S], 0)$ = **[R, S]**

$\delta([R, Q, S), 1)$ = [R, Q, S]

$\delta([R, S], 0)$ = S

$\delta([R, S), 1)$ = **[Q, S]** New state

$\delta([Q, S], 0)$ = [R, S]

$\delta([Q, S), 1)$ = [R, S]

As no new state is getting generated we will stop applying δ transitions. The transition table for DFA is as follows.

	State \ Input	0	1
→	[P]	[P, Q]	[R]
	[Q]	[R]	[R]
	[R]	[S]	[Q]
*	[S]	[S]	[S]
	[P, Q]	[P, Q, R]	[R]
	[P, Q, R]	[P, Q, R, S]	[R, Q]
*	[P, Q, R, S]	[P, Q, R, S]	[R, Q, S]
	[R, Q]	[R, S]	[R, Q]
*	[R, Q, S]	[R, S]	[R, Q, S]
*	[R, S]	[S]	[Q, S]
*	[Q, S]	[R, S]	[R, S]

Example 1.13.8 *Construct an equivalent DFA for the following NFA :*

SPPU : Aug.-17, In Sem, Marks 4

States/Σ	0	1
→ p	{p, q}	{q}
q	{r}	{r}
r	–	{r}

Solution : Let us apply δ transitions on each state using each input symbol.

$\delta(p, 0) = \{p, q\}$ = **[p, q] new state**

$\delta(p, 1) = \{q\}$

$\delta([p, q], 0) = \{p, q, r\}$ = **[p, q, r] new state**

$\delta([p, q], 1) = \{q, r\}$ = **[q, r] new state**

$\delta([p, q, r], 0) = [p, q, r]$

$\delta([p, q, r], 1) = [q, r]$

$\delta([q, r], 0) = [r]$

$\delta([q, r], 1) = [r]$

As no new states are getting generated. There are no δ transitions.

The transition table will be

State/Σ	0	1
[p]	[p, q]	[q]
* [p, q]	[p, q, r]	[q, r]
* [p, q, r]	[p, q, r]	[q, r]
* [q, r]	[r]	[r]
* [q]	[r]	[r]
[r]	-	[r]

Here r state can be eliminated

The DFA will be

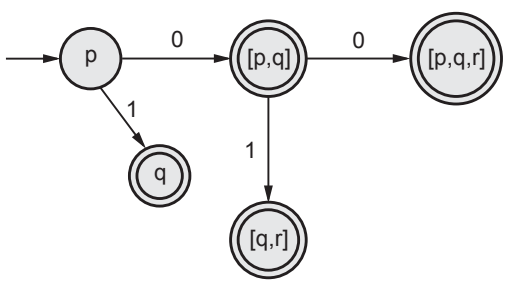


Fig. 1.13.5

Example 1.13.9 *Let M = {{q_0, q_1}, {0, 1}, δ, q_0, {q_1}} be an NFA*

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

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

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

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

Construct an equivalent DFA. **SPPU : Dec.-17, End Sem, Marks 4**

Solution : Let us assume δ function as follows :

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

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

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

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

Let us apply δ transition on $\{q_0, q_1\}$ state

$$\delta([q_0, q_1], 0) = [q_0, q_1]$$

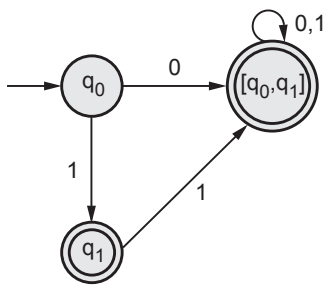
$$\delta([q_0, q_1], 1) = [q_0, q_1]$$

State \ i/p	0	1
→ q_0	$\{q_0, q_1\}$	$\{q_1\}$
q_1	ϕ	$\{q_0, q_1\}$

Transition table

As there are no new states getting generated, there is no application of δ transition. Hence the transition table and transition diagram for equivalent DFA is

State \ i/p	0	1
$[q_0]$	$[q_0, q_1]$	$[q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$
$[q_1]$	ϕ	$[q_0, q_1]$



1.14 NFA with Epsilon to DFA Conversion

SPPU : Dec.-11, Marks 6

Subset Construction Method Algorithm

Step 1 : Initialization step

Let, ε - closure (s) be the states in Dstates of DFA.

Step 2 : Repeatation step

```
While (T are the unmarked states in Dstates)
    {
        mark T
        for (each symbol a)
        {
            U = ε - closure (move (T, a))
            if (U is not in Dstates)
                Add U as unmarked state in Dstates
            Dtran [T, a] = U
        } // end of for
    } // end of while.
```

Example 1.14.1 *Convert the following NFA with ε to equivalent DFA.*

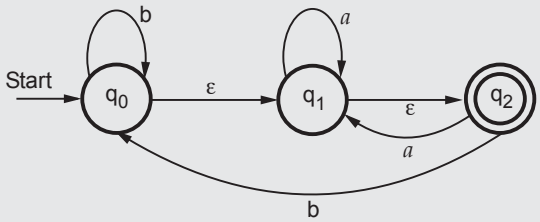


Fig. 1.14.1

Solution : To convert this NFA we will first find ε-closures.

ε - closures $\{q_0\} = \{q_0, q_1, q_2\}$

ε - closures $\{q_1\} = \{q_1, q_2\}$

ε - closures $\{q_2\} = \{q_2\}$

Let us start from ε-closure of start state

ε - closure $\{q_0\} = \{q_0, q_1, q_2\}$ we will call this **state as A.**

Now let us find transitions on A with every input symbol.

$$\delta'(A, a) = \varepsilon \text{ - closure } (\delta(A, a))$$

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

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

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

$= \{q_1, q_2\}$. Let us call it as **state B.**

$$\delta'(A, b) = \varepsilon \text{ - closure } (\delta(A, b))$$

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

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

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

$= \{q_0, q_1, q_2\}$ i.e. A.

Hence we can state that

$$\delta'(A, a) = B$$

$$\delta'(A, b) = A$$

Now let us find transitions for state $B = \{q_1, q_2\}$.

$$\delta'(B, a) = \varepsilon \text{ - closure } (\delta(q_1, q_2), a)$$

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

$$= \{q_1, q_2\} \text{ i.e. B}$$

$$\delta'(B, b) = \varepsilon \text{ - closure } (\delta(q_1, q_2), b)$$

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

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

$$= \{q_0, q_1, q_2\} \text{ i.e. A.}$$

Hence the generated DFA is

	a	b
→ A	B	A
B	B	A

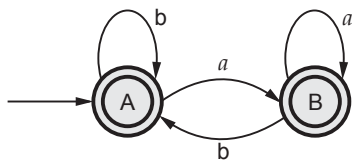


Fig. 1.14.2

Example 1.14.2 *Convert the given NFA into its equivalent DFA -*

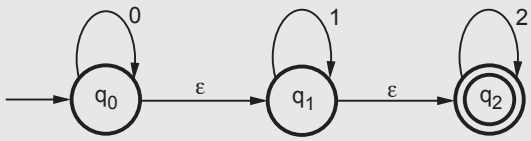


Fig. 1.14.3

Solution : Let us obtain ε - closure of each state.

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

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

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

Now we will obtain δ′ transition. Let ε - closure $(q_0) = \{q_0, q_1, q_2\}$ call it as **state A**.

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

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

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

$$= \{q_0, q_1, q_2\} \text{ i.e. } \textbf{state A}$$

$$\delta'(A,1) = \varepsilon\text{ - closure }\{\delta((q_0, q_1, q_2), 1)\}$$
$$= \varepsilon\text{ - closure }\{\delta(q_0,1) \cup \delta(q_1,1) \cup \delta(q_2,1)\}$$
$$= \varepsilon\text{ - closure }\{q_1\}$$
$$= \{q_1, q_2\}.\text{ Call it as } \textbf{state B}$$
$$\delta'(A,2) = \varepsilon\text{ - closure }\{\delta((q_0, q_1, q_2), 2)\}$$
$$= \varepsilon\text{ - closure }\{\delta(q_0,2) \cup \delta(q_1,2) \cup \delta(q_2,2)\}$$
$$= \varepsilon\text{ - closure }\{q_2\}$$
$$= \{q_2\}.\text{ Call it as } \textbf{state C.}$$

Thus we have obtained

$$\delta'(A,0) = A$$
$$\delta'(A,1) = B$$
$$\delta'(A,2) = C$$

i.e.

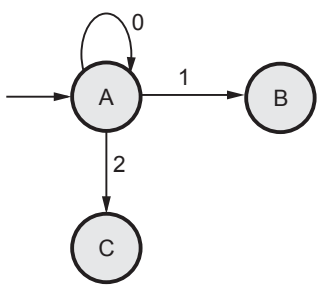


Fig. 1.14.4

Now we will find transitions on states B and C for each input.

Hence

$$\delta'(B,0) = \varepsilon\text{ - closure }\{\delta(q_1, q_2), 0\}$$
$$= \varepsilon\text{ - closure }\{\delta(q_1,0) \cup \delta(q_2,0)\}$$
$$= \varepsilon\text{ - closure }\{\phi\}$$
$$= \phi$$
$$\delta'(B,1) = \varepsilon\text{ - closure }\{\delta(q_1, q_2), 1\}$$
$$= \varepsilon\text{ - closure }\{\delta(q_1,1) \cup \delta(q_2,1)\}$$
$$= \varepsilon\text{ - closure }\{q_1\}$$

$$= \{q_1, q_2\} \text{ i.e. } \textbf{state B itself}.$$

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

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

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

$$= \{q_2\} \text{ i.e } \textbf{state C}.$$

Hence

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

$$\delta'(B, 1) = B$$

$$\delta'(B, 2) = C$$

The partial transition diagram will be

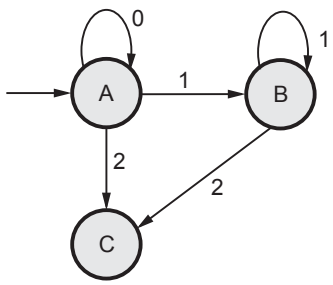


Fig. 1.14.5

Now we will obtain transitions for C :

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

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

$$= \phi$$

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

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

$$= \phi$$

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

$$= q_2$$

Hence the DFA is

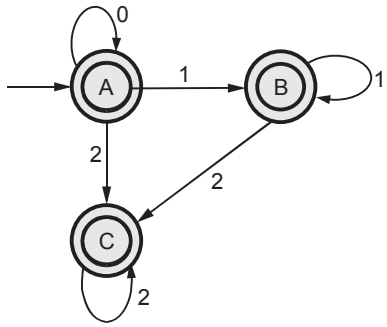


Fig. 1.14.6

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 1.14.3 *Convert the given NFA with ε to its equivalent DFA.*

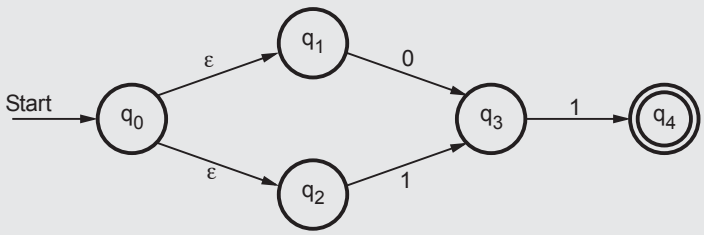


Fig. 1.14.7

Solution :

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

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

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

$$\varepsilon \text{ - closure } \{q_3\} = \{q_3\}$$

$$\varepsilon \text{ - closure } \{q_4\} = \{q_4\}$$

Now, Let ε - closure $\{q_0\} = \{q_0, q_1, q_2\}$ be state A.

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

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

$$= \varepsilon \text{ - closure } \{q_3\}$$

$$= \{q_3\} \text{ call it as state B.}$$

$$\delta'(A,1) = \varepsilon\text{ - closure }\{\delta((q_0, q_1, q_2),1)\}$$
$$= \varepsilon\text{ - closure }\{\delta(q_0,1) \cup \delta(q_1,1) \cup \delta(q_2,1)\}$$
$$= \varepsilon\text{ - closure }\{q_3\}$$
$$= q_3 = B.$$

The partial DFA will be

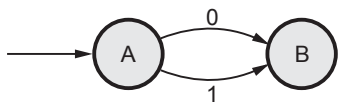


Fig. 1.14.8

Now,

$$\delta'(B,0) = \varepsilon\text{ - closure }\{\delta(q_3, 0)\}$$
$$= \phi$$
$$\delta'(B,1) = \varepsilon\text{ - closure }\{\delta(q_3, 1)\}$$
$$= \varepsilon\text{ - closure }\{q_4\}$$
$$= \{q_4\} \quad \text{i.e. state C}$$
$$\delta'(C,0) = \varepsilon\text{ - closure }\{\delta(q_4, 0)\}$$
$$= \phi$$
$$\delta'(C,1) = \varepsilon\text{ - closure }\{\delta(q_4, 1)\}$$
$$= \phi$$

The DFA will be,

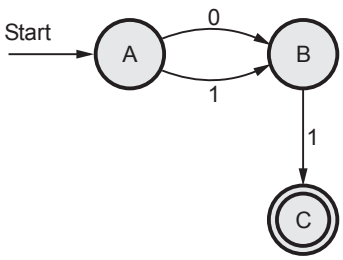


Fig. 1.14.9

Example 1.14.4 *Convert the given NFA– ∧ to an NFA.*

SPPU : Dec.-11, Marks 6

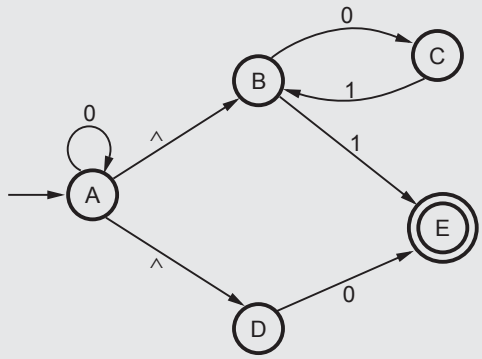


Fig. 1.14.10

Solution : We will ε-closures of A, B, C, D and E.

ε-closure (A) = {A, B, D}

ε-closure (B) = {B}

ε-closure (C) = {C}

ε-closure (D) = {D}

ε-closure (E) = {E}

The δ′ transitions on each input symbol is -

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

$$= \varepsilon\text{-closure}(\delta((A, B, D), 0))$$

$$= \varepsilon\text{-closure}(A, C, E)$$

$$= \varepsilon\text{-closure}(A) \cup \varepsilon\text{-closure}(C) \cup \varepsilon\text{-closure}(E)$$

$$= \{A, B, D\} \cup \{C\} \cup \{E\}$$

$$\delta'(A, 0) = \{A, B, C, D, E\}$$

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

$$= \varepsilon\text{-closure}(\delta((A, B, D), 1))$$

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

$$\delta'(A, 1) = \{E\}$$

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

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

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

$$\delta'(B, 0) = \{C\}$$

$$\delta'(B, 1) = \{E\}$$

$$\delta'(C, 0) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(C), 0))$$
$$= \varepsilon\text{-closure}(\delta(C, 0))$$
$$\delta'(C, 0) = \phi$$
$$\delta'(C, 1) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(C), 1))$$
$$= \varepsilon\text{-closure}(\delta(C, 1))$$
$$= \varepsilon\text{-closure}(B)$$
$$\delta'(C, 1) = \{B\}$$
$$\delta'(D, 0) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(D), 0))$$
$$= \varepsilon\text{-closure}(\delta(D, 0))$$
$$= \varepsilon\text{-closure}(E)$$
$$= \{E\}$$
$$\delta'(D, 1) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(D), 1))$$
$$= \varepsilon\text{-closure}(\delta(D, 1))$$
$$= \varepsilon\text{-closure}(\phi)$$
$$\delta'(D, 1) = \phi$$
$$\delta'(E, 0) = \varepsilon\text{-closure}(\delta(E, 0))$$
$$= \varepsilon\text{-closure}(\phi)$$
$$\delta'(E, 0) = \phi$$

Similarly, $\delta'(E, 1) = \phi$

The NFA can be represented by following transition table.

State \ i/p	0	1
A	{A, B, C, D, E}	E
→ B	C	E
C	ϕ	B
D	E	ϕ
E	ϕ	ϕ

1.15 Minimization of DFAs

SPPU : Dec.-12, Marks 8

- The minimization of FSM means reducing the number of states from given FA.
- While minimizing FSM we first find out which two states are equivalent we can represent those two states by one representative state.
- **Definition of equivalent states -** The two states q_1 and q_2 are equivalent if both $\delta(q_1, x)$ and $\delta(q_2, x)$ are final states or both of them are non final states for all $x \in \Sigma^*$ (Σ^* indicate any string of any length) we can minimize the given FSM by finding equivalent states.

Method for Construction of Minimum State Automata :

Step 1 : We will create a set π_0 as $\pi_0 = \left\{ Q_1^0, Q_2^0 \right\}$ where Q_1^0 is set of all final states and $Q_2^0 = Q - Q_1^0$ where Q is a set of all the states in DFA.

Step 2 : Now we will construct π_{K+1} from π_K. Let Q_i^K be any subset in π_K. If q_1 and q_2 are in Q_i^K they are (K + 1) equivalent provided $\delta(q_1, a)$ and $\delta(q_2, a)$ are K equivalent. Find out whether $\delta(q_1, a)$ and $\delta(q_2, a)$ are residing in the same equivalence class π_K. Then it is said that q_1 and q_2 are (K + 1) equivalent. Thus from Q_i^K we create (K + 1) equivalence classes. Repeat step 2 for every Q_i^K in π_K and obtain all the elements of π_{K+1}.

Step 3 : Construct π_n for n = 1, 2, ... until $\pi_n = \pi_{n+1}$.

Step 4 : Then replace all the equivalent states in one equivalence class by representative state. This helps in minimizing the given DFA.

Let us understand this method with the help of some examples.

Example 1.15.1 *Construct the minimum state automaton for the following transition diagram.*

SPPU : Dec.-12, Marks 8

Solution : We will first construct a transition table for the given DFA.

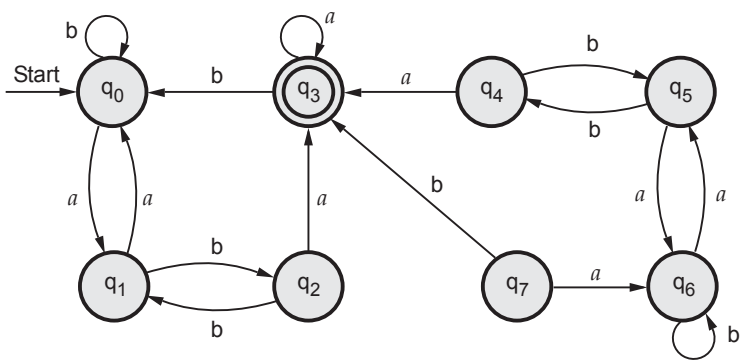


Fig. 1.15.1

We will start constructing equivalence classes.

State \ Input	a	b
→ q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
q_3	q_3	q_0
q_4	q_3	q_5
q_5	q_6	q_4
q_6	q_5	q_6
q_7	q_6	q_3

Step 1 : We will first find 0 - equivalence. For that purpose we will create two sets - one set containing all the final states and other set containing all the non-final states.

$\therefore$ 0 - Equivalent $= \{q_0, q_1, q_2, q_4, q_5, q_6, q_7\} \quad \{q_3\}$

Now will check equivalence among all the states of $\{q_0, q_1, q_2, q_4, q_5, q_6, q_7\}$ by means of δ mapping functions

$\delta(q_0, a) = q_1$

$\delta(q_1, a) = q_0$

↑ Belong to same set

$\delta(q_0, b) = q_0$

$\delta(q_1, b) = q_2$

↑ Belong to same set

$\therefore$ q_0 and q_1 are equivalent. In this way we will consider every pair from the set $\{q_0, q_1, q_2, q_4, q_5, q_6, q_7\}$

Now consider pair $\{q_0, q_2\}$

$\delta(q_0, a) = q_1$

$\delta(q_2, a) = q_3$

← Belongs to different sets

$\therefore$ q_0 and q_2 are not equivalent

Continuing in this way we get -

$$\boxed{1 \text{ - equivalence } = \{q_0, q_1, q_5, q_6\}, \{q_2, q_4\}, \{q_3\}, \{q_7\}}$$

Step 2 : Now will find 2 - equivalence from the sets obtained in 1 - equivalence.

Consider the set $\{q_0, q_1, q_5, q_6\}$ first. We will compare q_0 with q_1, q_5, q_6. Then we find

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

Belong to same set

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

Belong to different sets

$\therefore$ q_0 and q_1 now does not belong to same set.

Now campare states q_0, q_5 states.

$\delta(q_0, a) = q_1$
$\delta(q_5, a) = q_6$

$\delta(q_0, b) = q_0$
$\delta(q_5, b) = q_4$

Belongs to different sets

$\therefore$ q_0, q_5 are not equivalent. Now we will compare q_1 and q_5

$\delta(q_1, a) = q_0$
$\delta(q_5, a) = q_6$

Belong to same set

$\delta(q_1, b) = q_2$
$\delta(q_5, b) = q_4$

Belong to same set

That means - q_1 and q_5 are equivalent. Hence the 2 - equivalence set can be written as -

$$\boxed{2\text{ - equivalence } = \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}, \{q_3\}, \{q_7\}}$$

Step 3 : We will compare $\{q_0, q_6\}$

$\delta(q_0, a) = q_1$
$\delta(q_6, a) = q_5$

Belong to same set

$\delta(q_0, b) = q_0$
$\delta(q_6, b) = q_6$

Belong to same set

$\therefore$ $\{q_0, q_6\}$ are equivalent. Similarly, we will compare $\{q_1, q_5\}$, $\{q_2, q_4\}$ and we find them as equivalent. Hence 3 - equivalence set is as follows -

$$\boxed{3\text{ - equivalence } = \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}, \{q_3\}, \{q_7\}}$$

As 2 - equivalence and 3 - equivalence are same sets. We will stop finding further equivalences.

Hence we can have minimized transition table and minimized DFA as follows -

	State \ Input	a	b
→	$[q_0, q_6]$	$[q_1, q_5]$	$[q_0, q_6]$
	$[q_1, q_5]$	$[q_0, q_6]$	$[q_2, q_4]$
	$[q_2, q_4]$	$[q_3]$	$[q_1, q_5]$
*	$[q_3]$	$[q_3]$	$[q_0, q_6]$
	$[q_7]$	$[q_0, q_6]$	$[q_3]$

The transition diagram with minimized states is -

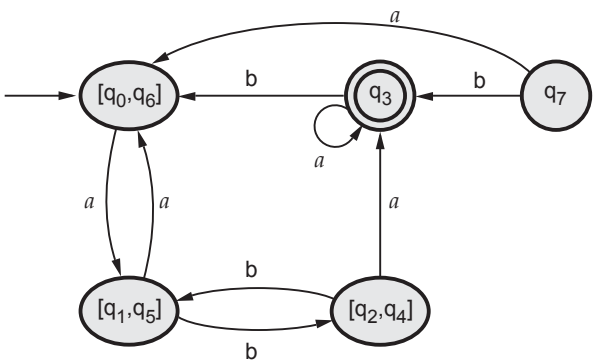


Fig. 1.15.2

Part III : FA with Output

1.16 Moore and Mealy Machines

SPPU : May-10,11,13,14,15, Dec.-06,11,13,14,19, Oct.-16, Marks 10

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

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

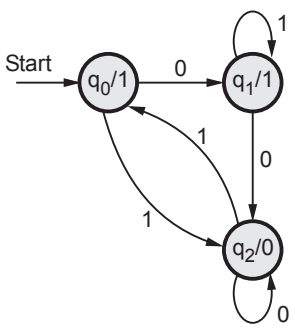


Fig. 1.16.1

Consider the Moore machine given below -

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.

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,

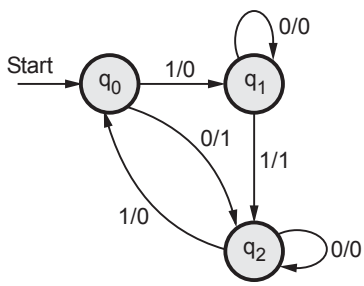


Fig. 1.16.2

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.

Difference between Moore and Mealy Machine

1) Length of Moore machine is one longer than Mealy machine for given input.
2) Secondly output of the input will be along the edges in case of Mealy machine but it should be associated with the state in case of Moore machine.

Problems Based on Moore and Melay Machine Modeling

Example 1.16.1 *Design a Moore machine to generate 1's complement of given binary number.*

SPPU : May-10, May-15, End Sem, Marks 6

Solution :

- **Logic**

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.

- **Design**

Hence the Moore machine will be as shown in Fig. 1.16.3.

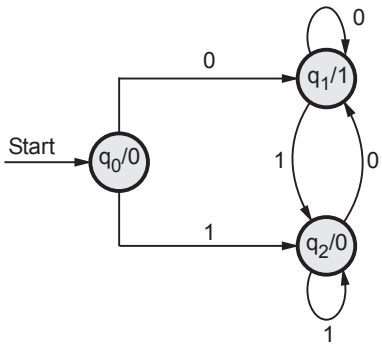


Fig. 1.16.3

- **Simulation**

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 (δ) 0	Next state (δ) 1	Output (λ)
→ 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 1.16.2 *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.* **SPPU : Dec.-06, May-07, Marks 10**

Solution :

- **Logic**

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 follows.

- **Design**

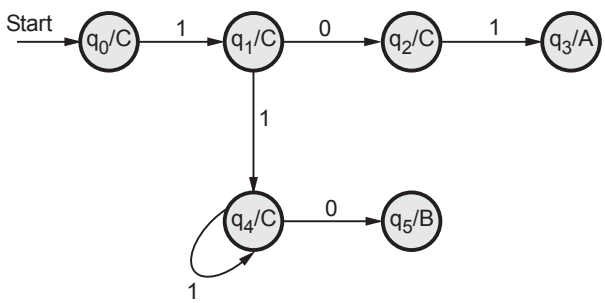


Fig. 1.16.4

Now we will insert the possibilities of 1's and 0's for each state. Then the Moore machine becomes.

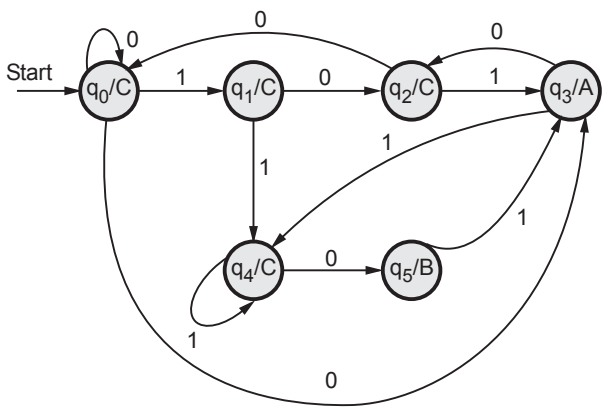


Fig. 1.16.5

Now the Mealy machine can be

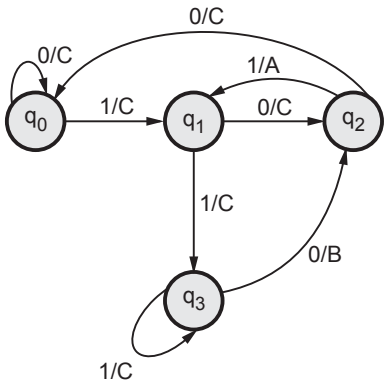


Fig. 1.16.6

Example 1.16.3 *Design a Mealy machine to find 2's complement of a given binary number.*

SPPU : Dec.-11, Dec.-19, End Sem, Marks 6

Solution :

- **Logic**

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

$\longleftarrow$

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 -

- **Design**

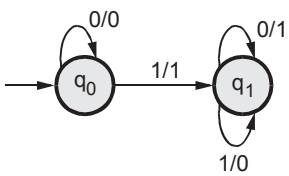


Fig. 1.16.7

Example 1.16.4 *Design a Moore machine which will increment the given binary number by 1.*

Solution :

- **Logic**

We will read the binary number form 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 built a mealy machine

- **Design**

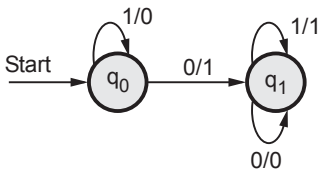


Fig. 1.16.8

Example 1.16.5 *Give Moore and Mealy machine for $\Sigma = \{0, 1, 2\}$, print the residue modulo 5 of input treated as a ternary number.*

Solution :

- **Logic**

The ternary number is made up of 0,1 and 2. The interpretation of ternary number n can be -

i) If we write 0 after n then number becomes 3n.

ii) If we write 1 after n then number becomes 3n + 1.

iii) If we write 2 after n then number becomes 3n + 2.

For example,

If n = 4 then its value is $4 \times 3^0 = 4$.

If we write 0 after 4 i.e.

$$40 = 4 \times 3^1 + 0 \times 3^0 = 12 \qquad \text{i.e. } (3 \times 4)$$

If we write 1 after 4 then

$$41 = 4 \times 3^1 + 1 \times 3^0 = 13 \qquad \text{i.e. } 3n + 1$$

If we write 2 after 4 we get

$$42 = 4 \times 3^1 + 2 \times 3^0 = 14 \qquad \text{i.e. } 3n + 2$$

For residue modulo 5 we will get remainder 0, remainder 1, remainder 2, remainder 3 and remainder 4 values. Then we assume various states for these remainders as -

q_0 - remainder 0 state

q_1 - remainder 1 state

q_2 - remainder 2 state

q_3 - remainder 3 state

q_4 - remainder 4 state

Now consider n = 4,

For 4.0 we get decimal value 12 that means 12 % 5 = 2 it gives remainder 2.

For 4.1 we get decimal value 13 that means 13 % 5 = 3 it gives remainder 3.

Similarly 4.2 given remainder 4.

n = 4 itself is remainder 4. Hence we can design,

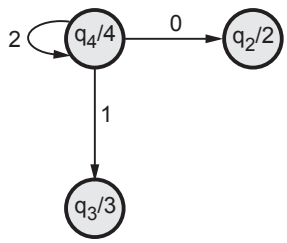


Fig. 1.16.9

- **Design**

Considering all possible cases we can design Moore machine as,

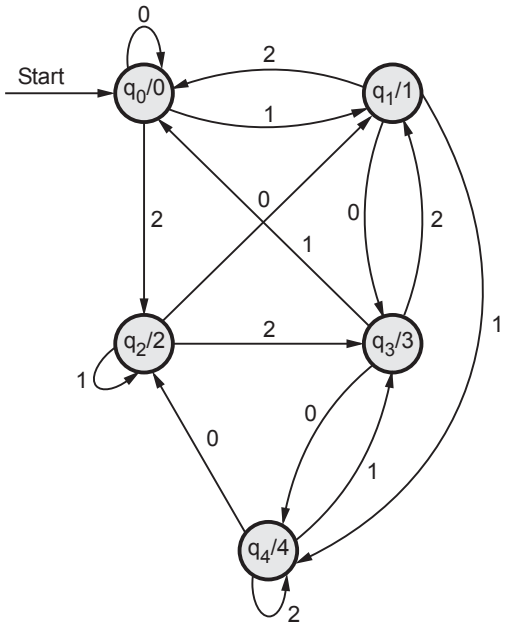


Fig. 1.16.10

Similarly the Mealy machine can be -

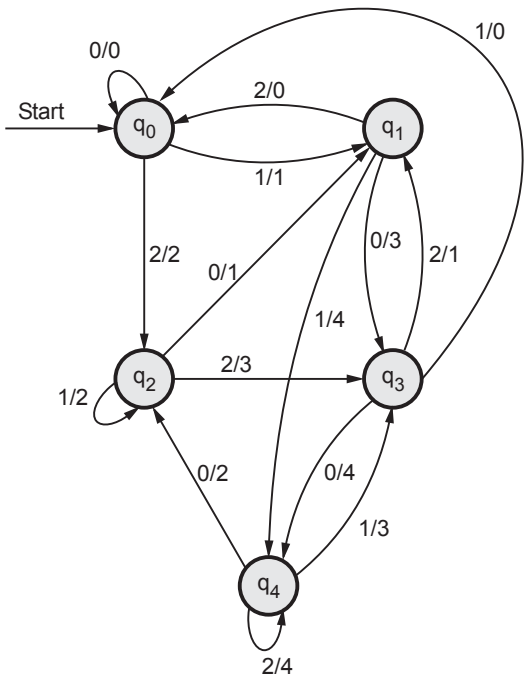


Fig. 1.16.11

Example 1.16.6 *Design a Moore machine that will read sequences made up of letters A, E, I, O, U and will give as output the same sequences except that in case where an I directly follows an E, it will be changed to U. Design the Mealy machine for the same.*

SPPU : May-13, Marks 8

Solution :

- **Logic**

We will assume a separate state for each alphabet. The output of that state will be corresponding letter. For instance

For alphabet A the state will be q_0 and output of q_0 will be A. For alphabet E the state will be q_1 and output of q_1 will be E. Continuing in this fashion we will have q_0, q_1, q_2, q_3 and q_4 states. The start state will be q_0 we will take care of one thing and that is when I comes immediately after E it will lead to the state q_4 because output of state q_4 is u. With this logic the corresponding Moore machine will be -

- **Design**

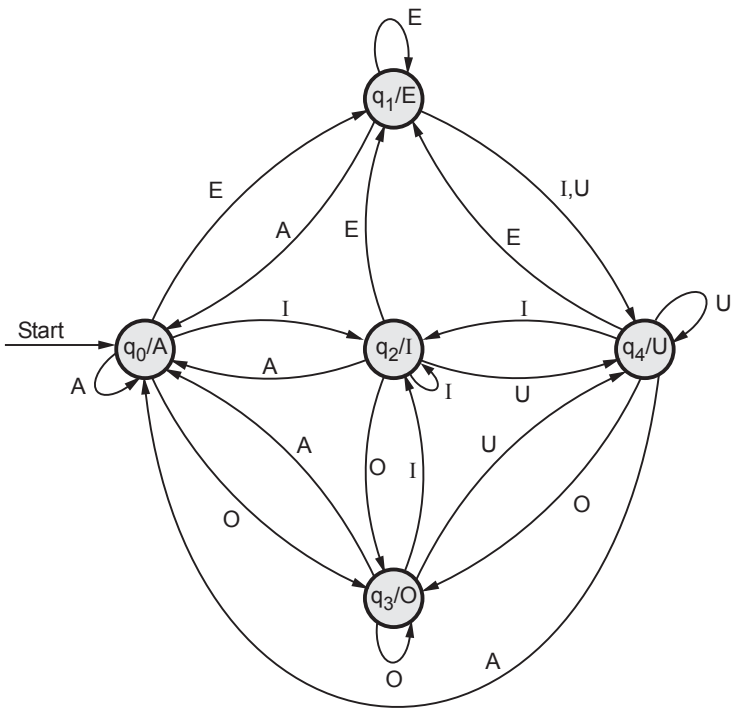


Fig. 1.16.12

Now we will draw the Mealy machine for the same problem.

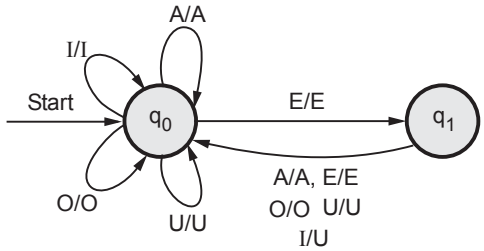


Fig. 1.16.13

Here the Mealy machine has the output along the edge itself we have got only two state. The transition changes from q_0 to q_1 when we get E. The I which is immediately following E will have the output U. Consider the string AIEOAEI will be give an output as AIEOAEU.

Example 1.16.7 *Construct a Moore machine to determine residue mod 3 for binary number.*

Solution :

- **Logic**

This Moore machine is also called remainder 3 tester. In this machine we will get remainder 0, remainder 1 and remainder 2. To interpret the given binary number in its decimal value we consider n as a number if 0 is written after n then its value becomes 2n. If 1 is written after n then it's value becomes 2n + 1. For instance, if n = 0 then its decimal value is 0 then,

$$01 = 2n + 1 = 1 \times 0 + 1 = 1$$

$$011 = 2n + 1 = (1 \times 2) + 1 = 3$$

Consider n = 1 its decimal value is 1. After 1 if 0 comes then its value will be

$$10 = 2n = 2 \times 1 = 2$$

If 1 comes after 10 then its value becomes,

$$101 = 2n + 1 = (2 \times 2) + 1 = 5$$

- **Design**

With this logic we can construct a Moore machine with 3 states. q_0 is the start state and is considered as remainder 0 state. q_1 is considered to be remainder 1 state and q_2 is considered as remainder 2 state.

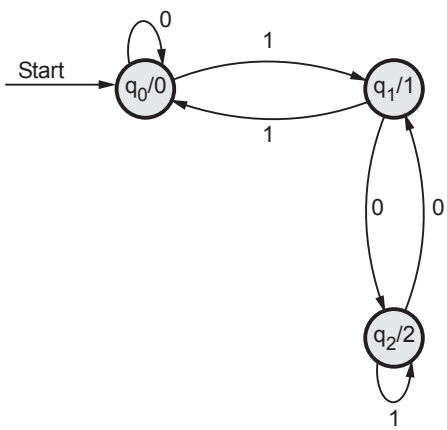


Fig. 1.16.14

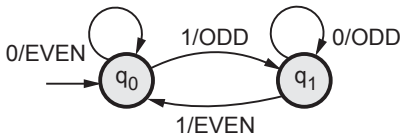
Example 1.16.8 *Define Mealy machine. Construct a Mealy machine which can output EVEN/ODD if the total number of 1′s in the input is even or odd. The input symbols are – 0 and 1.* **SPPU : Dec-13, Marks 5**

Solution :

- **Logic**

The restriction is on number of 1's. There is no restriction on number of 0's. The Mealy machine will then be -

- **Design**



Example 1.16.9 *Give Mealy machine for the following processes "For input from (0+1)*, if input ends in 101, output X; if input ends in 110, output Y, otherwise output Z".* **SPPU : May-11, 14, Marks 8**

Solution :

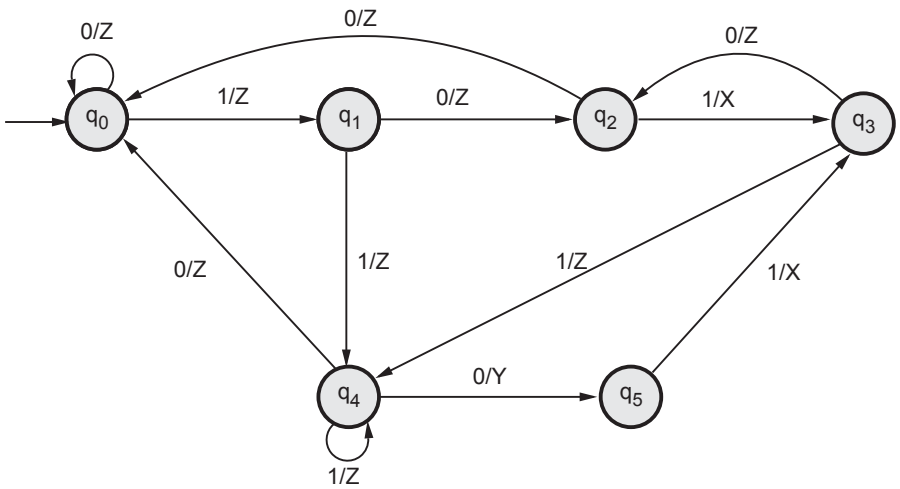


Fig. 1.16.15 Mealy machine

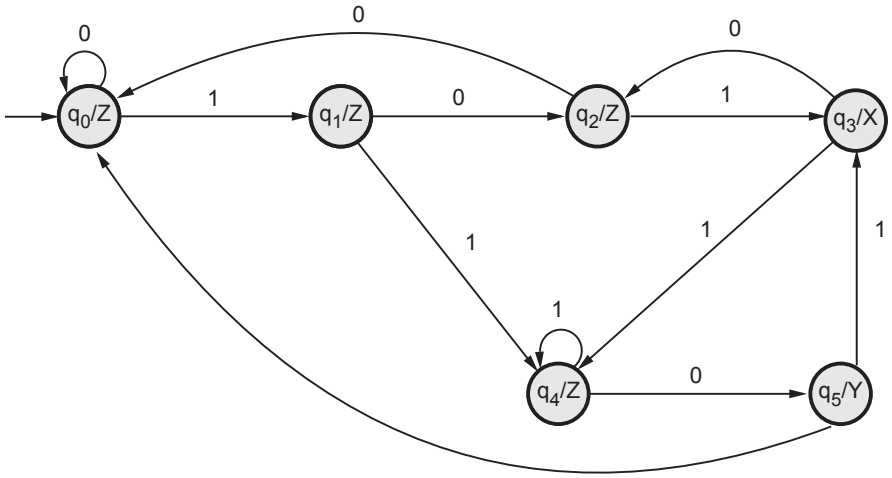
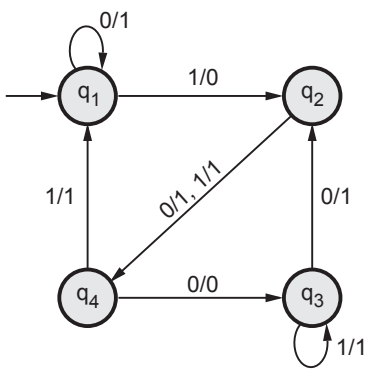


Fig. 1.16.16 Moore machine

Example 1.16.10 *Construct a Moore machine equivalent to the Melay machine M defined in the following table with input alphabet,*
$\Sigma = \{0, 1\}$. **SPPU : Dec.-14, In Sem, Marks 6**

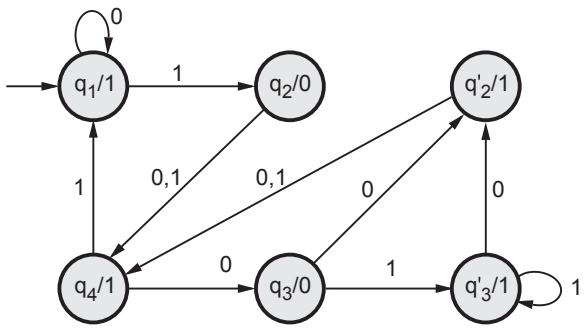
Present state	*Next state*			
	$\Sigma = 0$		$\Sigma = 1$	
	State	*Output*	*State*	*Output*
-> q_1	q_1	1	q_2	0
q_2	q_4	1	q_4	1
q_3	q_2	1	q_3	1
q_4	q_3	0	q_1	1

Solution : The Melay machine can be drawn as follows -



From this machine we can observe that the edges leading to q_1 and q_4 state have the output 1.

But the state q_3 has output 0 and 1. Similarly edges leading to state q_2 have output 0 and 1. Hence we will split q_2 as q_2 and q'_2. Similarly we will split q_3 as q_3 and q'_3. The Moore machine is as follows -



Example 1.16.11 *Design a Moore machine for computing the 2's complement of binary number. Convert it into its equivalent Mealy machine.* **SPPU : Oct.-16, In Sem, Marks 6**

Solution :

- **Logic Applied :** 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 and then change remaining 1's by 0's and 0's by 1's.
- **Example :** Let the binary number be

$$\underset{\substack{\leftarrow \\ \text{Read from LSB}}}{1\,0\,1\,1}$$

keep first 1 as it is and then toggle the bits. Hence we get 0101 ← This 2's complemented number.

The Moore machine will be -

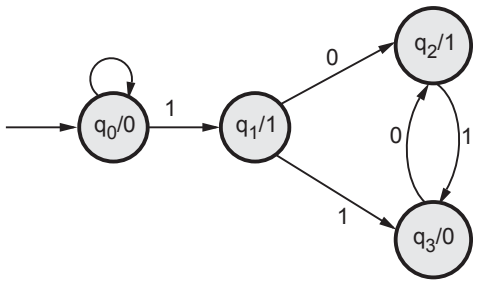


Fig. 1.16.17

- **Equivalent Mealy machine** - Refer ex. 1.16.3.

1.17 Inter-Conversion

SPPU : Dec.-05,11,12,18, May-06,07,12,14,19, Aug.-15,17, Oct.-19, Marks 8

1) **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 -

$$\lambda'(q, a) = \lambda(\delta(q, a))$$

2) **Method For conversion of Mealy Machine to Moore Machine**

Let $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ 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 $q_0, q_1, q_2, \ldots\ldots q_n$ on input $a_0, a_1, \ldots\ldots a_n$ and should emit the outputs $b_1, b_2, b_3, \ldots\ldots b_n$.

Problems Based on Inter-Conversion

Example 1.17.1 *The Moore machine to determine residue mod 3 for binary number is given below. Convert it to Mealy equivalent machine.*

Q \ Σ	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 -

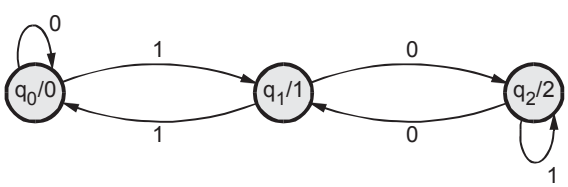


Fig. 1.17.1

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.

$$\lambda'(q_0, 0) = \lambda(\delta(q_0, 0))$$

$$= \lambda(q_0) \quad \text{i.e. output of } q_0$$

$$\boxed{\lambda'(q_0, 0) = 0}$$

$$\lambda'(q_0, 1) = \lambda(\delta(q_0, 1))$$

$$= \lambda(q_1) \quad \text{i.e. output of } q_1$$

$$\boxed{\lambda'(q_0, 1) = 1}$$

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

$$= \lambda(q_2)$$

$$\boxed{\lambda'(q_1, 1) = 2}$$

$$\lambda'(q_1, 1) = \lambda(\delta(q_1, 1))$$

$$= \lambda(q_0)$$

$$\boxed{\lambda'(q_1, 1) = 0}$$

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

$$= \lambda(q_1)$$

$$\boxed{\lambda'(q_2, 0) = 1}$$

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

$$= \lambda(q_2)$$

$$\boxed{\lambda'(q_2, 1) = 2}$$

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,

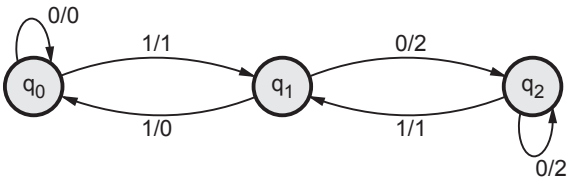


Fig. 1.17.2

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 1.17.2 *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 -

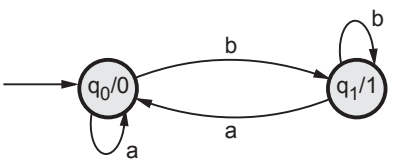


Fig. 1.17.3

$$\lambda'(q_0, a) = \lambda(\delta(q_0, a))$$
$$= \lambda(q_0)$$
$$= 0$$

$$\lambda'(q_0, b) = \lambda(\delta(q_0, b))$$
$$= \lambda(q_1)$$
$$= 1$$
$$\lambda'(q_1, a) = \lambda(\delta(q_1, a))$$
$$= \lambda(q_0)$$
$$= 0$$
$$\lambda'(q_1, b) = \lambda(\delta(q_1, b))$$
$$= \lambda(q_1)$$
$$= 1$$

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,

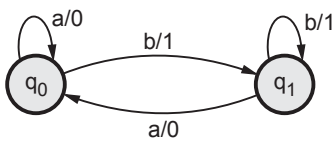


Fig. 1.17.4

Consider the output sequence for Moore machine for input sequence,

a b b a

Then

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

0 0 1 1 0

Similarly output sequence for Mealy machine will be

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

0 1 1 1 0

We can note that length of Moore machine is n + 1 and that of Mealy machine is n.

Example 1.17.3 *Consider the Moore machine described by the transition table given below. Construct corresponding Mealy machine.* **SPPU : Dec.-05, Marks 4; May-12, Marks 8**

Present state	*Next state*		*Output*
	a = 0	*a = 1*	
$\rightarrow q_1$	q_1	q_2	0
q_2	q_1	q_3	0
q_3	q_1	q_3	1

Solution : The transition diagram for given Moore machine will be -

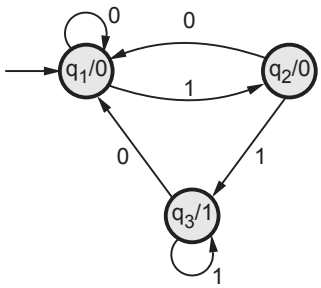


Fig. 1.17.5

The equivalent Mealy machine will be -

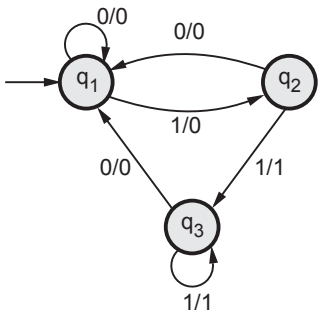


Fig. 1.17.6

Example 1.17.4 *Construct a Mealy Machine which is equivalent to the Moore machine given in the following table :* **SPPU : Aug.-17, In Sem, Marks 3**

Present state	*Next state*		*Output*
	a = 0	*a = 1*	
$\rightarrow q_0$	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

Solution : The transition diagram for the given problem can be drawn as

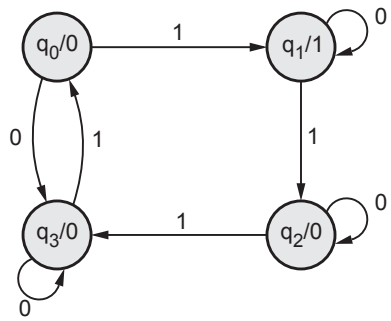


Fig. 1.17.7

The output function λ' can be obtained using following rule,

$$\lambda'(q, 0) = \lambda(\delta(q, 0))$$

Hence we will obtain output function λ' for each transition corresponding to input symbol.

$$\lambda'(q_0, 0) = \lambda(\delta(q_0, 0))$$

$$= \lambda(q_3)$$

$$\boldsymbol{\lambda'(q_0, 0) = 0}$$

$$\lambda'(q_0, 1) = \lambda(\delta(q_0, 1))$$

$$= \lambda(q_1)$$

$$\lambda'(q_0, 1) = 1$$

Thus continuing in this fashion we get following mealy machine.

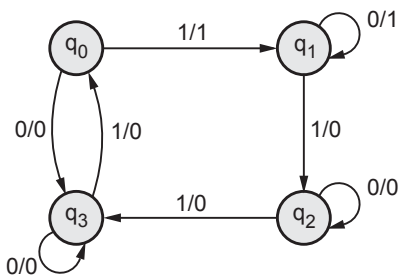


Fig. 1.17.8

Method for conversion of Mealy machine to Moore machine **SPPU : May-06, 10**

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 $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ 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 $q_0, q_1, q_2, ...q_n$ on input $a_0, a_1, ...a_n$ and should emit the outputs $b_1, b_2, b_3, ...b_n$.

Let us understand this method of conversion with the help of some examples.

Example 1.17.5 *Convert the following Mealy machine into equivalent Moore machine.*

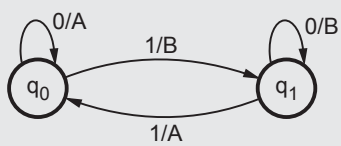


Fig. 1.17.9

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 -

$$\delta'((q_0, A), 0) = [\delta(q_0, 0), \lambda(q_0, 0)]$$

$$\text{i.e } [\underbrace{\text{next state of } q_0 \text{ on input } 0}_{q_0}, \underbrace{\text{output of } q_0 \text{ on input } 0}_{A}]$$

$$= [q_0, A]$$

$$\lambda'([q_0, A]) = A \rightarrow \text{ i.e. output of state } q_0.$$

$$\delta'([q_1, A], 0) = [\delta(q_1, 0), \lambda(q_1, 0)]$$

$$= [q_1, B]$$

$$\lambda'([q_1, A]) = A$$

$$\delta'([q_1, A], 1) = [\delta(q_1, 1), \lambda(q_1, 1)]$$

$$= [q_0, A]$$

$$\lambda'([q_1, A]) = A$$

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

$$\delta'([q_1, B], 0) = [\delta(q_1, 0), \lambda(q_1, 0)]$$

$$= [q_1, B]$$

$$\lambda'([q_1, B]) = B$$

$$\delta'((q_1, B), 1) = [\delta(q_1, 1), \lambda(q_1, 1)]$$

$$= [q_0, B]$$

$$\lambda'([q_1, B]) = B$$

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 -

$$\delta'([q_0, B], 0) = [\delta(q_0, 0), \lambda(q_0, 0)]$$

$$= [q_0, A]$$

$$\lambda'[q_0, B]) = B$$

$$\delta'((q_1, B), 1) = [\delta(q_0, 1), \lambda(q_0, 1)]$$

$$= [q_1, B]$$

$$\lambda'([q_0, B]) = B$$

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,

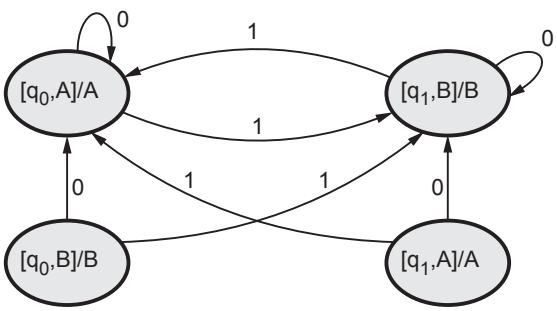


Fig. 1.17.10

Example 1.17.6 *Convert the following Mealy machine into equivalent Moore machine.*

SPPU : Dec.-18, In Sem, Marks 3

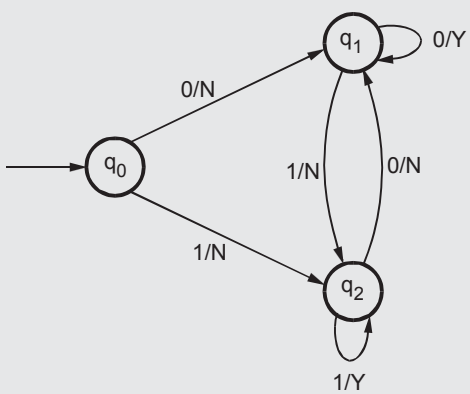


Fig. 1.17.11

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 -

$$\delta'([q_0, N)], 0) = [\delta(q_0, 0), \lambda(q_0, 0)]$$

$$= [q_1, N]$$

$$\lambda'([q_0, N]) = N$$

Similarly,

$$\delta'([q_0, N], 1) = [\delta(q_0, 1), \lambda(q_0, 1)]$$

$$= [q_2, N]$$

$$\lambda'([q_0, N]) = N$$

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 -

$$\delta'([q_0, Y], 0) = [\delta(q_0, 0), \lambda(q_0, 0)]$$

$$= [q_1, N]N$$

$$\lambda'([q_0, Y]) = Y$$

$$\delta'([q_0, Y], 1) = [\delta(q_0, 1), \lambda(q_0, 1)]$$

$$= [q_2 N]$$

$$\lambda'([q_0, Y]) = Y$$

$$\delta'([q_1 N], 0) = [\delta(q_1, 0), \lambda(q_1, 0)]$$

$$= [q_1 Y]$$

$$\lambda'([q_1, N]) = N$$

$$\delta'([q_1, N], 1) = [\delta(q_1, 1), \lambda(q_1, 1)]$$

$$= [q_2, N]$$

$$\lambda'([q_1, N]) = N$$

$$\delta'([q_1, Y], 0) = [\delta(q_1, 0), \lambda(q_1, 0)]$$

$$= [q_1, Y]$$

$$\lambda'([q_1, Y]) = Y$$

$$\delta'([q_2, N], 0) = [\delta(q_2, 0), \lambda(q_2, 0)]$$

$$= [q_1, N]$$

$$\lambda'([q_2, N]) = N$$

$$\delta'([q_2, N], 1) = [\delta(q_2, 1), \lambda(q_2, 1)]$$

$$= [q_2, Y]$$

$$\lambda'([q_2, N]) = N$$

$$\delta'([q_1, Y], 1) = [\delta(q_1, 1), \lambda(q_1, 1)]$$

$$= [q_2, N]$$

$$\lambda'([q_1, Y]) = Y$$

$$\delta'([q_2, Y], 0) = [\delta(q_2, 0), \lambda(q_2, 0)]$$

$$= [q_1, N]$$

$$\lambda'([q_2, Y]) = Y$$

$$\delta'([q_2, Y], 1) = [\delta(q_2, 1), \lambda(q_2, 1)]$$

$$= [q_2, Y]$$

$$\lambda'([q_2, Y]) = Y$$

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 1.17.7 *Construct a Moore machine equivalent to the Mealy machine M given below. δ is*

SPPU : Dec.-11, Marks 6

	$a = 0$		$a = 1$	
q_1	q_1	1	q_2	0
q_2	q_4	1	q_4	1
q_3	q_2	1	q_3	1
q_4	q_3	0	q_1	1

Solution : We will first design the transition diagram from given transition table.

The transition table for Moore machine will be

	0	1	Output
$[q_1]$	$[q_1]$	$[q_2]$	1
$[q_2]$	$[q_4]$	$[q_4]$	0 and 1
$[q_3]$	$[q_2]$	$[q_3]$	0 and 1
$[q_4]$	$[q_3]$	$[q_1]$	1

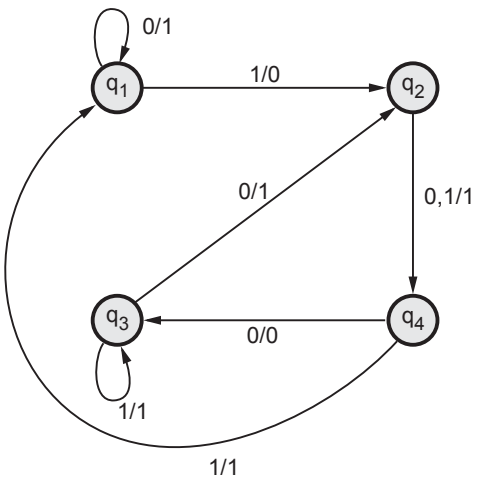


Fig. 1.17.12

The Moore machine will be

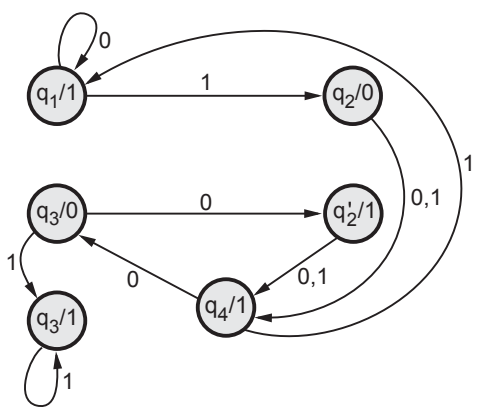


Fig. 1.17.13

Example 1.17.8 *Construct Mealy machine equivalent to Moore machine given as :*

SPPU : Dec.-12, Marks 6

Present state	*Next state*		*Output*
	a = 0	*a = 1*	
$\rightarrow q_0$	q_1	q_2	1
q_1	q_3	q_2	0
q_2	q_2	q_1	1
q_3	q_0	q_3	1

Solution : Equivalent Mealy machine can be obtained as follows :

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

$$\lambda'(q_0, 1) = \lambda(\delta(q_0, 1)) = \lambda(q_2) = 1$$

$$\lambda'(q_1, 0) = \lambda(\delta(q_1, 0)) = \lambda(q_3) = 1$$

$$\lambda'(q_1, 1) = \lambda(\delta(q_1, 1)) = \lambda(q_2) = 1$$

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

$$\lambda'(q_2, 1) = \lambda(\delta(q_2, 1)) = \lambda(q_1) = 0$$

$$\lambda'(q_3, 0) = \lambda(\delta(q_3, 0)) = \lambda(q_0) = 1$$

$$\lambda'(q_3, 1) = \lambda(\delta(q_3, 1)) = \lambda(q_3) = 1$$

Present state	Next state			
	a = 0	Output	a = 1	Output
$\rightarrow q_0$	q_1	0	q_2	1
q_1	q_3	1	q_2	1
q_2	q_2	1	q_1	0
q_3	q_0	1	q_3	1

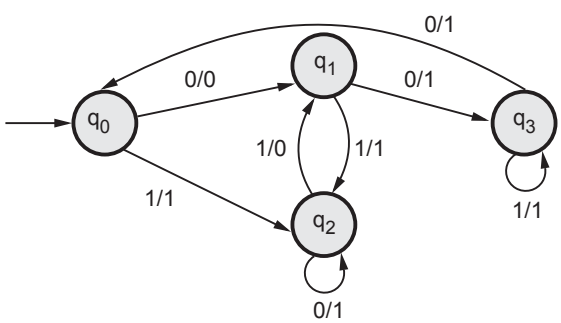


Fig. 1.17.14

Example 1.17.9 *Convert the following Mealy machine to Moore machine.*

SPPU : May-07, Marks 4

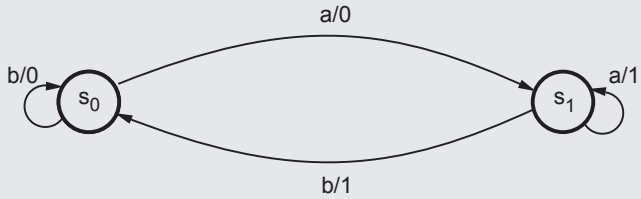


Fig. 1.17.15

Solution : If we observe state so then we notice that state so gets two incoming edges one for input b and output 0 and the other with input b and output 1. Similarly state S_1 gets two incoming edges with output 0 and output 1. Hence we split up the states as.

$S_0 \Rightarrow S_0$ and S_0' with output 0 and 1 respectively.

$S_1 \Rightarrow S_1$ and S_1' with output 0 and 1 respectively.

The Moore machine will then be -

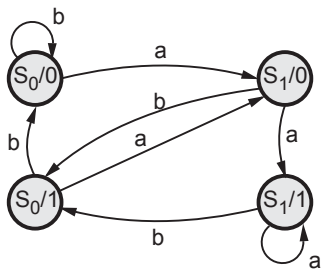


Fig. 1.17.16

Example 1.17.10 *Convert the following Mealy machine to More machine.*

SPPU : Oct.-19, In Sem, Marks 4

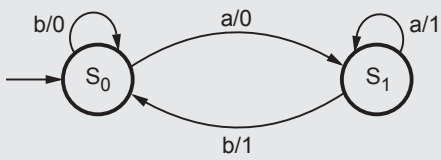


Fig. 1.17.17

Solution : The states for Moore machine will be $[S_0,0]$, $[S_0,1]$, $[S_1,0]$, $[S_1,1]$. Then we will be calculate δ' and λ' as follows :

1) $\delta'((S_0,0), a) = [\delta(S_0, a), \lambda(S_0, a)]$

$= [S_1, 0]$

$$\lambda'([S_0, 0]) = 0$$

2) $$\delta'((S_0,0), b) = [\delta(S_0, b), \lambda(S_0, b)] = [S_0, 0]$$

$$\lambda'([S_0, 0]) = 0$$

$$\delta'((S_1,0), a) = [\delta(S_1, a), \lambda(S_1, a)] = [S_1, 1]$$

$$\lambda'([S_1, 0]) = 0$$

3) $$\delta'((S_1,0), b) = [\delta(S_1, b), \lambda(S_1, b)] = [S_0, 1]$$

$$\lambda'([S_1, 0]) = 0$$

4) $$\delta'((S_0,1), a) = [\delta(S_0, a), \lambda(S_0, a)] = [S_1, 0]$$

$$\lambda'([S_0, 1]) = 1$$

5) $$\delta'((S_0,1), b) = [\delta(S_0, b), \lambda(S_0, b)] = [S_0, 0]$$

$$\lambda'([S_0, 1]) = 1$$

6) $$\delta'([S_1,1], a) = [\delta(S_1, a), \lambda(S_1, a)] = [S_1, 1]$$

$$\lambda'([S_1, 1]) = 1$$

7) $$\delta'([S_1,1], b) = [\delta(S_1, b), \lambda(S_1, b)] = [S_0, 1]$$

$$\lambda'([S_1, 1]) = 1$$

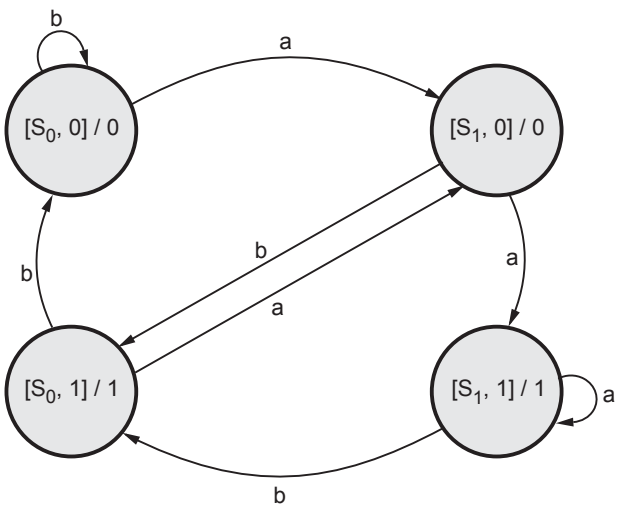


Fig. 1.7.17 (a)

The Moore machine will be :

Review Questions

1. *Define Moore and Mealy machine and how the equivalence of Moore and Mealy machine can be established with example.* **SPPU : May-14, Marks 6**
2. *Give formal definition for a Moore machine with a suitable example* **SPPU : Aug.-15, In Sem, May-19, End Sem, Marks 2**
3. *Compare Moore and Mealy machine.* **SPPU : Dec.-18, In Sem, Marks 3, Oct.-19, In Sem, Marks 2**

1.18 Case Study

FSM for Vending Machine

The states of coffee vending machine can be represented using finite State machine. Here is an illustrative vending machine with following assumptions

- When ₹ 15 are deposited then only release coffee.
- There will be single coin slot for ₹ 10 and ₹ 5

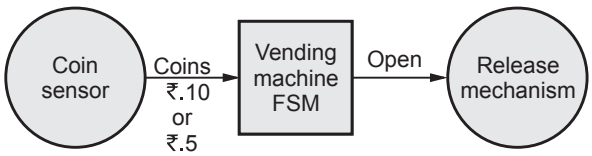


Fig. 1.18.1 Abstract model for vending machine

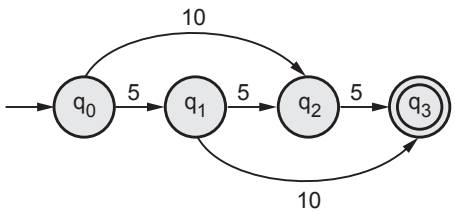
- No Change

The abstract representation of the machine will be

There are various cases that will lead to the state of release mechanism.

Case 1 : 3 coins of ₹ 5

Case 2 : 2 coins - firstly ₹ 5 coin and then ₹ 10 coin.



Case 3 : 2 coins - firstly ₹ 10 coin and then ₹ 5 coin.

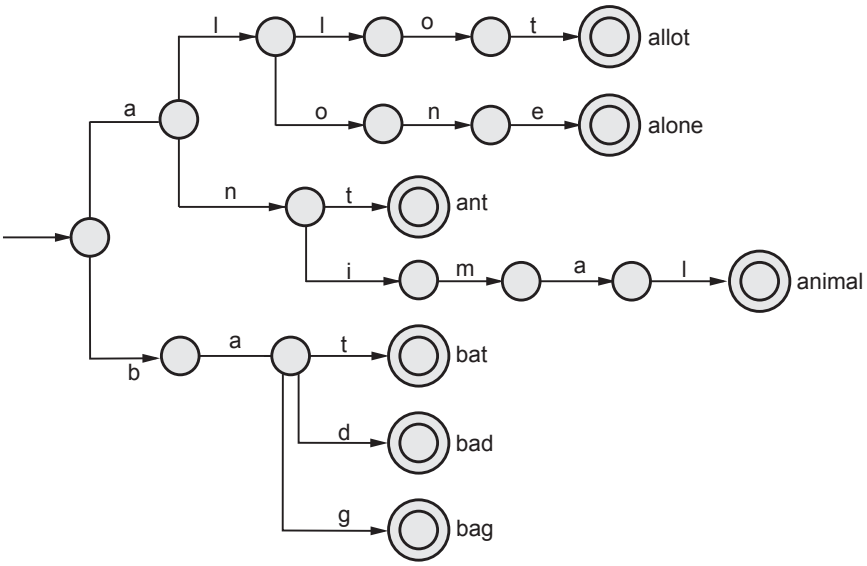
The FSM can be as follows

Her q_0 is the initial state with no coin inserted. The value on each of edge of FSM represents the value of coin. The state q_3 is a final state. When machine reaches to this state, the coffee will be released from the vending machine.

FSM in Spell Check

- Finite State machines can be used for simple spell checking.
- In this implementation, one or more words are provided by the user and those words would either be confirmed or rejected as being valid in that language.
- At the basic level, the finite state automata is generated for each of the valid string in the language.
- The common prefixes or common suffixes are used to recognize the words. For example

The use of finite state machine improves the performance of pattern matching procedure.



All the final states in above FSM indicate the correct spellings and all the intermediate states represent incomplete or incorrect spelling of a particular word.

1.19 Fill in the Blanks

Q.1 DFA stands for _____.

Q.2 The language generated by DFA is _____ .

Q.3 Power of DFA and NFA is ____ .

Q.4 The ___ character is used to represent empty null string in non deterministic finite automata.

Q.5 $\{L = a^P | P \text{ is prime}\}$ is a ___ language.

Q.6 Finite automata can recognize only one type of language and that is ____ .

Q.7 Can DFA simulate NFA ?

Q.8 How should be the recognizing capacity of NFA and DFA same or different ?

Q.9 An automation is ___device and grammar is a ___ device.

Q.10 Finite state machine ___ recognize panlindrome (can/can't).

Q.11 Removing __ states from a machine M does not affect the language it accepts.

Q.12 Let maximum number of states in DFA=128, then its equivalent NFA has ___ states.

Hint : The maximum number of states in DFA= 2^{n+1} where n is number of states in NFA.

1.20 Multiple Choice Questions

Q.1 For a DFA accepting binary numbers whose decimal equivalent is divisible by 4, what are all the possible remainders ?

a 0 b 0,2

c 0,2,4 d 0,1,2,3

Q.2 The number of states to accept string "ba" are _____.

a 1 b 2

c 3 d 4

Q.3 Language of finite automata is_____.

a Type 0 b Type 1

c Type 2 d Type 3

Q.4 There are ___ number of stacks used in finite automata.

a 0 b 1

c 2 d 3

Q.5 Number of final state require to accept ϕ in minimal finite automata.

a 1 b 2

c 3 d No final state is required

Q.6 How many DFA's exits with two states over input alphabet {a,b} ?

a 2 b 16

c 64 d 32

Hint : Number of DFA's = 2n * n(2*n).

Q.7 An FSM with _____.

a one stack is powerful than two stack

b one stack is powerful than two stack

c both are of equal power

d none of these

Q.8 Every NFA accepts a _____.

a string b function

c regular language d context free language

Q.9 N = (Q, ε, δ, S, F) represents a _____.

a NFA b PDA

c LDA d Turing Machine

Q.10 An NFA's transition function returns _____.

a boolean value b state

c set of states d edge

Q.11 Any given transition graph has equivalent _____.

a regular expression b DFA

c NFA d all of these

Answer Keys for Fill in the Blanks :

Q.1	Deterministic Finite Automate	**Q.2**	Regular language
Q.3	same	**Q.4**	ε or ∧
Q.5	non regular	**Q.6**	Regular language
Q.7	Yes	**Q.8**	Same
Q.9	cognitive, generative	**Q.10**	can't
Q.11	dead or unreachable	**Q.12**	6

Answer Keys for Multiple Choice Questions :

Q.1	d	**Q.2**	c	**Q.3**	d	**Q.4**	a
Q.5	d	**Q.6**	c	**Q.7**	b	**Q.8**	c
Q.9	a	**Q.10**	c	**Q.11**	d		

❑❑❑

UNIT - II

2 Regular Expressions

Syllabus

Introduction, Operators of RE, Precedence of operators, Algebraic laws for RE, Language to Regular Expressions, Equivalence of two REs. ***Conversions****: RE to NFA, DFA, DFA to RE using Arden's theorem, Pumping Lemma for Regular languages, Closure and Decision properties of Regular languages. Myhill-Nerode theorem.*

Contents

2.1 Operators of RE . . . . . **Dec.-16, Oct.-19,** · · · · · · · · Marks 6

2.2 Language to Regular Expressions . . . . . **Dec.-07, 11, 12, 13, 14, 15,** . . . . . **May-08, 09, 10, 11, 13, 14, 16,** . . . . . **Aug.-15, Oct.-16, 17, 18, 19,** Marks 8

2.3 Precedence of Operators

2.4 Algebraic Laws for RE . . . . . **Aug.-17,** · · · · · · · · · · · · · · · Marks 3

2.5 Equivalence of Two Regular Expressions . . . . **May-06, 07, 09, 11, Dec.-08, 09, 11,** . . . . . **Oct.-16, Aug.-17,** · · · · · · · Marks 10

2.6 RE to DFA . . . . . **May-10, 12, 14, 15, 19, Dec.-11, 13, 14,** . . . . . **16, 18, 19, Aug.-15, 17, Oct.-18, 19,** . . . . . · · · · · · · · · · · · · · · · · · · Marks 10

2.7 DFA to RE using Arden's Theorem . . . . . **May-09, 10, 14, 15, 16, Dec.-08, 09, 10,** . . . . . **13, 14, 15, Aug.-15,** · · · · · · Marks 10

2.8 Pumping Lemma for Regular Languages. . . . . **Dec.-06, 08, 12, May-06, 07, 09, 12, 13,** . . . . . **14, 16, Aug.-17, Oct.-18,** · · · Marks 8

2.9 Closure Properties of Regular Languages . . . . **May-18, Oct 18,** · · · · · · · · · Marks 6

2.10 Decision Properties of Regular Languages

2.11 Limitation of FA . . . . . **Aug.-15,** · · · · · · · · · · · · · · · Marks 2

2.12 Myhill - Nerode Theorem

2.13 Case Study . . . . . **Dec.-05, 06, 10, 13, 14, May-13, 14,** . . . . . **Oct.-19,** · · · · · · · · · · · · · · · Marks 6

2.14 Fill in the Blanks

2.15 Multiple Choice Questions

2.1 Operators of RE

SPPU : Dec.-16, Oct.-19, Marks 6

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 $\{\varepsilon\}$ 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 = $\{\varepsilon, a, aa, aaa, aaaa, ...\}$

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 ^

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

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

We can construct L* as

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

Example 2.1.1 *What is kleen closure ? What is positive closure ? For a given language L under what circumstances will L^+ and L^* be equal ?* **SPPU : Dec.-16, End Sem, Marks 6**

Solution :

- The Kleen closure is a set of strings of any length (including null string ε).
- The positive closure Σ^+ can be defined as $\Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots$ That means positive closure consists of all strings of any length except null string.
- Basically $L^+ \subseteq L^*$ But $L^+ = L^*$ if and only if $\varepsilon \in L^+$. The $\varepsilon \in L^+$ if and only if $\varepsilon \leftarrow$ L. That mean ε is the shortest word in L.

Review Question

1. *Define the following terms : i) Kleen closure ii) Positive closure.*

SPPU : Oct.-19, In Sem, Marks 2

2.2 Language to Regular Expressions

SPPU : Dec.-07, 11, 12, 13, 14, 15, May-08, 09, 10, 11, 13, 14, 16, Aug.-15, Oct.-16, 17, 18, 19, Marks 8

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

Example 2.2.1 *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, tripple 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 {ε, *a*, *aa*, *aaa*, ...}. So we can give regular expression for this as

$$R = a^*$$

That is kleen closure of *a*.

Example 2.2.2 *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

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

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 2.2.3 *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 = {ε, *a*, *aa*, *a*b, b, b*a*, b*a*b, *ab*a*b*, ... any combination of *a* and b}.

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

Example 2.2.4 *Construct the regular expression for the language containing all strings having any number of a's and b's, except the null string.*

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

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

Example 2.2.5 *Construct the r.e. for the language accepting all the strings which are ending with 00 over the set Σ = {0, 1}.*

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

r.e. = (any combination of 0's and 1's) 00

i.e. r.e. = (0+1)* 00

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

Example 2.2.6 *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.

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 2.2.7 *If L = {The language starting and ending with a and having any combination of b's in between }, then what is r ?*

Solution : The regular expression

$r = a\,b^*\,a$

Example 2.2.8 *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*, *aba*, *abab*, *aab*, *aaa*, ...}

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

Example 2.2.9 *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 2.2.10 *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 2.2.11 *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) (any number of 0's and 1's)

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

Similarly,

L_2 = (any number of 0's and 1's) (00 + 11)

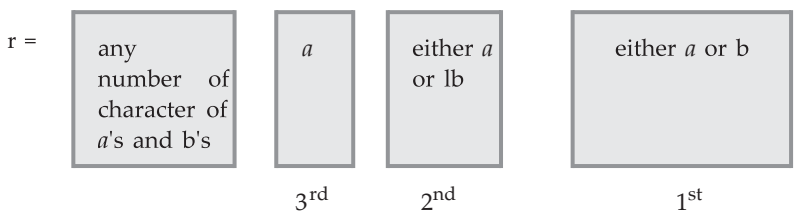
= (0 + 1)* (00 + 11)

Hence

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

Example 2.2.12 *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 b*a*bb or b*aaa* or b*aba* or *aaa*b and so on.

Example 2.2.13 *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 2.2.14 *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.

$$R = a^* \text{ b } a^* \text{ b } a^*$$

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 2.2.15 *Write r.e. which contains L having strings which should have atleast one 0 and atleast one 1.*

Solution : The required expression will be

$$R = \left[(0+1)^* 0\,(0+1)^* 1 (0+1)^*\right] + \left[(0+1)^* 1 (0+1)^* 0 (0+1)^*\right]$$

Example 2.2.16 *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 = {ε, 00, 0000, 000000, ...}.

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 2.2.17 *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, ...}. Note the difference between previous example and this one.

Example 2.2.18 *Describe the language denoted by following regular expression*
r.e. = $\left(b^*(aaa)^* b^*\right)^*$

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

r.e. = (any combination of b's) (*aaa*)* (any combination of b's).

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

Example 2.2.19 *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 = {b*aaa*, b*a*b*a*b*a*, *a*b*a*b*a*b, ...}

The regular expression can be

r.e. = (b* *a* b* *a* b* *a* b*)*

Example 2.2.20 *Write the r.e. to denote the language L over Σ = {a, b} such that all the strings do not contain the substring "ab".*

Solution : The L = {ε, *a*, b, bb, *aa*, b*a*, b*aa* ...}

The r.e. = (b* *a**)

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

Example 2.2.21 *Find all possible regular expression L ⊆ {a, b}* for*

a) The set of all strings ending in b

b) The set of all strings ending in ba

c) The set of all strings ending neither b nor ba. **SPPU : Dec.-07, Marks 6**

Solution :

a) The regular expression for language containing strings ending with b is

r.e. = (a + b)* b.

b) The regular expression for language containing strings ending with ba is

r.e. = (a + b) * ba

c) The regular expression for language containing strings ending with neither b nor a is

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

Example 2.2.22 *Let L be a language. It is clear from the definition that $L^+ \subseteq L^*$. Under what circumstances are they equal ?* **SPPU : Dec.-07, May-08, Marks 4**

Solution : The language L^+ denotes any combination of input symbols without ε.

The language L^* denotes any combination of input symbols along with ε.

Hence $L^+ \subseteq L^*$

But if $L_1 = \varepsilon + L^+$ then L_1 becomes equal to L^*. That means if ε is added as one symbol in the set generated by L^+ then L^* become equal.

Example 2.2.23 *Find all possible regular expression $L \subseteq \{a, b\}^*$ for*

a) The set of all strings ending in ab

b) The set of all strings ending in ba

c) The set of all stings ending neither ab nor ba. **SPPU : May-08, Marks 6**

Solution :

a) The regular expression is

r.e. = (a + b)* ab

b) The regular expression is

r.e. = (a + b)* ba

c) The regular expression is

(a + b)* aa + (a + b)* bb

Example 2.2.24 *Consider the two regular expressions $r = 0^* + 1^*$ $s = 01^* + 10^* + 1^*0 + (0^*1)^*$.*

a) Find the string corresponding to r but not to s

b) Find the string corresponding to s but not to r

c) Find the string corresponding to both r and f. **SPPU : May-10, Marks 8**

Solution :

a) The string corresponding to r but not to s : {00,000,0000, ...}

b) The string corresponding to s but not to r : {01 or 10}

c) The string corresponding to both r and f : {0, 1, 11, 111, 1111, 11111, ...}

Example 2.2.25 *Find a regular expression corresponding to each of the following subsets of {0, 1}**

1) The language of all strings containing exactly two 0's.

2) The language of all strings containing at least two 0's.

3) The expression of all strings that do not end with 01. **SPPU : May-09, Marks 6**

Solution :

1) If a language contains all strings containing exactly two 0's then, it can be at any position, may or may not be surrounded by 1's. Hence r.e. Will be
r.e. = 1* 0 1* 01*

2) If a language contains all strings containing at least two 0's, then it may be at any position, and may or may not be surrounded by (0+1)*. Hence r.e. will be
r.e. = (0+1)* 0 (0+1)* 0 (0+1)*

3) The language contains all the strings that do not end with 01 means it may end with either 0's or end with either 1's or end with 10. Hence r.e. will be -
r.e. = [(0+1)* 0*] + [(0+1)* (10)*]

Example 2.2.26 *Obtain the regular expressions for the following sets :*

1) $L1 = \{b^2, b^5, b^8, b^{11}, b^{14} \ldots\}$ *2)* $L2 = \{a^{2n+1} | n > 0\}$ **SPPU : May-11, Marks 4**

Solution :

1) $\mathbf{L1 = \{b^2, b^5, b^8, b^{11}, b^{14} \ldots\}}$

The regular expression for L1 = $\{b^2, b^5, b^8, b^{11}, \ldots\}$ is

Regular expression = bb (bbb)*

2) $\mathbf{L2 = \{a^{2n+1} | \ n > 0\}}$

The regular expression for L2 = $\{a^{2n+1} \ | \ n > 0\}$ is

Regular expression = $a\,(aa)^+$

Example 2.2.27 *Obtain in plain english language represented by following regular expression.*

a) $0^*(10^*10^*)^*1(0^*10^*1)^*0^*$ *b)* $0^*(0^*10^*1)^*0^*$ **SPPU : May-11, Marks 4**

Solution :

a) $\mathbf{0^*\,(10^*10^*)^*1\,(0^*10^*1)^*\,0^*}$:

L = {The language containing group of even number of 1's separated by single 1}

b) $\mathbf{0^*(0^*10^*1)^*\,0^*}$:

L = {The language containing even number of 1's and any number of 0's}

Example 2.2.28 *Represent following formal languages using regular expressions.*

1) All string's of a's and b's without any combination of double letters.

2) All string's of 0's and 1's with even number of 0's. 3) All string's of a's and b's containing at least two a's. **SPPU : May-11, Marks 6**

Solution :

1) **All string's of a's and b's without any combination of double letters :** $(01)^* + (10)^*$

2) **All string's of 0's and 1's with even number of 0's :** $(1^*01^*01^*)^*$

3) **All string's of a's and b's containing at least two a's :** $(b^*a\,b^*a\,b^*)^+$

Example 2.2.29 *Find the regular expressions representing the following sets :*

i) The set of all strings over {a, b} with three consecutive b's.

ii) The set of all strings over {0, 1} beginning with 00.

iii) The set of all strings over {0, 1} ending with 00 and beginning with 1.

SPPU : Dec.-11, Marks 6

Solution :

i) r.e. = $a^*(bbb)^* a^*$

ii) r.e. = $00\,(0+1)^*$

iii) r.e. = $1\,(0+1)^*\,00$

Example 2.2.30 *Find regular expressions representing the following sets :*

i) The set of all strings over {a, b} having atmost one pair of a's or atmost one pair of b's.

ii) The set of all strings over {a, b} in which the number of occurrences of a is divisible by 3.

iii) The set of all strings over {a, b} in which there are at least two occurrences of b between any two occurrences of a. **SPPU : Dec.-12, Marks 6**

Solution :

i) The set of all strings over {a, b} having atmost one pair of a's or atmost one pair of b's.

$((ab)+b)^*\, a\, a\, ((ba)+b)^* / ((ba)+a)^*\, bb\, ((ab)+a)^*$

ii) The set of all strings over {a, b} in which the number of occurrence of a is divisible by 3.

$\Rightarrow \quad b^*\,(aaa)^*\,b^*$

iii) The set of all strings over {a, b}, in which these are at least two occurrences of b between any two occureances of a.

$\Rightarrow \quad (a+b)^*\,(aa)\,(bb)^+\,(aa)\,(a+b)^*$

Example 2.2.31 *Write R.E. for the following*

i) Σ = (0, 1) odd number of 1's in strings

ii) Σ = (0, 1) triple 0 must never appear in string. **SPPU : May-14, Marks 6**

Solution :

i) r.e. = (0 * 10 *) (10 * 10 *) *

ii) r.e. = (1 + 01) * 00 (1 + 10) *

Example 2.2.32 *Give english description of the language of the following regular expression*

*(1 + ε) (00*1)* 0** **SPPU : Dec.-13, Marks 6**

Solution : This expression generates the strings as -

{ ε, 1, 101, 1001, 10, 100, 1010, …}

This regular expression gnerates all strings in which every 1 is separated by one or more 0. It also accepts the string which has only one 1 or the empty string.

Example 2.2.33 *Write regular expressions for the following languages over {0, 1}**

i) The set of strings that begin with 110

ii) The set of all strings not containing 101 as a substring. **SPPU : Dec.-13, Marks 6**

Solution :

i) r.e. = 110 (0 + 1) *

ii) r.e. = (0 + 11 * 00) * (ε + 11 * (ε + 0))

Example 2.2.34 *Write regular expressions for each of the following languages :*

i) For ∑ = {a, b}, set of all strings with no consecutive 'a's and 'b's.

ii) For ∑ = {0, 1}, set of all strings in which every 0 is immediately followed by at least two 1's.

iii) $L = \{1^{(2n+1)} \mid n \geq 1\}$ **SPPU : Dec.-14, In Sem, Marks 6**

Solution : i) [(ab)* + (ba)*] **ii)** [(011 + 1)*] **iii)** [1(11)*]

Example 2.2.35 *Describe the languages accepted by the following Regular Expressions and justify with suitable examples :*

i) $\mathbf{a}^* \cdot (\mathbf{a}+\mathbf{b})^* \cdot \mathbf{ab}$ *ii)* $(1^*.0.1^*.0.1^*)^*$ *iii)* $\mathbf{a}^*\mathbf{b}+\mathbf{b}^*\mathbf{a}$

SPPU : Aug.-15, In Sem, Marks 6

Solution :

i) The language containing the strings that begins with single a and end with 'ab'.

ii) The language containing the strings having even number of 0's.

iii) The language containing strings that either begin with any number of a's and end with single b or begin with any number of b's and end with single a.

Example 2.2.36 *Write regular expressions for the following languages over the alphabet ∑ = {a, b}*

i) All strings that do not end with 'aa'.

ii) All strings that contain an even number of 'b's.

iii) All strings which do not contain the substring 'ba'.

SPPU : Dec.-15, End Sem, Marks 6

Solution :

i) The language contains all the strings that do not end with aa means it may end with bb, ba or ab. Hence r. e. will be

(a + b) * ((ab)* + (ba)* + b*)

ii) r. e. = (a* b a* b)*

iii) The L = { ε, a, b, bb, aa, ab, abb, … }

r. e. = (a* b*)

Example 2.2.37 *Determine a regular expression over the alphabet {x, y} For the following :*

i) All strings containing exactly two x's.

ii) All strings that do not end with xy.

iii) All strings starting with yy. **SPPU : May-16, End Sem, Marks 6**

Solution :

i) The regular expression is

$$r.e = y^* \, xy^* \, xy^*$$

ii) The strings do not end with xy means string can end with x or y or with yx.

The r.e. will then be –

$$(x+y)^* \, yx + (x+y)^* \, x + y^*$$

iii) The regular expression will be

$$yy\,(x+y)^*$$

Example 2.2.38 *Give the regular expression for the following languages :*

i) $L = \{x \mid x \in \{a, b\}^*$ *and x contains exactly two a's}*

ii) L = {a, c, ab, cb, abb, cbb, abbb, cbbb, ...}

iii) $L = \{x \mid x \in \{a, b\}^*$ *and x is any string that begins in "abb" or a}*

SPPU : Oct.-16, In Sem, Marks 6

Solution :

i) $b^* \, ab^* \, ab^*$

ii) The set contains all the string that either begin with a or c but always end with any number of b's.

$$\therefore \quad r.e. = (a+c)\,b^*$$

iii) r.e. = (abb + a) $(a+b)^*$

Example 2.2.39 *Give the regular expression for the following languages.*

i) The set of strings over the alphabet {a,b} starting with b and ending with odd number of a's or even number of b's.

ii) The set {10, 1010} **SPPU : Oct.-18, In Sem, Marks 3**

Solution : i) r.e. = b(a(aa)* + (bb)*) **ii)** r.e. = $(10)^+$

Example 2.2.40 *Determine a regular expression over the alphabet Σ = {a,b}.*

i) All strings that contain an even number of 'b's

ii) All strings that do not end with 'aa' **SPPU : Oct.-19, In Sem, Marks 4**

Solution :

i) r.e. = $(a^* \; ba^* \; ba^*)^*$

ii) r.e. = $(a+b)^* (ab+ba+bb)$

Example 2.2.41 *Justify if true or false the following :*
Every subset of a regular language is regular **SPPU : Oct.-19, In Sem, Marks 3**

Solution : Every subset of regular language is regular is **False**

Justification :

Let, $a^* \; b^*$ be a regular expression. The subset of regular expression $a^* \; b^*$ is $a^n \; b^n$.

For $a^* \; b^*$ we can draw DFA but for $a^n \; b^n$ we can not draw DFA.

Hence, $a^n \; b^n$ is said to be non-regular expression.

This shows that subset of regular expression is non-regular expression.

Review Questions

1. *Define with example - Regular expression.* **SPPU : May-13, Marks 2**
2. *Define with suitable example - Regular expression and Operation.* **SPPU : Aug.-17, In Sem, Marks 3**

2.3 Precedence of Operators

There are three main operators that can be used in RE - **Union**, **concatenation** and **kleen closure**. For regular expressions, the following is the order of precedence for the operators :

i) The **star operator (kleen closure)** is having highest precedence. That means the r.e. with star operator must be grouped first.

ii) The next precedence is for **dot operator** i.e. for concatenation. The concatenation is an **associative** operator, hence the way we group the elements of regular expression does not matter much. For example 0110 can be grouped as (01)(10), or 0(110) or (011)0

iii) Final precedence is given to + **(i.e. Union) operation.** The union or + operator is also associative operator.

2.4 Algebraic Laws for RE

SPPU : Aug.-17, Marks 3

The two regular expressions P and Q are equivalent (denoted as P = Q) if and only if P represents the same set of strings as Q does. For showing this equivalence of regular expressions we need to show some identities of regular expressions.

Let P, Q and R are regular expressions then the identity rules are as given below.

1.	εR	=	$R\varepsilon = R$	
2.	ε^*	=	ε	ε is null string
3.	$(\phi)^*$	=	ε	ϕ is empty string.
4.	ϕR	=	$R\phi = \phi$	
5.	$\phi + R$	=	R	
6.	$R + R$	=	R	
7.	RR^*	=	$R^*R = R^+$	
8.	$(R^*)^*$	=	R^*	
9.	$\varepsilon + RR^*$	=	R^*	
10.	$(P + Q) R$	=	$PR + QR$	
11.	$(P + Q)^*$	=	$(P^* Q^*) = (P^* + Q^*)^*$	
12.	$R^* (\varepsilon + R)$	=	$(\varepsilon + R) R^* = R^*$	
13.	$(R + \varepsilon)^*$	=	R^*	
14.	$\varepsilon + R^*$	=	R^*	
15.	$(PQ)^* P$	=	$P (QP)^*$	
16.	$R^* R + R$	=	$R^* R$	

Review Question

1. *What are algebraic laws of regular expression.* **SPPU : Aug.-17, In Sem, Marks 3**

2.5 Equivalence of Two Regular Expressions

SPPU : May-06, 07, 09, 11, Dec.-08, 09, 11, Oct.-16, Aug.-17, Marks 10

Let us see one important theorem named **Arden's Theorem** which helps in checking the equivalence of two regular expressions.

Arden's Theorem : Let, P and Q be the two regular expressions over the input set Σ. The regular expression R is given as

$$R = Q + RP$$

Which has a unique solution as $R = QP^*$.

Proof : Let, P and Q are two regular expressions over the input string Σ.

If P does not contain ε then there exists R such that

$$R = Q + RP \quad \text{... (2.5.1)}$$

We will replace R by QP^* in equation (2.5.1)

Consider R.H.S. of equation (2.5.1)

$$= Q + Q P^* P$$

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

$$= QP^* \qquad \because \varepsilon + R^* R = R^*$$

Thus $\boxed{R = QP^*}$

is proved. To prove that $R = QP^*$ is a unique solution, we will now replace L.H.S. of equation 1 by Q + RP. Then it becomes

$$Q + RP$$

But again R can be replaced by Q + RP.

$$\therefore \quad Q + RP = Q + (Q + RP) P$$

$$= Q + QP + RP^2$$

Again replace R by Q + RP.

$$= Q + QP + (Q + RP) P^2$$

$$= Q + QP + QP^2 + RP^3$$

Thus if we go on replacing R by Q + RP then we get,

$$Q + RP = Q + QP + QP^2 + ... + QP^i + RP^{i+1}$$

$$= Q(\varepsilon + P + P^2 + ... P^i) + RP^{i+1}$$

From equation (2.5.1),

$$R = Q(\varepsilon + P + P^2 + ... + P^i) + RP^{i+1} \quad \text{... (2.5.2)}$$

Where $i \geq 0$

Consider equation (2.5.2),

$$R = Q\underbrace{(\varepsilon + P + P^2 + ... + P^i}_{P^*} + RP^{i+1}$$

$$\therefore \quad R = QP^* + RP^{i+1}$$

Let w be a string of length i.

In RP^{i+1} has no string of less than i + 1 length. Hence w is not in set RP^{i+1}. Hence R and QP^* represent the same set. Hence it is proved that

$$R = Q + RP \text{ has a unique solution.}$$

$$R = QP^*.$$

Problems for Equivalence of Two

Example 2.5.1 *Prove* $(1+00^*1)+(1+00^*1)\,(0+10^*1)^*\,(0+10^*1)=0^*1\,(0+10^*1)^*$

SPPU : Dec.-11, Marks 4

Solution : Let us solve L.H.S. first,

$$(1+00^*1)+(1+00^*1)\,(0+10^*1)^*\,(0+10^*1)$$

We will take $(1+00^*1)$ as a common factor

$$1+00^*1 \quad \underbrace{(\varepsilon+(0+10^*1)^*(0+10^*1))}_{\downarrow}$$

$$(\varepsilon+R^*R) \quad \text{where} \quad R=(0+10^*1)$$

As we know, $(\varepsilon + R^*R) = (\varepsilon + RR^*) = R^*$

$\therefore\ (1+00^*1)\,((0+10^*1)^*)$ out of this consider

$$\underbrace{(1+00^*1)}_{\downarrow}\,(0+10^*1)^*$$

Taking 1 as a common factor

$$(\varepsilon+00^*)\ 1\ (0+10^*1)^*$$

Applying $\varepsilon+00^*=0^*$

$0^*1\,(0+10^*1)^*$ = R.H.S.

Hence the two regular expressions are equivalent.

Example 2.5.2 *Show that* $(0^*1^*)^*=(0+1)^*$.

Solution :

Consider L.H.S. $=(0^*1^*)^*$

$$= \{\varepsilon, 0, 00, 1, 11, 111, 01, 10, \ldots\}$$

= {any combination of 0's, any combination of 1's, any combination of 0 and 1, ε}

Similarly,

$$\text{R.H.S.} = (0+1)^*$$

$$= \{\varepsilon, 0, 00, 1, 11, 111, 01, 10, \ldots.\}$$

= {ε, any combination of 0's, any combination of 1's,

any combination of 0 and 1}

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

Example 2.5.3 *Show that* $(a\,b)^* \neq (a^*\,b^*)$ **SPPU : Oct.-16, In Sem, Marks 2**

Solution : Consider

L.H.S. = (*a*b)* = {ε, *a*b, *aba*b, *ababa*b, ...}

and R.H.S. = (*a** b*) = {ε, *a*, *aa*, *aa*, b, bb, bbb, *a*b ...}

Note that in the L.H.S, the r.e. will give the strings of '*a*b' combination i.e. *a*b, *aba*b, *ababa*b and so on. But in R.H.S. there is possibility of *a*b but not of *aba*b and so on.

Thus L.H.S. ≠ R.H.S. is proved.

Example 2.5.4 *Show that* $(r+s)^* \neq r^* + s^*$

Solution : Let us consider the L.H.S. first,

L.H.S. = (r + s)*

= {ε, r, rr, s, ss, rs, sr, rsrs, ...}

= {ε , any combination of r and s}

R.H.S. = {ε, r, rr, rrr, s, ss, ...}

{ε, any combination of only r or any combination of only s}

Note that in R.H.S. there is no combination of r and s together. Hence

L.H.S. ≠ R.H.S. is proved.

Example 2.5.5 *Prove that* $r(s+t) = r\,s + r\,t$

Solution : In case of L.H.S.

L.H.S. = r (s + t) solving this further

= r · s + r · t

i.e. r is concatenated with s or with t

= R.H.S.

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

Example 2.5.6 *Check for the following regular expressions for equivalence and justify.*

$R_1 = [(a + bb)^* \ (b + aa)^*]^*$

$R_2 = (a + b)^*$

Solution : Here R_1 and R_2 are equivalent because the strings which can be generated by R_1 can be generated by R_2 or vice versa.

For instance, consider,

$R_1 = abab$ – this string can be generated by selecting **a** from $(a + bb)^*$ then **b** from $(b + aa)^*$ again **a** from $(a + bb)^*$ and **b** from $(b + aa)^*$.

$R_2 = abab$ – this sting can be selecting **a** then **b** repetitively from $(a + b)^*$

Thus $R_1 = R_2$ is proved

Example 2.5.7 $R_1 = (a + b)^* \; abab^*$

$R_2 = b^* a \; (a + b)^* \; ab^*$

Solution : Here strings generated by R_1 are same as R_2

For instance R_1 can generate **abababb**. Similarly R_2 can generate abababb.

Thus $R_1 = R_2$ is proved.

Example 2.5.8 *Prove* $\varepsilon + 1^* (011)^* (1^* (011)^*)^* = (1 + 011)^*$

Solution : Let, L.H.S. is

$$\varepsilon + \underbrace{1^* (011)^*} \; \underbrace{(1^* (011)^*)^*}$$

If we consider $1^* (011)^*$ as P_1 then,

$$= \varepsilon + P_1 P_1^*$$

$$= P_1^* \qquad \because \varepsilon + P_1 P_1^* = P_1^*$$

We can put $P_1 = 1^* (011)^*$ then,

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

Now consider $P_2 = 1$ and $P_3 = (011)$ then it becomes

$$= (P_2^* \; P_3^*)^*$$

$$= (P_2 + P_3)^* \qquad \because (P^* Q^*)^* = (P + Q)^*$$

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

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

Thus L.H.S. = R.H.S. Hence

$\varepsilon + 1^* (011)^* (1^* (011)^*)^* = (1 + 011)^*$ is proved.

Example 2.5.9 *Prove or disprove the following for regular expression r, s and t.*

a) $(rs + r)^* \; r = r \; (sr + r)^*$ *b)* $s(rs + s)^* \; r = rr^* s(rr^* \; s)^*$

c) $(r + s)^* = r^* + s^*$ **SPPU : May-07, Marks 10, Aug.-17, In Sem, Marks 3**

Solution :

a) $(rs + r)^* r = r(sr + r)^*$

Let $r_1 = (rs + r)^* r$

$= \{\varepsilon, rs, r, rsrs, rsr, rsr, rr...\} r$

$= \{\varepsilon, rsr, rr, rsrsr, rsrr, rsrr, rrr ....\}$

$r_2 = r(sr + r)^*$

$= r\{\varepsilon, sr, r, sr\ sr, srr, rsr, rr ...\}$

$= \{\varepsilon, rsr, rr, srsrr, r ...\}$

Here L.H.S. = R.H.S.

Hence $(rs + r)^* r = r(sr + r)^*$

b) $s(rs + s)^* r = rr^* s(rr^* s)^*$

Let $r_1 = s(rs + s)^* r$

$= s\{\varepsilon, rs, s, rss, srs, rsrs, ss ...\} r$

$= \{\varepsilon, rss, ss, rsss, srss, rsrss, sss ...\} r$

$= \{\varepsilon, rrss, ssr, rrsss, srssr, rsrssr, sssr\}$

Let $r_2 = rr^* s(rr^* s)^*$

$r\{\varepsilon, r, rr, rrr, rrrr .....\} s$

$(r\{\varepsilon, r, rr, rrr, rrrr .....\} s)^*$

$rs\{\varepsilon, r, rr, rrr, rrrr .....\}$

$\{rs\} \cup \{\varepsilon, r, rr, rrr, rrrr .....\}$

$\{\varepsilon, rrs, rrrs, rrrrs, rrrrrs .....\}$

$\{\varepsilon, rrs, rrrs, rrrs, rrrrrs\}$

$= \{\varepsilon, rrss, ssr, rrsss .....\}$

Here L.H.S. = R.H.S.

Hence $s(rs + s)^* r = rr^* s(rr^* s)^*$

c) $(r + s)^* = r^* + s^*$

Let $r_1 = (r + s)^*$

$= (\varepsilon, r, s, rs, sr, rsr, srs .....)$

Let $r_2 = r^* + s^*$

$= (\varepsilon, r, rr, rrr, rrr....) \cup (\varepsilon, s, ss, sss, ssss)$

$= (\varepsilon, r, s, s, rs, sr, rsr, srs ....)$

Here L.H.S. = R.H.S.

Hence $(r + s)^* = r^* + s^*$

Example 2.5.10 *Prove a)* $\phi^* = \varepsilon$ *b)* $(r^* s^*)^* = (r + s)^*$

SPPU : Dec.-08, Marks 4, Oct.-16, In Sem, Marks 2

Solution :

a) Let L.H.S. $= \phi^*$

$= \{\varepsilon, \phi, \phi\phi, \phi\phi\phi, \ldots\}$ $\because R^* = \{\varepsilon, R, RR, RRR, \ldots\}$

$= \{\varepsilon\}$ or empty set $\because \phi R = R\phi = \phi$ i.e. empty set.

$= \varepsilon$

= R.H.S.

As, L.H.S. = R.H.S. , $\phi^* = \varepsilon$ is proved.

b) Let, $(r^*s^*)^*$

L.H.S. $= \{\varepsilon, r, rr, rs, sr, rrs, srr, srs, \ldots\}$

$= \varepsilon$ + any combination of r and s

R.H.S. $= \{\varepsilon, r, rr, rs, sr, rsr, rrs, srr, \ldots\}$

$= \varepsilon$ + any combination of r and s

As, L.H.S. = R.H.S.

$(r^*s^*)^* = (r + s)^*$ is true.

Example 2.5.11 *Explain your answer in each of the following :*

1) Every subset of a regular language is regular

2) Every regular language has a regular proper subset. **SPPU : May-06, Dec.-08, Marks 6**

Solution :

1) Every subset of regular language need not be regular. To support if consider a language.

$L_1 = 0^*$ is regular but

$L_2 = \{0^{2n} \mid n \geq 1\}$ is not a regular language.

where $L_2 \subseteq L_1$ Hence given statement is **false.**

2) Every regular language has a regular proper subset. To support this statement, consider a language.

$L_1 = \phi$ it has no proper subset.

L_1 = {ε} then L_2 = ϕ is a proper subset of L_1

L_1 = Language of any substring, has any substring which forms its proper subset.

Hence given statement is **true.**

Example 2.5.12 *If S = {aa, b} write all the string in S* which are having length 4 or less, also say the following is True or False.*

i) $(S^+)^* = (S^*)^*$ *ii)* $(S^+)^+ = S^+$ *iii)* $(S^*)^+ = (S^+)^*$ **SPPU : May-09, Marks 4**

Solution : S = {aa, b}

1) String having length 0 = { ε }
2) Strings having length 1 = { b }
3) Strings having length 2 = { aa, bb }
4) Strings having length 3 = { aab, bbb, baa }
5) Strings having length 4 = { aaaa, aabb, bbbb, bbaa, baab }

Thus we obtain the string with length 4 or less by combining all above obtained sets, { ε, b, aa, bb, aab, bbb, baa, aaaa, aabb, bbbb, bbaa, baab }

i) $(S^+)^* = (S^*)^*$

L.H.S. = $\{aa, b, aab, bb, bbb, ...\}^*$

= { ε + any combination of aa and b }

R.H.S. = $\{ \varepsilon, aa, b, aab, bb, bbb, ...\}^*$

= { ε + any combination of aa and b}

As L.H.S. = R.H.S. given statement is **true.**

ii) $(S^+)^+ = S^+$

L.H.S. = $\{aa, b, aab, bb, bbb, ...\}^+$

= any combination of aa and b without ε.

R.H.S. = {aa, b, aab, bb, bbb, ...}

= any combination of aa and b without ε.

As L.H.S. = R.H.S. given statement is **true.**

iii) $(S^*)^+ = (S^+)^*$

L.H.S. = $\{ \varepsilon + \text{any combination of aa and b} \}^+$

R.H.S. = { any combination of aa and b without ε } + ε

L.H.S. = R.H.S.

Hence given statement is **true**.

Example 2.5.13 *Give the examples of sets that demonstrate the following inequality. Here r_1, r_2, r_3 are regular expressions :*

1) $r_1 + \varepsilon \neq r_1$ *2)* $r_1 \cdot r_2 \neq r_2 \cdot r_1$

3) $r_1 \cdot r_1 \neq r_1$ *4)* $r_1 + (r_2 \cdot r_3) \neq (r_1 + r_2) \cdot (r_1 r_3)$. **SPPU : Dec.-09, Marks 10**

Solution : Assume $r_1 = a$, $r_2 = b$, $r_3 = ab$

1) L.H.S. = $r_1 + \varepsilon = \{ a, \varepsilon \}$

R.H.S. = $r_1 = \{a\}$

$\therefore$ L.H.S. $\neq$ R.H.S.

2) L.H.S. = $r_1 \; r_2 = ab$

R.H.S. = $r_2 \; r_1 = ba$

$\therefore$ L.H.S. $\neq$ R.H.S.

3) L.H.S. = $r_1 \; r_1 = aa$

R.H.S. = $r_1 = a$

$\therefore$ L.H.S. $\neq$ R.H.S.

4) L.H.S. = $r_1 + (r_2 r_3)$

$= a + (b \cdot ab)$

$= a + bab = \{a, bab\}$

R.H.S. = $(r_1 + r_2) \cdot (r_1 r_3)$

$= (a + b)\,(a \cdot ab)$

$= \{aaab, baab)$

$\therefore$ L.H.S. $\neq$ R.H.S.

Example 2.5.14 *Check the equivalence of the regular expressions.*

1) $(a^* bbb)^* a^*$ *AND* $a^*(bbb\, a^*)^*$

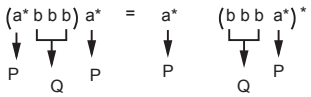
2) $((a + bb)^* aa)^*$ *AND* $\varepsilon + (a + bb)^*.aa$ **SPPU : May-11, Marks 4**

Solution :

1) $(a^* bbb)^* a^*$ AND $a^*(bbb\, a^*)^*$

By identity rule $(PQ)^* P = P\,(QP)^*$

Let us map this rule with given expression



Hence given expressions are equivalent

2) $((a+bb)^* aa)^*$ AND $\varepsilon+(a+bb)^*.aa$

L.H.S. = $((a+bb)^* aa)^*$

= $\{\varepsilon, aaa, bbaa, aaaa, bbaaa, abbaa, ...\}$

R.H.S. = $\varepsilon+(a+bb)^* aa$

= $\{\varepsilon, aaa, bbaa, aaaa, bbaaa, abbaa, ...\}$

L.H.S. = R.H.S.

Hence given expressions are equivalent.

2.6 RE to DFA

SPPU : May-10, 12, 14, 15, 19, Dec.-11, 13, 14, 16, 18, 19, Aug.-15, 17, Oct.-18, 19, Marks 10

Regular expression can be converted to FA using two methods -

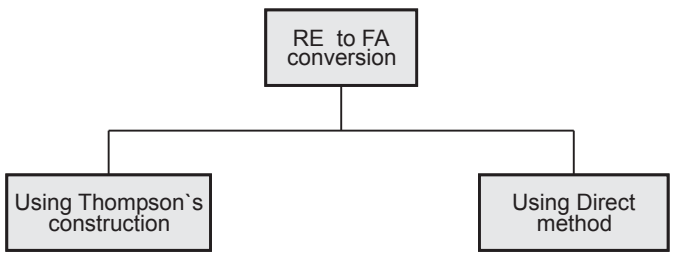


Fig. 2.6.1 Methods of conversion

1. Thomson's Construction

- There are some patterns of regular expressions for which the finite automata is constructed.
- These patterms are -

Case 1 : Zero operators

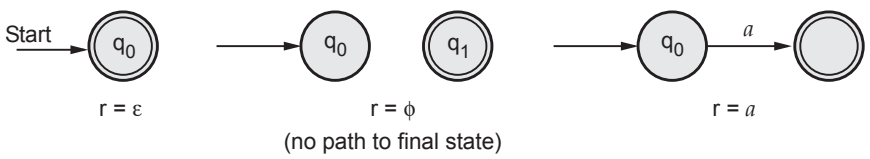


Fig. 2.6.2 Finite automata for given regular expression

Case 2 : Union case

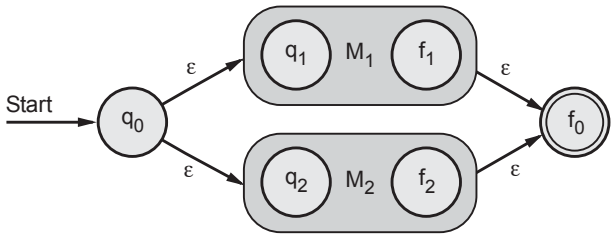


Fig. 2.6.3 The machine M for union

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

Case 3 : Concatenation case

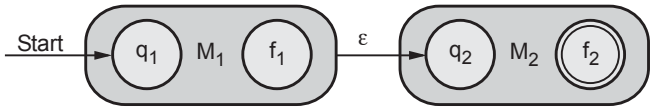


Fig. 2.6.4 Machine M for concatenation

$r = r_1 r_2$ be regular expression

Case 4 : Closure case

$r = r_1^*$ be regular expression

The machine M will be

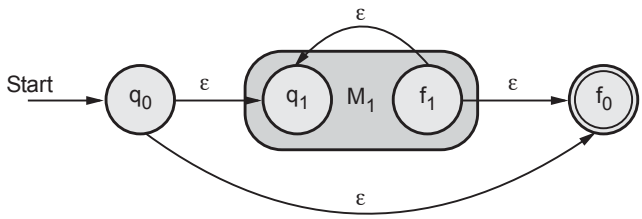


Fig. 2.6.5

- **Direct Method**
  - 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.

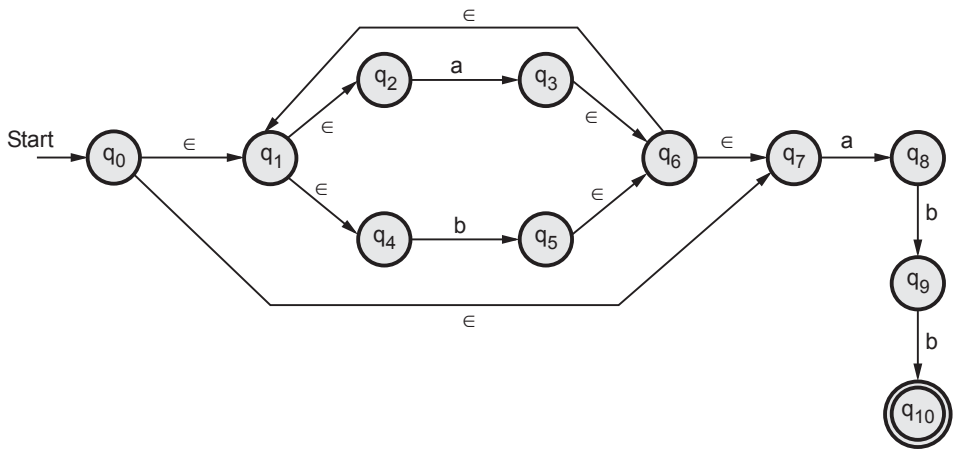
Step 2 : Convert this NFA with ε to NFA without ε.

Step 3 : Convert the obtained NFA to equivalent DFA.

Example 2.6.1 *Construct DFA for the Regular Expression (a + b)* abb.*

SPPU : May-12, Marks 6, Aug.-15, In Sem, Marks 8

Solution : We will first construct NFA - ε for given r.e.



ε - closure (q_0) = $\{q_1, q_2, q_4, q_7\}$ = Call it as state A.

$\delta'(A, a)$ = ε - closure $(\delta(q_1, q_2, q_4, q_7), a)$

= ε - closure (q_3, q_8)

= $\{q_1, q_2, q_3, q_4, q_6, q_7, q_8\}$ = Call it as state B.

$\therefore$ $\delta'(\mathbf{A, a})$ = **B**

$\delta'(A, b)$ = ε - closure $(\delta(q_1, q_2, q_4, q_7), b)$

= ε - closure $(q_5) = \{q_1, q_2, q_4, q_5, q_6, q_7\}$

$\delta'(\mathbf{A, a})$ = **C**

$\delta'(\mathbf{B, a})$ = ε - closure $(\delta(q_1, q_2, q_3, q_4, q_6, q_7, q_8), a)$

= ε - closure (q_3, q_8)

$\delta'(\mathbf{B, a})$ = **B**

(B, b) = ε - closure $(\delta(q_1, q_2, q_3, q_4, q_6, q_7, q_8), b)$

= ε - closure (q_5, q_9)

= $\{q_1, q_2, q_4, q_5, q_6, q_7, q_9\}$

$\delta'(\mathbf{B, b})$ = **D**

$\delta'(C, a)$ = ε - closure $(\delta(q_1, q_2, q_4, q_5, q_6, q_7,), a)$

= ε - closure (q_3, q_8)

$$\delta'(\mathbf{C, a}) = \mathbf{B}$$

$$\delta'(c, b) = \varepsilon\text{ - closure }(\delta(q_1, q_2, q_4, q_5, q_6, q_7), b)$$

$$= \varepsilon\text{ - closure }(q_5)$$

$$\delta'(\mathbf{C, b}) = \mathbf{C}$$

$$\delta'(D, a) = \varepsilon\text{ - closure }(\delta(q_1, q_2, q_4, q_5, q_6, q_7, q_9), a)$$

$$= \varepsilon\text{ - closure }(q_3, q_8)$$

$$\delta'(\mathbf{D, a}) = \mathbf{B}$$

$$\delta'(D, b) = \varepsilon\text{ - closure }(\delta(q_1, q_2, q_4, q_5, q_6, q_7, q_9), b)$$

$$= \varepsilon\text{ - closure }(q_5, q_{10})$$

$$= \{q_1, q_2, q_4, q_5, q_6, q_7, q_{10}\}$$

$$\delta'(\mathbf{D, b}) = \mathbf{E}$$

$$\delta'(E, a) = \varepsilon\text{ - closure }(\delta(q_1, q_2, q_4, q_5, q_6, q_7, q_{10}), a)$$

$$= \varepsilon\text{ - closure }(\{q_3, q_8\})$$

$$\delta'(\mathbf{E, a}) = \mathbf{B}$$

$$\delta'(E, b) = \varepsilon\text{ - closure }(\delta(q_1, q_2, q_4, q_5, q_6, q_7, q_{10})\, b)$$

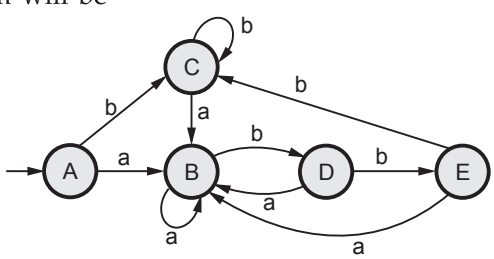
$$= \varepsilon\text{ - closure }(q_5)$$

$$\delta'(\mathbf{E, b}) = \mathbf{C}$$

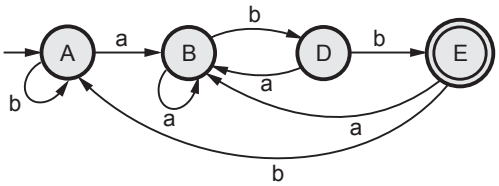
The transition table will be

State \ I/P	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

The transition diagram will be



The states A and C are equivalent. Hence we can have minimized DFA as

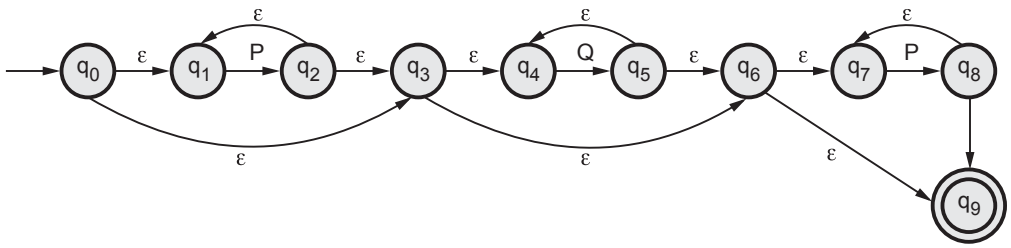


Example 2.6.2 *Show that P*(QP*)* = (P + Q)**

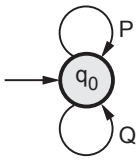
SPPU : May-14, Marks 4

Solution : We will show this equality using FA from R.E.

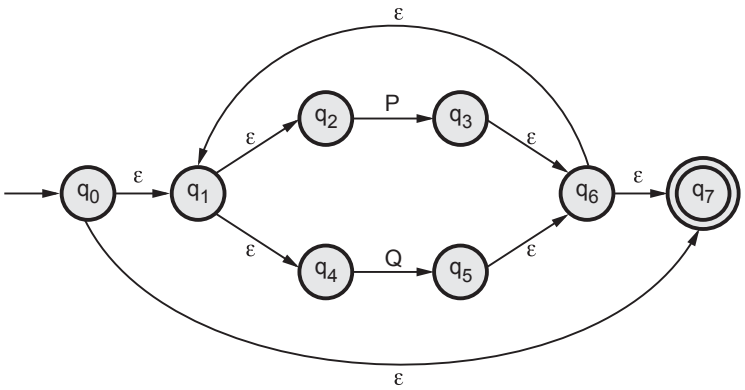
Step 1 : L.H.S. = P* (QP*)*. The NFA with ε for this regular expression will be



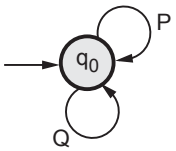
On eliminating ε moves we get -



Step 2 : R.H.S. = (P + Q)*



On elimiating ε moves we get -



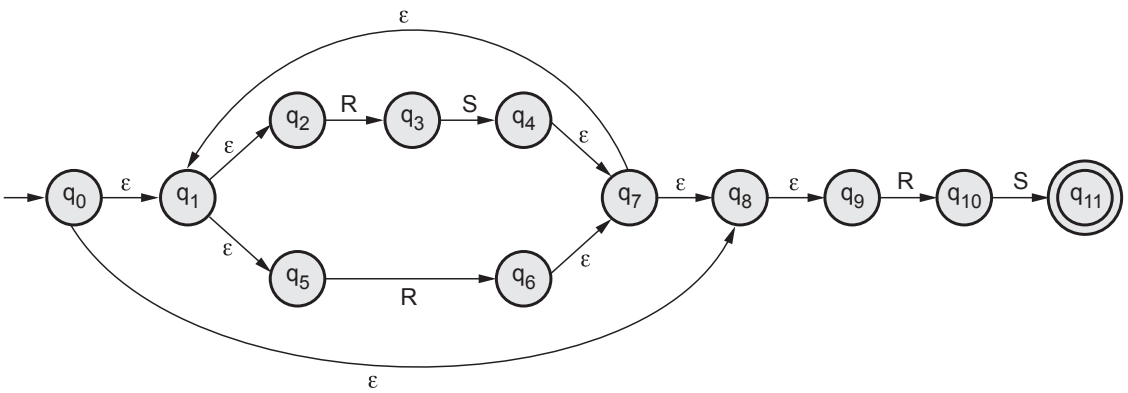
From Step 1 and Step 2 P*(QP*)* = (P + Q)* true.

Example 2.6.3 *Prove or disprove each of the following about regular expression :*

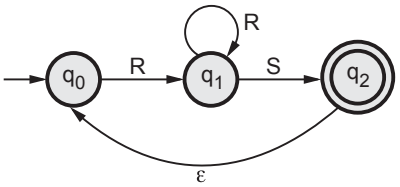
*1) (RS + R)*RS = (RR*S)* 2) (R + S)* = R* + S** **SPPU : Dec.-13, Marks 4**

Solution :

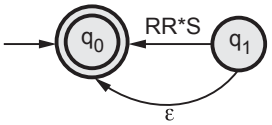
1) Step 1 : L.H.S. = (RS + R)* RS



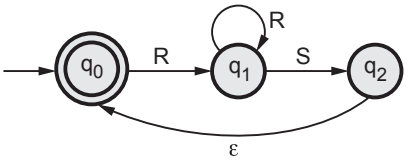
On eliminating ε transitions we get -



Step 2 : R.H.S. = (RR*S)*

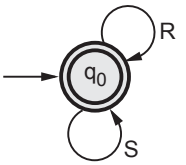


As L.H.S. and R.H.S. both does not yield same states (change in final state)

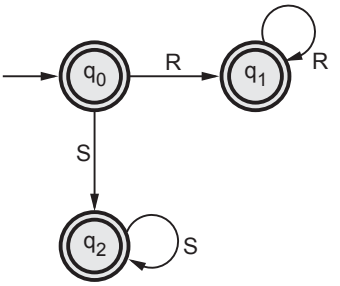


(RS + R)* RS ≠ (RR*S)*

2) Step 1 : (R + S)*



Step 2 : R* + S*

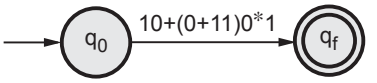


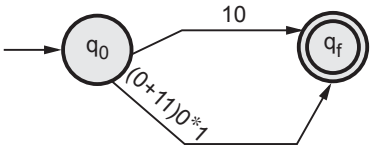
This proves that (R + S)* ≠ (RR*S)*

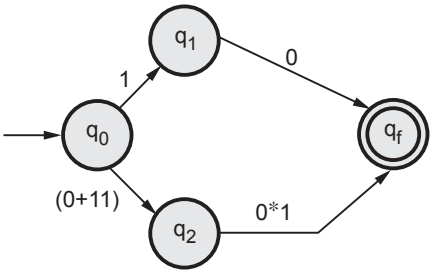
Example 2.6.4 *Design a FA from given regular expression* **10 + (0 + 11) 0* 1**.

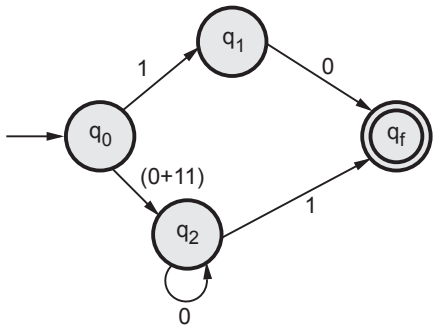
SPPU : Dec.-11, Marks 10

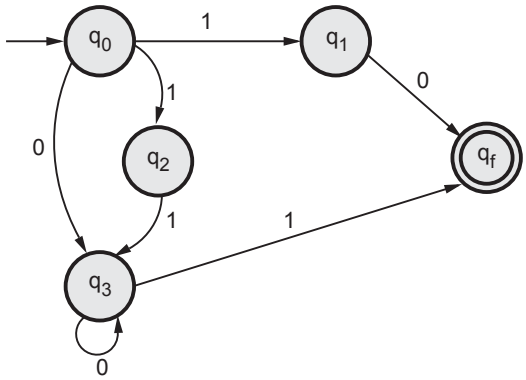
Solution : First we will construct the transition diagram for given regular expression.











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
q_0	q_3	$\{q_1, q_2\}$
q_1	q_f	ϕ
q_2	ϕ	q_3
q_3	q_3	q_f
q_f	ϕ	ϕ

The equivalent DFA will be

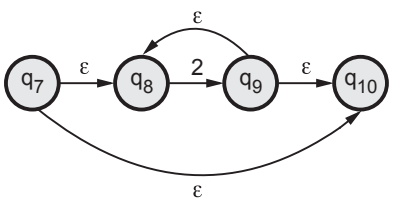
State \ Input	0	1
$[q_0]$	$[q_3]$	$[q_1, q_2]$
$[q_1]$	$[q_f]$	ϕ
$[q_2]$	ϕ	$[q_3]$
$[q_3]$	$[q_3]$	$[q_f]$
$[q_1, q_2]$	$[q_f]$	$[q_3]$
$[q_f]$	ϕ	ϕ

Example 2.6.5 *Construct minimized DFA accepting language represented by regular expression 0*1*2*. Convert given regular expression to NFA with ε moves and then convert NFA with ε moves to DFA using direct method.*

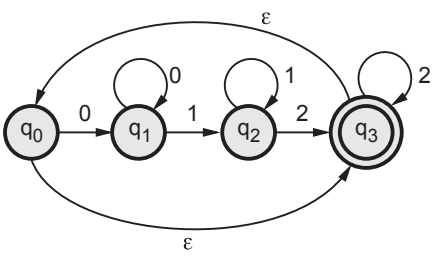
SPPU : Dec.-13, Marks 8, Oct.-19, In Sem, Marks 4

Solution :

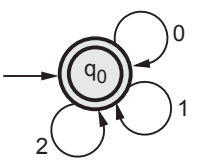
Step 1 :



Step 2 : Byeliminating ε moves



Step 3 : The DFA will be



Example 2.6.6 *Find FA for a given RE : $(0 + 1)^* \cdot (010 + 101) \cdot (0 + 1)^*$.*

SPPU : Dec.-14, End Sem, Marks 4

Solution : The language L contains substring 010 or 101. Hence FA will be,

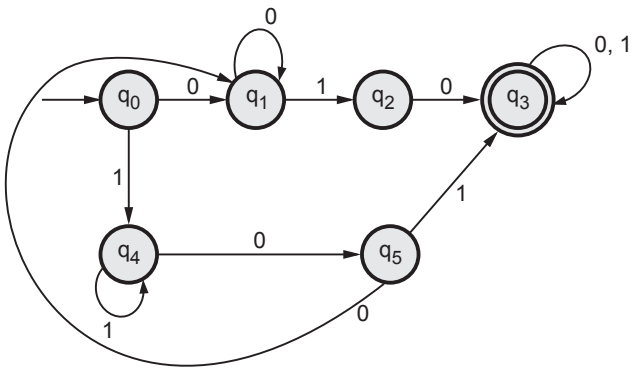
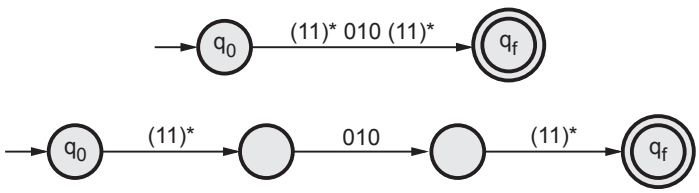


Fig. 2.6.6

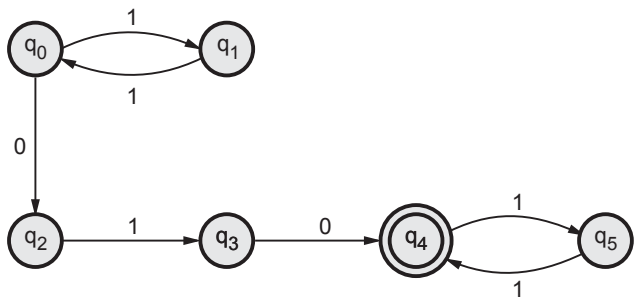
Example 2.6.7 *Construct FA for the regular expression : $(11)^* \cdot 010.(11)^*$.*

SPPU : May-15, End Sem, Marks 4

Solution :



The FA will be :



Example 2.6.8 *Design a Moore machine and Mealy to generate 1's complement of given binary number.*

SPPU : May-10,15, End Sem, Marks 6

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 be as shown in Fig. 2.6.7.

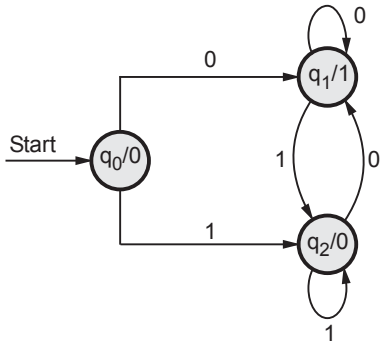


Fig. 2.6.7

For instance, take one binary number 1011 then

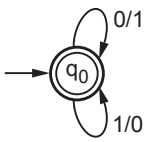
Input		1	0	1	1
State	q_0	q_2	q_1	q_2	q_2
Output	0	0	1	0	0

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 (δ) 0	Next state (δ) 1	Output (λ)
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.



Mealy machine

Example 2.6.9 *Construct finite automata for the following regular expressions :*

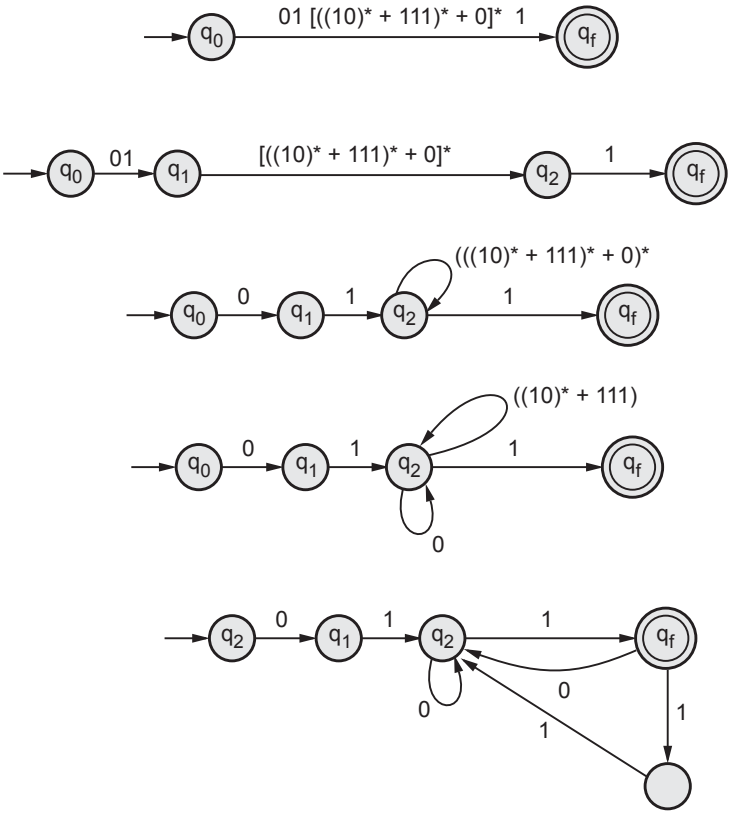
i) $01\,[((10)^* + 111)^* + 0]^*\,1$

ii) $1(1+10)^* + 10(0+01)^*$

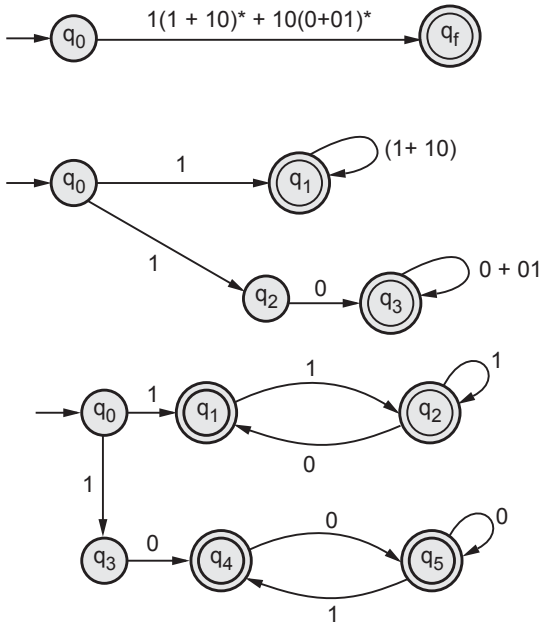
SPPU : Dec.-16, End Sem, Marks 6

Solution :

i)



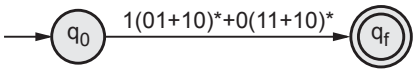
ii)



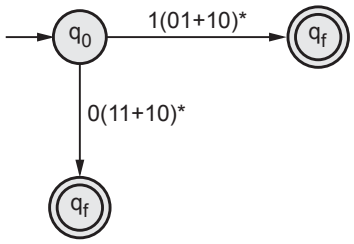
Example 2.6.10 *Construct the finite Automata defined over Σ = (0, 1} for the following regular expression 1 (01 + 10)* + 0 (11 + 10)**

SPPU : Aug.-17, In Sem, Marks 3

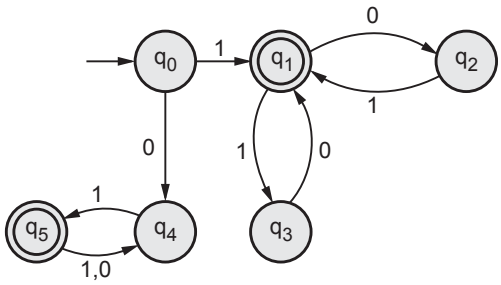
Solution : Step 1 :



Step 2 :



Step 3 :



Example 2.6.11 *Convert the following regular expression to ε - NFA and find the ε - closure of all the states.* $(0+1)^* \cdot 1 \cdot (0+1)$

SPPU : Aug.-17, In Sem, Marks 3

Solution :

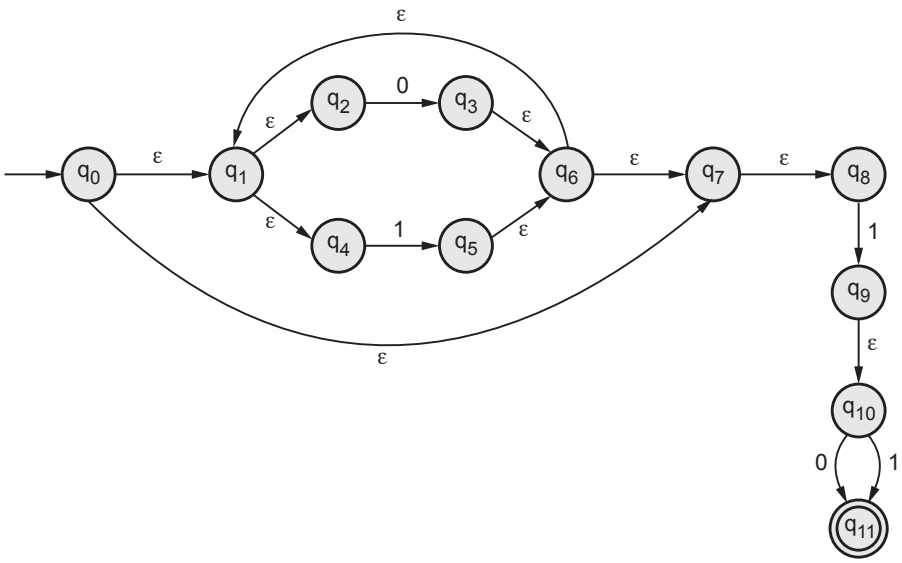


Fig. 2.6.8

ε - closure $(q_0) = \{q_0\}$

ε - closure $(q_1) = \{q_1, q_2, q_4\}$

ε - closure $(q_2) = \{q_2\}$

ε - closure $(q_3) = \{q_3, q_6, q_1, q_2, q_4, q_7, q_8\}$

ε - closure $(q_4) = \{q_4\}$

ε - closure $(q_5) = \{q_5, q_6, q_1, q_2, q_4, q_7, q_8\}$

ε - closure $(q_6) = \{q_6, q_7, q_8, q_1, q_2, q_4\}$

ε - closure $(q_7) = \{q_7, q_8\}$

ε - closure $(q_8) = \{q_8\}$

ε - closure $(q_9) = \{q_9\}$

ε - closure $(q_{10}) = \{q_{10}\}$

ε - closure $(q_{11}) = \{q_{11}\}$

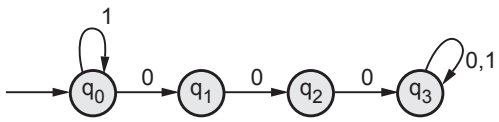
Example 2.6.12 *Design FA accepting the following language over the alphabet {0, 1}*

i) Set of all strings having at least three consecutive zeros.

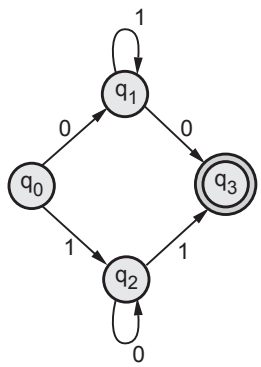
ii) Set of all strings that begin and end with same symbol.

SPPU : Dec.-18, In Sem, Marks 8

Solution : i)



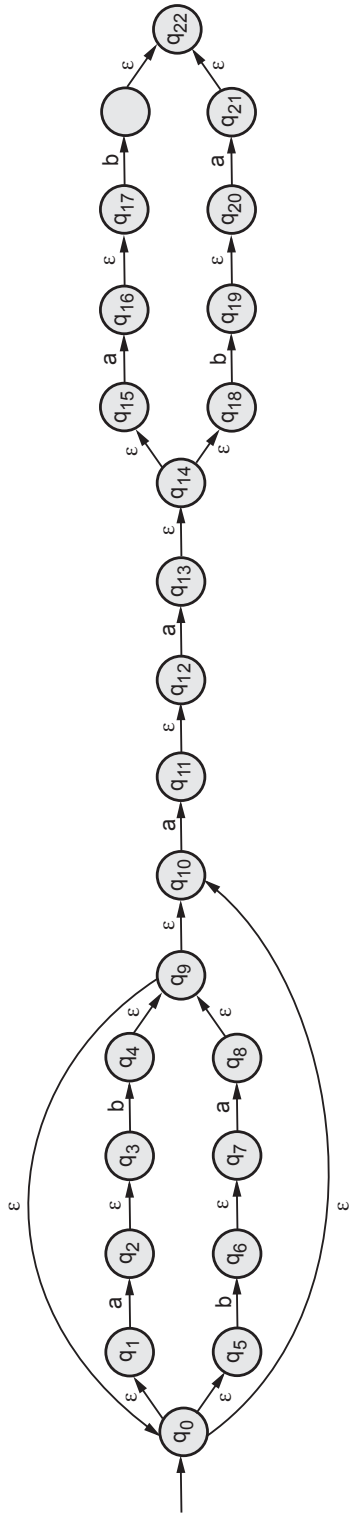
ii)



Example 2.6.13 *For the following regular expression defined over $\sum = \{a, b\}$, draw ε - NFA recognizing the corresponding language (ab + ba)* aa (ab + ba).*

SPPU : Oct.-18, In Sem, Marks 4

Solution :


q0
ε
q1
a
q2
ε
q3
b
q4
ε
q5
b
q6
ε
q7
a
q8
ε
q9
ε
q10
a
q11
ε
q12
a
q13
ε
q14
ε
q15
a
q16
ε
q17
b
ε
ε
q18
b
q19
ε
q20
a
q21
ε
q22
ε
ε

Example 2.6.14 *Show that : i)* $(a.b)^* \neq a^* \cdot b^*$ *ii)* $(a+b)^* = (a+b)^* \cdot (a+b)^*$

SPPU : May-19, End Sem, Marks 6

Solution : i) $(a.b)^* \neq a^* \cdot b^*$

1) Let us represent $(ab)^*$ by FA

L.H.S.

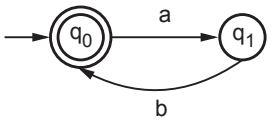


Fig. 2.6.9

L = {^, ab, abab, ababab ...}

2) Let us represent a^* b^*

R.H.S.

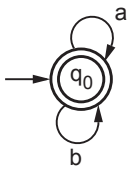


Fig. 2.6.10

L = {^, a, aa, aaa, ..., b, bb, ...,

ab, abab, ba, baba, ...}

From (1) and (2) L.H.S. ≠ R.H.S.

Hence $(ab)^* \neq a^* b^*$

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

Let, R.H.S. be $(a+b)^* + (a+b)^* = (a+b)^*$ $\because$ R + R = R

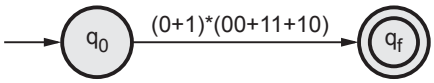
L.H.S.

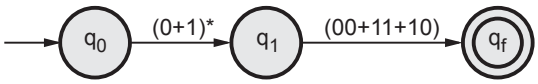
Hence $(a+b)^* = (a+b)^* + (a+b)^*$

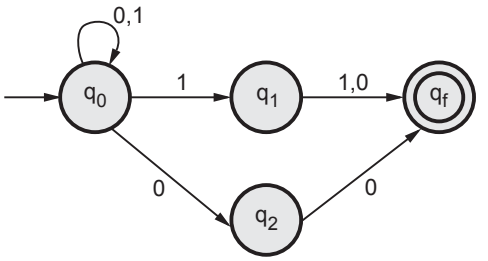
Example 2.6.15 *Construct a finite automata for the following regular expression.*
i) $(1+0)^* (00+11+10)$ *ii)* $1(01+10)^* 0(11+10)^*$ **SPPU : Dec.-19, End Sem, Marks 8**

Solution :

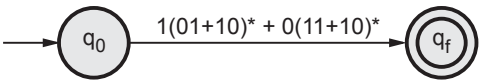
i) r.e. = $(1+0)^* (00+11+10)$

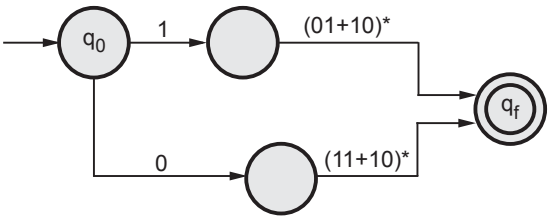


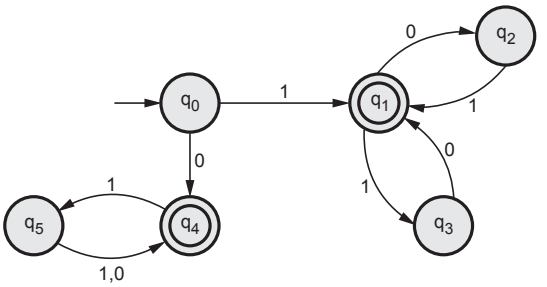




ii) r.e. $= 1(01+10)^* + 0(11+10)^*$







2.7 DFA to RE using Arden's Theorem

SPPU : May-09, 10, 14, 15, 16, Dec.-08, 09, 10, 13, 14, 15, Aug.-15, Marks 10

There are two methods for conversion of DFA to Regular Expression.

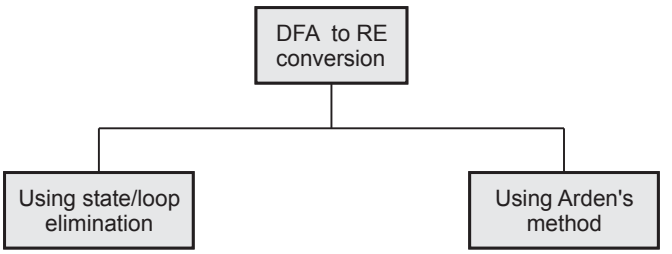


Fig. 2.7.1 DFA to RE conversion methods

2.7.1 State / Loop Elimination

1) The state elimination is the simplest method in which the states of DFA are removed one by one. Until we left with one start and one final state, for each removed state regular expression is generated.
2) In this method, each newly generated regular expression acts as input for a state which is next to removed state.
3) The advantage of this method is it becomes easier to visualize regular expression from the states of FA.

Example 2.7.1 *Convert the following FA to RE using state elimination method.*

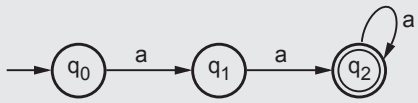


Fig. 2.7.2

Solution :

Step 1 :

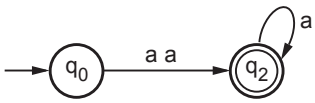


Fig. 2.7.3

Step 2 :

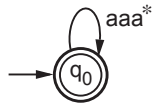


Fig. 2.7.4

Step 3 : R.E. = aaa^*

Example 2.7.2 *Obtain RE from given FA using loop /state elimination method.*

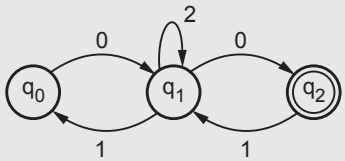


Fig. 2.7.5

Solution :

Step 1 :

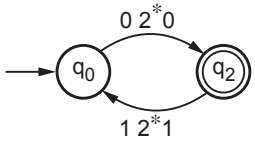
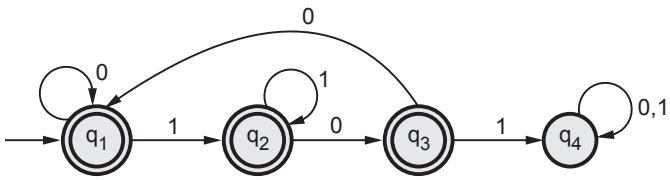


Fig. 2.7.6

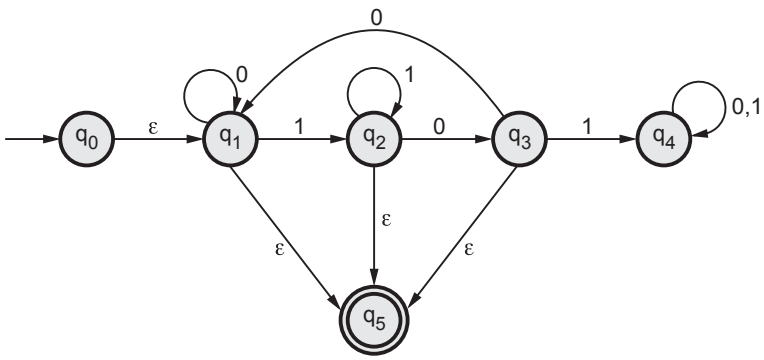
Step 2 : R.E. = $[(02^*0)(1\ 2^*1)]^*$

Example 2.7.3 *Construct NFA and DFA for accepting all possible string of 0's amd 1's not containing 101 as substring. Find regular expression for it.* **SPPU : May-14, Marks 8**

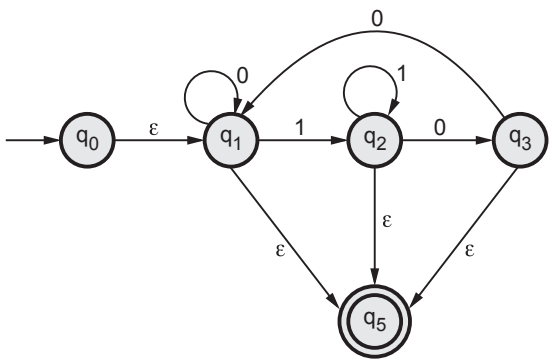
Solution : The DFA for the given language is



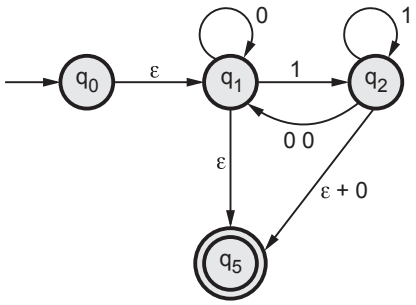
Now we will convert this DFA to NFA with ε. We will add one start state q_0 and new final state i.e. q_5.



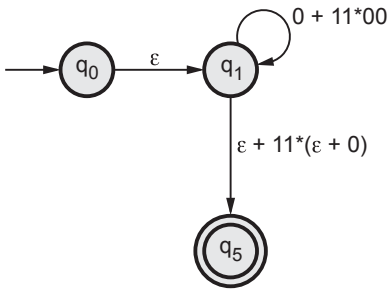
Now we will eliminate state q_4. The NFA will be -



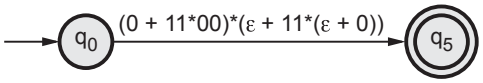
We will eliminate state q_3, without changing meaning of it.



We will eliminate state q_2.



Now we will eliminate state q_1. The NFA will be -



The regular expression will be **(0 + 11*00)* (ε + 11* (ε + 0))**

Example 2.7.4 *Find the regular expression corresponding to each of the following subset of* $\{0,1\}^*$

a) The language of all strings not containing the substring 000

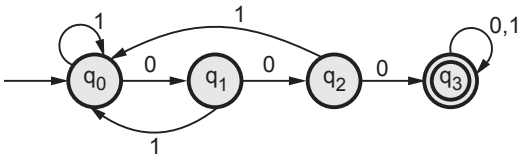
b) The language of all strings that do not contain the substring 110

c) The language of all strings containing both 101 and 010 as substring.

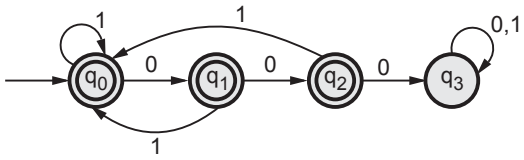
SPPU : May-10, Marks 10

Solution :

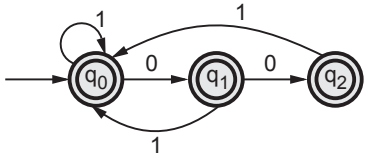
a) To obtain r.e. for this L we will first design FA for strings containing 000 as substring.



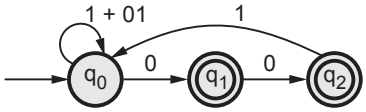
Now to obtain FA for strings not containing 000 as substring just make final state of above FA as non-final and non-final states as final state. The FA will then be -



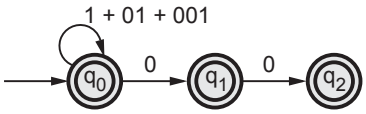
Clearly in above FA q_3 is a dead state. So eliminate it.



Now we will eliminate the edge between q_0 and q_1 which is giving $(01)^*$

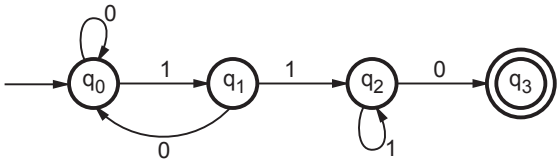


Similarly, we will eliminate edge from q_2 to q_0 giving $(001)^*$

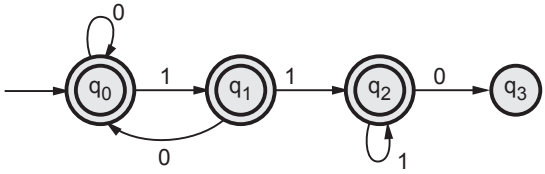


This generates **r.e. = (1+01+001)*** **[$\varepsilon + 0 + 00$]**

b) Let us draw the transition graph for the language that contains 110.



Now for the language that do not contain 110, we will change final states to non-final states and non-final states to final state.



From this diagram, the regular expression will be.

r.e. = $[0^* (10)^* + 1^*]$

c) Regular expression = $[(0+1)^* 101 (0+1)^* 110 (0+1)^*]$

$+ (0+1)^* 110(0+1)^* 101(0+1)^*]$

2.7.2 Arden's Theorem

Conversion Method

Following algorithm is used to build the r.e. from given DFA.

1. Let q_1 be the initial state.
2. There are $q_2, q_3, q_4, \ldots q_n$ number of states. The final state may be some q_j where $j \leq n$.
3. Let α_{ji} represents the transition from q_j to q_i.
4. Calculate q_i such that

   $$q_i = \alpha_{ji} \cdot q_j$$

   If q_i is a start state

   $$q_i = \alpha_{ji}\, q_j + \varepsilon$$

5. Similarly compute the final state which ultimately gives the regular expression r.

Example 2.7.5 *Construct r.e. from given DFA.*

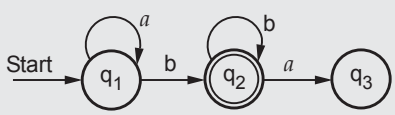


Fig. 2.7.7

Solution : Let us write down the equations

$$q_1 = q_1 a + \varepsilon$$

Since q_1 is a start state ε will be added and the input a is coming to q_1 from q_1 hence we write

State = Source state of input × Input coming to it

Similarly $q_2 = q_1 b + q_2 b$

Let us simplify q_1 first

$$q_1 = q_1 a + \varepsilon$$

We can re-write it as

$$q_1 = \varepsilon + q_1 a$$

which is similar to R = Q + RP which further gets reduced to R = QP*.

Assuming $R = q_1$, $Q = \varepsilon$, $P = a$

We get $q_1 = \varepsilon \cdot a^*$

$q_1 = a^* \quad \because \quad \varepsilon \cdot R^* = R^*$

Substituting value of q_1 in q_2 we get

$$q_2 = q_1 b + q_2 b$$

$$q_2 = a^* b + q_2 b$$

We can compare this equation with R = Q + R P assuming $R = q_2$, $Q = a^*b$, $P = b$ which gets reduced to R = QP*.

$\therefore \quad q_2 = a^*b \cdot b^*$

As $R R^* = R^+$

$$q_2 = a^* \cdot b^+$$

From the given DFA, if we want to find out the regular expression, we normally calculate the equation for final state. Since in the given DFA q_2 is a final state and $q_2 = a^* b^+$. We can conclude as the DFA represents $a^* b + b^+$ as regular expression.

Example 2.7.6 *Find out the regular expression from given DFA.*

SPPU : Dec.-15, End Sem, Marks 6

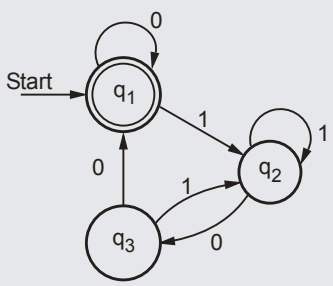


Fig. 2.7.8

Solution : Let us solve the DFA by writing the regular expression, for each state.

$q_1 = q_1 0 + q_3 0 + \varepsilon \quad \because$ Initial state

$q_2 = q_2 1 + q_3 1 + q_1 1$

$q_3 = q_2 0$

For getting the r.e. we have to solve q_1 the final state.

$$q_2 = q_2 1 + q_2 01 + q_1 1$$

$$q_2 = q_2 (1 + 01) + q_1 1$$

We will compare R = Q + R P with above equation, so R = q_2, Q = $q_1 1$, $P = (1+01)$ which ultimately gets reduced to QP*.

$$q_2 = q_1 1(1+01)^*$$

Substituting this value to q_1

$$q_1 = q_1 0 + q_3 0 + \varepsilon$$

$$= q_1 0 + q_2\ 00 + \varepsilon$$

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

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

Again $R = Q + R\ P$

where $R = q_1$

$Q = \varepsilon$

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

Hence $q_1 = \varepsilon \cdot [0 + 1(1+01)^* \cdot 00]^*$

$q_1 = [0 + 1(1+01)^* \cdot 00]^*$ $\because$ $\varepsilon \cdot R = R$ is required r.e.

Example 2.7.7 *Represent the language given by following DFA.*

SPPU : Dec.-14, End Sem, Marks 4

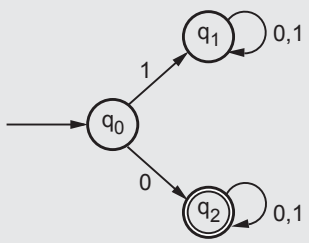


Fig. 2.7.9

Solution : To find the language, we will solve for regular expression,

$$q_0 = \varepsilon$$

$$q_2 = q_2 0 + q_2 1 + q_0 0$$

Substituting the value of q_0 in q_2

$$q_2 = q_2\ (0 + 1)\ \varepsilon \cdot 0$$

$$q_2 = q_2\ (0 + 1) + 0 \quad \because \quad R = Q + R\ P \Rightarrow QP^*$$

$$\boxed{q_2 = 0\ (0 + 1)^*} \quad \text{where } Q = 0,\ P = (0 + 1),\ R = q_2$$

This is a final regular expression, which represents the language containing all the strings which start with letter 0.

Example 2.7.8 *Describe in english the language indicated by following regular expression.*

SPPU : Dec.-10, Marks 6

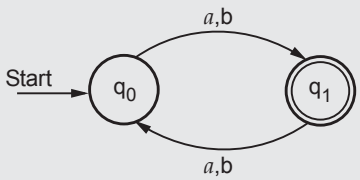


Fig. 2.7.10

Solution : The equations for the states are

$$q_0 = q_1 a + q_1 b + \varepsilon \quad \ldots(1)$$

$$q_1 = q_0 (a + b) \quad \ldots(2)$$

We will simplify q_0 first

$$q_0 = (a + b) q_1 + \varepsilon$$

Let us put this value of q_0 into equation (2).

$$q_1 = \left((a+b) q_1 + \varepsilon\right) (a + b)$$

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

This is equivalent to R = Q + R P where R = q_1, Q = $(a + b)$, P = $(a + b)(a + b)$ which gets reduced to QP*.

$$\therefore \quad \boxed{q_1 = (a + b) \cdot \left[(a+b)(a+b)\right]^*}$$

This is a regular expression indicated by DFA.

The language given by this DFA is the language in which it accepts all the strings of odd length.

Example 2.7.9 *Consider the FA and construct regular expressions that is accepted by it. Fig. 2.7.11.*

SPPU : Dec.-13, Marks 6

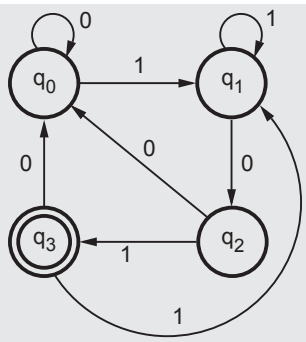


Fig. 2.7.11

Solution : We will first write the equations for each state, assuming q_0 as initial state

$$q_0 = q_0 0 + q_2 0 + q_3 0 + \varepsilon \quad \ldots (1)$$

$$q_1 = q_0 1 + q_1 1 + q_3 1 \quad \ldots (2)$$

$$q_2 = q_1 0 \quad \text{... (3)}$$

$$q_3 = q_2 1 \quad \text{... (4)}$$

Now we will place value of q_2 in equation (4)

$$q_3 = q_1 01 \quad \text{... (5)}$$

Put value of equation (5) in equation (2)

$$q_1 = q_0 1 + q_1 1 + (q_1 01)\, 1$$

$$q_1 = q_0 1 + q_1 1 + q_1 011$$

$$\underbrace{q_1}_{R} = \underbrace{q_1}_{R} \underbrace{(1 + 011)}_{P} + \underbrace{q_0\ 1}_{Q}$$

gives us

$$q_1 = q_0 1\,(1 + 011)^* \quad \because R = Q + RP \text{ gives } R = QP^*$$

Now solving q_0

$$q_0 = q_0 0 + q_2 0 + q_3 0 + \varepsilon$$

$$= q_0 0 + (q_1 0)\, 0 + (q_1 01) 0 + \varepsilon$$

$$q_0 = q_0 0 + q_1 (00 + 010) + \varepsilon$$

$$= q_0 0 + q_0 1\,(1 + 011)^*\ (00 + 010) + \varepsilon$$

$$q_0 = q_0\ (0 + 1\,(1 + 011)^*\ (00 + 010)) + \varepsilon$$

$$\therefore \quad q_0 = \varepsilon\,(0 + 1(1 + 011)^*\ (00 + 010))^*$$

Now we will solve

$$q_3 = q_1 01$$

$$= q_0 1\,(1 + 011)^*\ 01$$

$$q_3 = (0 + 1(1 + 011)^*\ (00 + 010))^*\ 1\,(1 + 011)^*\ 01.$$

As q_3 is a final state, we get the r.e. as equation for q_3.

$$\therefore \quad \boxed{\textbf{r.e.} = (0 + 1(1 + 011)^*\ (00 + 010))^*\ 1\,(1 + 011)^*\ 01)}$$

Example 2.7.10 *Design the finite automata and then equivalent regular expression using Arden's theorem that accepts the set of all strings over the alphabet {a, b} with an equal number of a's and b's, such that each prefix has atmost one more a than b's and atmost one more b than a's.* **SPPU : Dec.-08, 09; May-09, Marks 10**

Solution : The FA will be as shown in Fig. 2.7.12.

The valid strings 0101 contains prefix of 1 as 010 in which number of 0's are more than 1. Similarly in a valid string 1010 the prefix of 0 contains more number of 1's than 0's .

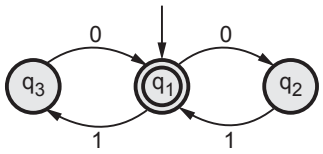


Fig. 2.7.12

Now in order to obtain the regular expression for this FA, we will write state equations as follows -

$$q_1 = q_2 1 + q_3 0 + \varepsilon \quad \text{... (1)}$$

$$q_1 = q_1 0 \quad \text{... (2)}$$

$$q_3 = q_1 1 \quad \text{... (3)}$$

There is only one final state and that is q_1. Hence equation of q_1 state is the regular expression.

$$\underset{R}{q_1} = \underset{R}{q_1}\,\underbrace{(01 + 10)}_{P} + \underset{Q}{\varepsilon}$$

$\because$ Applying Arden's theorem.

Let, $q_1 = q_2 1 + q_3 0 + \varepsilon$

$= q_1 01 + q_1 10 + \varepsilon$ $\quad\because$ Put equations 2 and 3 in equation 1

$q_1 = \varepsilon (01 + 10)^*$ $\quad\because$ R = Q + RP is R = QP*

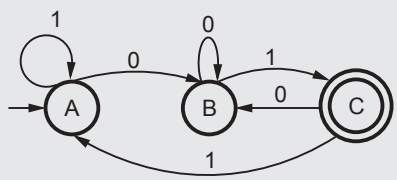
$q_1 = (01 + 10)*$ $\quad\because$ $\varepsilon R^* = R^*$

Hence regular expression for given language is

$$\boxed{\textbf{r.e.} = (01 + 10)^*}$$

Example 2.7.11 *Convert the following finite automation into its equivalent regular expression using Arden's Theorem.*

SPPU : May-15, End Sem, Marks 6



Solution : A = A1 + C1 + ε … (1)

B = B0 + A0 + C0 … (2)

C = B1 … (3)

Putting equation (3) in equation (1), we get -

A = A1 + B11 + ε

$$A = \underset{R}{A}\,\underset{P}{1} + \underbrace{B11 + \varepsilon}_{Q}$$

$\therefore \quad A = (B11+\varepsilon)1^* \quad \ldots (4)$

Now, put equation (3) in equation (2),

$$B = B0 + A0 + B10 \quad \ldots(5)$$

put equation (4) in equation (5),

$$B = B0 + (B11 + \varepsilon)\, 1^*\, 0 + B\, 10$$

$$= B0 + B\, 111^*\, 0 + 1^*0 + B\, 10$$

$$B = \underset{R}{B}\,(\underbrace{0 + 111^*\, 0 + 10}_{P}) + \underbrace{1^*\, 0}_{Q}$$

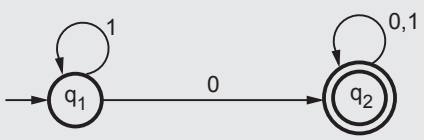
$\therefore \quad B = 1^*0\ (0 + 111^*\ 0 + 10)^* \quad \ldots (6)$

Put equation (6) in equation (3),

$$\mathbf{C = 1^*0\ (0 + 111^*\ 0 + 10)^*\ 1} \text{ is R.E.}$$

Example 2.7.12 *Obtain the regular expression that denotes the language acepted by the following DFA, using Arden's Theorem :*

SPPU : Aug.-15, In Sem, Marks 4



Solution : Let us write the equations for each state

$q_1 = q_1 1 + \varepsilon \quad \ldots (1)$

$q_2 = q_1 0 + q_2(0+1) \quad \ldots (2)$

Let us solve equation (1) using Arden's theorem

$$q_1 = \underset{R}{q_1},\ \underset{R}{1} + \underset{P}{\varepsilon}\ \underset{Q}{}$$

$\therefore \quad q_1 = \varepsilon 1^* = 1^* \quad \because \ R = RP + Q$ gives $R = QP^*$

$\therefore \quad q_1 = 1^* \quad \ldots (3)$

Putting equation (3) in (2) we get

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

$$\therefore \quad q_2 = 1^* 0(0+1)^* \quad \ldots (4)$$

As q_2 is a final state equation for q_2 represents regular expression.

$$\therefore \quad \boxed{\textbf{r.e.} = 1^* 0(0+1)^*}$$

Example 2.7.13 *Make use of Arden's theorem to determine the regular expression for the finite automata shown in Fig. 2.7.13.* **SPPU : May-16 Marks 6**

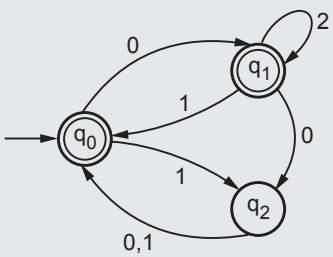


Fig. 2.7.13

Solution :

$$q_0 = \varepsilon + q_1 1 + q_2 (0+1) \quad \ldots(1)$$

$$q_1 = q_0 0 + q_1 2 \quad \ldots(2)$$

$$q_2 = q_0 1 + q_0 0 \quad \ldots(3)$$

Now we will put equation (3) in (1)

$$q_0 = \varepsilon + q_1 1 + (q_0 1 + q_1 0)(0+1)$$

i.e. $$q_0 = \varepsilon + q_1 1 + q_0 10 + q_0 11 + q_1 00 + q_1 01$$

i.e. $$= \varepsilon + q_1 (1 + 00 + 01) + q_0 1(0+1)$$

$$q_0 = \underbrace{\varepsilon}_{R} + \underbrace{q_1(1 + 00 + 01)}_{Q} + \underbrace{q_0}_{R}\underbrace{1(0 + 1)}_{P}$$

$$\therefore \quad q_0 = [\varepsilon + q_1(1+00+01)](1(0+1))^* \quad \ldots (4)$$

Now solving equation (2) with the help of equation (4)

$$q_1 = q_0 0 + q_1 2$$

$$q_1 = [\varepsilon + q_1(1+00+01)](1(0+1))^* 0 + q_1 2$$

$$= (1(0+1))^* 0 + q_1(1+00+01)(1(0+1))^* 0 + q_1 2$$

$$q_1 = \underbrace{(1\ (0+1))^*}_{Q}\underbrace{0}_{R} + \underbrace{q_1}_{R}\ \underbrace{[(1+00+01)(1(0+1))^* 0 + 2]}_{P}$$

$$\therefore \quad q_1 = (1(0+1))^* 0[(1+00+01)(1(0+1))^* 0 + 2]^*$$

From equation (4) the q_0 will be

$$q_0 = (\varepsilon + q_1 + (1+00+01))(1(0+1))^*$$

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

$$q_0 = (1(0+1))^* + ((1(0+1)^* 0)\ (1+00+01)(1(0+1)^* 0 + 2)\ (1+00+01)(1(0+1))^*$$

Now for final regular expression,

$$\text{R.E.} = q_0 + q_1$$

$$\therefore \quad \text{R.E.} = [(1+(0+1))^* + ((1\ (0+1))^* 0)(1+00+01)$$

$$((1(0+1)^* 0 + 2)(1+00+01)(1\ (0+1))^*] + [1(0+1)^* 0[(1+00+01)(1(0+1))^* 0 + 2]$$

2.8 Pumping Lemma for Regular Languages

SPPU : Dec.-06, 08, 12, May-06, 07, 09, 12, 13, 14, 16, Aug.-17, Oct.-18, Marks 8

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 $a_1, a_2, a_3, \ldots a_m$, $m \geq n$. The mapping function δ could be written as $\delta(q_0, q_1, q_2, q_3, \ldots q_i) = q_i$.

The transition diagram is as shown in Fig. 2.8.1.

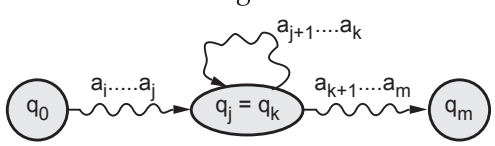


Fig. 2.8.1 Pumping lemma

If q_m is in F i.e. $q_1, q_2, q_3, \ldots q_m$ is in L(M) then $a_1, a_2, \ldots a_j\, a_{k+1}\, a_{k+2} \ldots\ldots a_m$ 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 $a_{j+1} \ldots a_k$. Thus

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

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.**

2.8.1 Applications of Pumping Lemma

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 2.8.1 *Show that the set* $L = \left\{ b^{i^2} \mid i > 1 \right\}$ *is not regular.*

SPPU : Dec.-12, Marks 6, Aug.-17, Oct.-18, In Sem, Marks 4

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

$$L = b^{1^2} = b \qquad \text{the length} = 1^2$$

$$= b^{2^2} = bbbb \qquad \text{length} = 2^2$$

and so on.

Now let us consider

$$L = b^{n^2} \text{ where} \qquad \text{length} = n^2$$

it is denoted by z.

$$|z| = n^2$$

By pumping lemma $z = u\ v\ w$

where $1 \le |v| \le n$

As $z = u\, v^i\, w$ where i = 1

Now we will pump v i.e. make i = 2.

As we made i = 2 we have added one n^2.

$1 \leq \mid v \mid \leq n.$

$n^2 + 1 \leq \mid u\,v\,w \mid \leq n + n^2$

i.e. $n^2 + 1 \leq \mid u\,v\,w \mid \leq n^2 + n + n + 1$

i.e. $n^2 + 1 \leq \mid u\,v\,w \mid \leq (n+1)^2$

$= n^2 \leq \mid u\,v\,w \mid \leq (n+1)^2$

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,

$$L = b^{i^2}$$

Let $i = 2$

$L = bbbb$

$L = uvw$

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.

uvw

$= uv.vw$

$= bbbbbb$

$=$ length of b is not a perfect square

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 2.8.2 *Is the following language regular ? Justify your answer* $L = \{0^{2n} \mid n \geq 1\}$.

Solution : This is a language in which length of string is always even.

i.e. $n = 1$; $L = 00$

$n = 2$ $L = 00\ 00$ and so on.

Let $L = uvw$

$L = 0^{2n}$

$|z| = 2n = u\,v^{i}\,w$

If we add 2n to this string length.

$|z| = 4n = uv.vw$

$= $ Even length of string.

Thus even after pumping 2n to the string we get the even length. So the language L is regular language.

Example 2.8.3 *Prove* $L = \{a^{p} \mid p \text{ is a prime}\}$ *is not regular.*

SPPU : Dec.-06, May-07, Marks 6, May-12, Marks 8

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

$L = a^{P}$

$|z| = uvw \qquad i = 1$

Now consider $L = u\,v^{i}\,w$ where $i = 2$

$= uv.vw$

Adding 1 to p we get,

$p < |uvvw|$

$p < p+1$

But $p+1$ is not a prime number. Hence what we have assumed becomes contradictory. Thus L behaves as it is not a regular language.

Example 2.8.4 *Show that* $L = \{0^{n}\,1^{n+1} \mid n > 0\}$ *is not regular.*

Solution : Let us assume that L is a regular language.

$|z| = |uvw|$

$= 0^{n}\ 1^{n+1}$

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 2 n + 1

$$2n + 1 < (2n + 1) + 2n + 1$$

$$2n + 1 < 4n + 2$$

But if n = 1 then we obtain 4 n + 2 = 6 which is no way odd. Hence the language becomes irregular.

Even if we add 1 to the length of |z|, then

$$|z| = 2n + 1 + 1 = 2n + 2$$

$$= \text{Even length of the string.}$$

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 2.8.5 *Show that set $L = \{0^i 1^i \mid i \geq 1\}$ is not regular.*

Solution : Assume that $L = \{0^i 1^i \mid i \geq 1\}$ is regular.

Let, $w = 0^n 1^n$ such that $|w| = 2n$. By pumping lemma we can write

$w = xyz$ such that $|xy| \leq n$ and $|y| \neq 0$.

Now if $xy^iz \in L$ then the language L is said to be regular.

There are many cases – i) y has only 0's ii) y has only 1's iii) y has both 0's and 1's.

i) If y has only 0's then the string

$w = 0^{n-k} 1^n = xz$ since $y = 0^k$ and $i = 0$

Surely $n - k \neq n$. Hence $xz \notin L$.

Hence our assumption of being L regular is wrong.

ii) If y has only 1's then, for i = 0 and $y = 1^k$

$$\therefore \quad w = xz = 0^n 1^{n-k}$$

As $n \neq n - k$, $xz = w \notin L$

Again L is not regular.

iii) If y has 0's and 1's then

$$w = 0^{n-k} 0^k 1^j 1^{n-j} = xy^iz$$

If i = 2 then

$$w = 0^{n-k} 0^{2k} 1^{2j} 1^{n-j} \notin L$$

Hence from all these 3 cases it is clear that language L is not regular.

Example 2.8.6 *Use pumping lemma to show whether or not the following languages are regular :*

i) $A_1 = \{www \ / \ w \ \varepsilon \ (a, b)^*\}$ *ii)* $A_2 = \{a^{2n} \ / \ n \geq 0\}$.

Solution : i) Let,

$A_1 = \{www \mid w \ \varepsilon \ (a, b)^*\}$ is assumed to be a regular language. Then by pumping lemma the string $w = xyz \in A_1$ such that

$$w = xy^iz$$

Here $|w| = |www| = 3\ n$ where $n > 0$

By pumping lemma if

$$w = xy^iz$$
$$= ww^iw$$
$$|w| = > 3\ n$$

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

$$|w| = a^{2n} = xy^iz$$

Then $|w| > 2\ n$. That means we may get a string of odd length. Hence given language is not regular.

Example 2.8.7 *Let L be any subset of* 0^*. *Prove that* L^* *is regular.*

SPPU : May-07, 14, Marks 4

Solution : Let $L \subseteq 0^*$ i.e. L can be $\{\varepsilon, 0, 00, 000 \ldots\}$

Now we will apply pumping lemma to determine whether L is subset of 0^*.

Assume $w = xyz$ such that $|xy| \leq n |y| \geq 0$

If we consider string w = 0000

We will map $w = xyz$ to 0000

By pumping lemma, if $w = xy^iz \in L$ the L is regular. If **i = 2** then

i.e. $w \in L$

This proves that string w is regular. Hence if L is subset of 0^* then L^* is regular is proved.

Example 2.8.8 *Using pumping lemma for regular sets prove that the language* $L = \{a^m b^n \mid m > n$ *is not regular}* **SPPU : Dec.-08, Marks 6**

Solution : Let us assume

$$L = \{a^m b^n \mid m > n\} \text{ is regular.}$$

As m > n, if we assume m = n + 1 then

$$L = a^{n+1} b^n = a^n a b^n$$

By pumping lemma $L \in W$ such that

$$W = xyz$$

If we map $L = a^n a b^n$ to W then

$$W = \mathbf{a^n\ a\ b^n}. \text{ Assume } x = a^n \quad y = ab,\ z = b^{n-1}$$

If $W = xy^i z$ with i = 2 then

$$W = a^n a b a b b^{n-1} \notin L.$$

$$W = a^n b^n \notin L$$

This proves that our assumption of L being regular is wrong. Given L is not regular.

Example 2.8.9 *Prove that the following is not regular languages the set of strings of the form* $0^i 1^j$ *such that the greatest common divisor of i and j is 1.*

Solution : Let $L = 0^i 1^j$ such that gcd (i, j) = 1.

We assume that this is a regular language. Now consider three cases -

Case 1 : Let

$$z = \overbrace{0 \ldots 0}^{i}\ \overbrace{1 \ldots 1}^{j} \text{ be a string} \in L.$$

According to pumping lemma if z = uvw is a string $\in$ L. then $z = uv^i w \in L$. That means even if we pump some substring the string $z \in L$.

We map z = uvw as

$$z = \underbrace{0 \cdots 0}_{u} \underbrace{\overbrace{0 \cdots 0}^{i}\ \overbrace{1 \cdots 1}^{j}}_{v} \underbrace{1 \cdots 1}_{w}$$

This is a case in which v consists of substring $0^n 1^n$.

By pumping lemma $z = uv^i w$ if i = 2, z = uvvw

Then we get

$$z = 0^{i-m}(01)^m 1^{j-m} \notin L = \{0^i\ 1^j\}$$

Case 2 : Let,

$$z = \underbrace{\overbrace{0 \ldots 0}^{i}}_{uv} \underbrace{\overbrace{1 \ldots 1}^{j}}_{w} \in L$$

By pumping lemma $z = uv^iw$. If i = 2 then,

$$z = uvvw$$

i.e. $$z = 0^{i+m}\, 1^j$$

But gcd (i+m, j) ≠ 1.

Hence $z \notin [L = \{0^i\, 1^j\}]$

Case 3 : Let,

$$z = \underbrace{\overbrace{0 \ldots 0}^{i}}_{u} \underbrace{\overbrace{1 \ldots 1}^{j}}_{vw}$$

By pumping lemma, $z = uv^iw$. If i = 2 then,

$$z = uvvw$$

i.e. $$z = 0^i\, 1^{j+m}$$

But gcd (i, j+m) ≠ 1.

Hence $z \notin [L = \{0^i\, 1^j\}]$

From above 3 cases we can prove that $L = 0^i1^j | \gcd(i, j) = 1$ is not a regular language.

Example 2.8.10 *Prove or disprove that the language L given by $L = \{ a^m b^n \mid m \neq n$, m and n are positive integers$\}$ is regular.* **SPPU : Aug.-17, In Sem, Marks 4**

Solution : Let $L = \{ a^m b^n \mid m \neq n\}$ be a language. Assume that L is regular language. Now consider following case -

Case 1 : Assume $z = aaabbbb \in L$

By pumping lemma if z = uvw then if we pump some strings to make $z = uv^iw$ then if $z \in L$ then such a language is called regular.

Let,

$$z = \underset{u}{\underset{\downarrow}{a}}\ \underbrace{a\ \ a\ \ b\ \ b}_{v}\ \underbrace{b\ \ b}_{w}$$

if $z = uv^iw$ and if i = 2 then z = uvvw

$$\therefore \quad z = \underset{u}{a}\ \underbrace{(a\ a\ b\ b)}_{v}\ \underbrace{(a\ a\ b\ b)}_{v}\ \underbrace{b\ b}_{w}$$

$$z = a^3\, b^2\, a^2\, b^4 \notin a^m\, b^n \notin L$$

Case 2 : Assume $L \in z$ such that

$$z = \underset{u}{a}\ \underbrace{a\ a}_{v}\ \underbrace{a\ b\ b\ b\ b\ b\ b}_{w}$$

By pumping lemma, $z = uv^iw \in L$ for a regular language. If i = 2 then,

$$z = uvvw$$

$$\therefore \quad z = \underset{u}{a}\ \underbrace{a\ a}_{v}\ \underbrace{a\ a}_{v}\ \underbrace{a\ b\ b\ b\ b\ b}_{w}$$

$$z = a^6\, b^6 \in a^m\, b^n \text{ because } m = n.$$

From these case we get $z \notin L$. Thus our assumption of L being regular is wrong. Hence given language $\{ L = a^m\, b^n \mid m \neq n \}$ is not a regular.

Example 2.8.11 *Show that $L = \{ww / w \in \{a, b\}\}$ is not regular.* **SPPU : Dec.-12, Marks 6**

Solution : Assume that the language L is regular. This also means that there exists some DFA with n states.

If we consider $w = a^n\, b$ i.e. string of {a, b}

then $\quad ww = a^n\, b\, a^n\, b$ is in L

i.e. $\quad |ww| = 2n + 2 > n$

by pumping lemma, we can write

$$ww = xyz \text{ with } |y| = 0 \ |xy| \leq n$$

iii) For checking regularity we need to find whether $xy^iz \in L$ or not.

Case 1 : $w = xy^iz$

we assume i = 0. This is a case with y having no b's.

Then $\quad w = xy$ i.e. $y = a^k$

Y containing no b's at all

$$w = a^m\, b\, a^n\, b$$

where $m = n - k$

Then we can not write xz in the form of ww. Because w belongs to set $\{a, b\}^*$.

Case 2 : This is a case with y having only one b.

Then assume

$$w = xy^i z$$

These w = xz with only one b.

But then $xz \notin L$ as in language L there should be even number of a's and even number of b's. [we have assumed $|ww| = 2n + 2$].

Thus from case 1 and case 2 we get the contradictions. So L is not regular.

Example 2.8.12 *Define Pumping Lemma and apply it to prove the following :*

$L = \{0^m 1^n 0^{m+n} \mid m >= 1 \text{ and } n >= 1\}$ *is not regular.* **SPPU : May-09, 16, Marks 6**

Solution : Assume that L is regular by pumping lemma, we can write w = xyz with $|y| = 0$ $|xy| \le n$

For checking the regularity we need to find whether $xy^i z \in L$ or not.

$$w = xyz$$

$$= 0^m 1^n 0^{m+n}$$

Case 1 : If we assume y has only 1's i.e.

$$w = \underset{x}{\underset{\downarrow}{0^m}} \underset{y}{\underset{\downarrow}{1^n}} \underset{z}{\underset{\downarrow}{0^{m+n}}} \quad \text{then with } i = 2,$$

$$w = xy^i z \text{ becomes } xyyz$$

$$\text{i.e.} \quad = 0^m 1^n 1^n 0^{m+n}$$

$$w = 0^m 1^{2n} 0^{m+n} \notin L$$

Case 2 : If we assume y is set of x y 01 i.e.

$$w = \underset{x}{\underset{\downarrow}{0^{m-k}}} \underbrace{0^k \quad 1^{n-p}}_{y} \underbrace{1^p \quad 0^{m+n}}_{z}$$

then with i = 2 $w = xyyz$

$$\text{i.e. } w = 0^{m-k} (0^k 1^{n-p}) (0^k 1^{n-p}) 1^p 0^{m+n}$$

$$w = 0^{m-k} 0^{2k} 1^{2n-2p} 1^p 0^{m+n}$$

$$= 0^{m-k} 1^{2n-p} 0^{m+n} \notin L$$

From above two cases by applying pumping lemma i.e. by using

$$w = xy^i z \notin L$$

Hence our assumption of L being regular is wrong.

∴ Given L is not regular.

Example 2.8.13 *Find whether the language is regular or not*
$L = \{WW^R \in \{a, b\}\}$

Solution : Consider, $L = \{w \in \{a, b\} | w = w^R\}$

Consider $w = abab$

$w^R = baba$

Hence $L = ww^R$

The string z can be denoted by

$z = uvw.$ where $|z| \geq n$ and $|uv| \leq n$ $|v| \geq 1$.

We assume $z = \underbrace{ab}_{n}\underbrace{ab}_{n}\underbrace{ba}_{n}\underbrace{ba}_{n}$

We can split $z = uvw$ as

$z = \underbrace{a}_{u}\underbrace{b}_{1}\underbrace{ab\,ba\,ba}_{w}$

$|u| = n - 1$

$|v| = 1$

i.e. $|uv| = |u| + |v|$

$= n - 1 + 1$

$= n$

Now according to pumping lemma,

when $z = uv^iw \in L$ then language is said to be regular. We will assume that $L = ww^R$ is regular. And we will find uv^iw.

∴ $z = uv^iw = ababbaba$

Assume $i = 0$ i.e. there will be

$z = uv^0w = uw$

Then the string becomes

$z = aabbaba$

$\neq ww^R$

i.e. $z = aabbaba \notin L$ …(1)

Now consider i = 2 then

$$z = uv^2w$$
$$= abbabbaba$$
$$\neq ww^R$$

i.e. $z = abbabbaba \notin L$ ...(2)

From equations (1) and (2) we can state that

$$z = uv^iw \notin L$$

Hence our assumption that $L = ww^R$ being regular is wrong. Hence we can prove that

$$z = ww^R \text{ is not regular.}$$

Review Questions

1. *Explain the property pumping lemma of regular set. Where do we apply this property ? Give the suitable example.* **SPPU : May-06, Marks 6**
2. *State pumping lemma for regular sets.* **SPPU : May-13, Marks 2**

2.9 Closure Properties of Regular Languages

SPPU : May-18, Oct 18, Marks 6

If certain languages are regular and language L is formed from them by certain operations (such as union or concatenation) then L is also regular. These properties are called closure properties of regular languages. Such languages represent the class of regular laguages which is closed under the certain specific operations.

The closure properties express the idea that when one or many languages are regular then certain related languages are also regular. The closure properties of regular languages are as given below.

1. The union of two regular languages is regular.
2. The intersection of two regular languages is regular.
3. The complement of a regular languages is regular.
4. The difference of two regular languages is regular.
5. The reversal of a regular languages is regular.
6. The closure operation on a regular language is regular.
7. The concatenation of regular language is regular.
8. A homomorphism of regular languages is regular.
9. The inverse homomorphism of regular language is regular.

Theorem 1 : If L_1 and L_2 are two languages then $L_1 \cup L_2$ is regular.

Proof : If L_1 and L_2 are regular then they have regular expression $L_1 = L(R_1)$ and $L_2 = L(R_2)$. Then $L_1 \cup L_2 = L(R_1 + R_2)$ thus we get $L_1 \cup L_2$ as regular language. (any language given by some regular expression is regular).

Theorem 2 : The complement of regular language is regular.

Proof : 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 $\overline{L}_1$ 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'.

Thus as $\overline{L}_1$ is accepted by DFA M', it is regular.

Theorem 3 : If L_1 and L_2 are two regular languages then $L_1 \cap L_2$ is regular.

Proof : Consider that languages L_1 is regular. That means there exists some DFA M_1 that accepts L_1. We can write $M = (Q_1, \Sigma, \delta_1, q_1, F_1)$. 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 the 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.

It is clear that there exists some DFA which accepts $L_1 \cap L_2$ i.e. L. Hence L is a regular language. This proves that if L_1 and L_2 are two regular languages then $L_1 \cap L_2$ is regular. In other words the regular language is closed under intersection.

Theorem 4 : If L_1 and L_2 are two regular languages then $L_1 - L_2$ is regular.

Proof : The $L_1 - L_2$ can also be denoted as $L_1 \cup \overline{L}_2$

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 $\overline{L}_1$ 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 $\overline{L}_2$ are two regular languages. That also means : these languages are accepted by regular expressions. If $L_1 = L(R_1)$ and $\overline{L}_2 = L(R_2')$. Then $L_1 \cup \overline{L}_2 = L(R_1 + R_2')$. This ultimately shows that $L_1 \cup \overline{L}_2$ is regular. In other words $L_1 - L_2$ is regular. Thus regular languages are closed under difference.

Theorem 5 : The reversal of a regular languages is regular.

Proof : Reversal of a string means obtaining a string which is written from backward that is w^R is denoted as reversal of string w. That means $L(w^R) = (L(w))^R$. This proof can be done with basis of induction.

Basis : If $w = \varepsilon$ or ϕ then w^R is also ε or ϕ

i.e. $(\varepsilon)^R = \varepsilon$ and $(\phi)^R = \phi$

Hence $L(w^R)$ is also regular.

Induction :

Case 1 : If $w = w_1 + w_2$ then

$$w = (w_1)^R + (w)^R$$

As the regular language is closed under union. Then w is also regular.

Case 2 : If $w = w_1 w_2$

Consider $w_1 = (ab, bb)$

and $w_2 = (bbba, aa)$

The $w_1^R = (ba, aa)$

$w_2^R = (aaab, bb)$ then

$$w = w_1^R w_2^R$$

(ba, aa, aaab, bb) is also regular. Thus given language is regular one.

Theorem 6 : The closure operation on a regular language is regular.

Proof : If language L_1 is regular then it can be expressed as $L_1 = L(R_1^*)$. Thus for a closure operation a language can be expressed as a language of regular expressions. Hence L_1 is said to be a regular language.

Theorem 7 : If L_1 and L_2 are two languages then $L_1 \cdot L_2$ is regular. In other words regular languages are closed under concatenation.

Proof : If L_1 and L_2 are regular then they can be expressed as $L_1 = L(R_1)$ and $L_2 = L(R_2)$. Then $L_1 \cdot L_2 = L(R_1 \cdot R_2)$ thus we get a regular language. Hence it is proved that regular languages are closed under concatenation.

Theorem 8 : A homomorphism of regular languages is regular.

Proof : The term homomorphism means substitution of string by some other symbols.

For instance the string " aabb" can be written as 0011 under homomorphism. Clearly here, a is replaced by 0 and b is replaced by 1. Let $\sum$ is the set of input alphabets and Γ be the set of substitution symbols then $\sum^* \rightarrow \Gamma^*$ is homomorphism. The definition of homomorphism can be extended as

Let, $w = a_1 a_2 \ldots a_n$

$h(w) = h(a_1) h(a_2) \ldots h(a_n)$

If L is a language that belongs to set $\sum$, then the **homomorphic image** of L can be defined as :

$$h(L) = \{ h(w) : w \in L \}$$

To prove that if L is regular h(L) is also regular consider following example -

Let, $\sum = \{a, b\}$ and w = abab

Let $h(a) = 00$

and $h(b) = 11$

Then we can write h(w) = h(a) h(b) h(a) h(b)

= 00110011

The homomorphism to language is applied by applying homomorphism on each string of language.

$\therefore$ If L = ab*b then,

L = {ab, abb, abbb, abbbb, ...}

Now h(L) = {0011, 001111, 00111111, 0011111111, ...}

$\therefore$ $h(L) = 00\ (11)^+$ As it can be represented by a regular expression, it is a regular language. Hence it is proved that if L is regular then h(L) is also regular. In other words, family of regular languages is closed under homomorphism.

Theorem 9 : The inverse homomorphism of regular language is regular.

Proof : Let $\sum^* \rightarrow \Gamma^*$ is homomorphism.

The $\sum$ is the input set and Γ be the substitution symbols used by homomorphic function.

Let, L be the regular language where $L \in \sum$, then h(L) be homomorphic language.

The inverse homomorphic language can be represented as $h^{-1}(L)$

Let,

$$h^{-1}(L) = \{ w \mid w \in L \}$$

If L is regular then h(L) is also regular because regular language is closed under homomorphism. That if there exist a FA M = $(Q, \in, \delta, q_0, F)$ which accepts L then h(L) must also be accepted by FA M. For complement of L i.e. language L' the inverse homomorphic language is $h^{-1}(L)$. Let M' be the FA in which all the final states of M

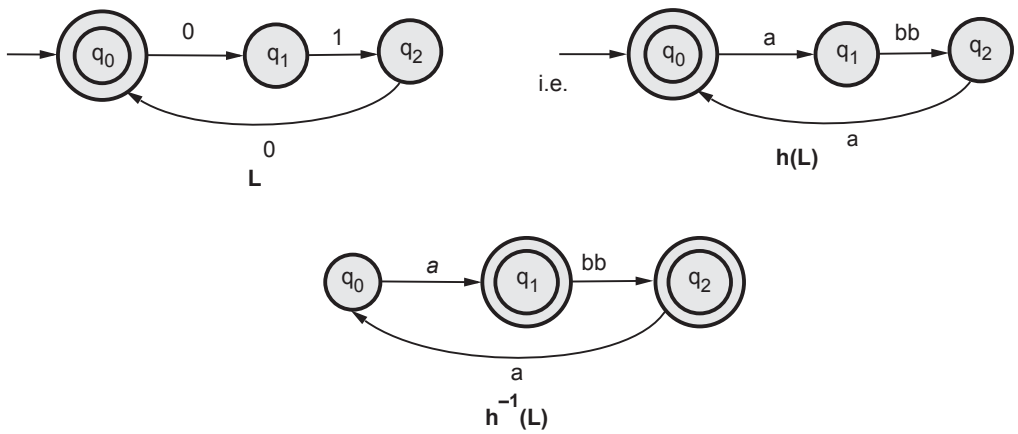
become non-final states and all the non-final states of M become the final states. Clearly the language L' can be accepted by M' Hence $h^{-1}(L)$ must also be accepted by FA M'.

For example - Let L = (010)* be a regular language. And $L = \{010, 010010, \ldots\}$

Let h(0) = a and h(1) = bb be homomorphic function. Then

h(L) = (abba)*

This L can be represented by following DFA.



Thus there exists a FA which accepts $h^{-1}(1)$. This shows that, inverse homomorphism of regular language is regular.

Review Questions

1. *With respect to properties of regular languages explain what is pumping lemma and closure properties of regular languages.* **SPPU : May-18, End Sem, Marks 6**
2. *Prove that "Regular language is closed under complementation"* **SPPU : Oct 18, In Sem, Marks 3**

2.10 Decision Properties of Regular Languages

Following is the set of questions for determining the decision properties of regular languages.

1. Is a particular string w belongs to some language L ? **[Membership Property]**

   Let L be a language which can be represented by DFA. If a string w is chosen for simulating this DFA processing and then if w reaches to an accept state starting from the start state then it is said that.

2. Is the given language empty ? **[Emptiness property]**

   The regular language is modelled by a finite automata to decide whether it is empty or not. If there exists some path from start state to accept state for deriving such language then the given language is non empty. But if accepting states are separated from start state or in other words if there is no path reaching to an accept state from a start state while deriving a given language then such a language is called empty language.

3. Do the two descriptions of a language represent the same language ? or whether two languages are equivalent ? **[Equivalence property]**

   Any regular language can be represented by NFA, NFA with DFA or by regular expression. These representations can be interconverted. This property is called equivalence of corresponding two models. For instance : NFA to DFA, DFA to r.e. and so on. If two different descriptions can be modelled by two equivalent but different models, the one can decide whether they are describing the same language or not.

2.11 Limitation of FA

SPPU : Aug.-15, Marks 2

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 palindrome condition of given language.

Thus FA cannot accept non regular and context free languages. It can accept only regular languages.

Review Question

1. *What are the limitations of finite automata ? Justify with suitable examples*

   SPPU : Aug.-15, In Sem, Marks 2

2.12 Myhill - Nerode Theorem

The Myhill - Nerode theorem was proposed by **John Myhill and Anil Nerode** who proved it at University of Chicago. This theorem is used to prove that the given language is not regular. The theorem can be stated as follows -

Theorem - Given a language L, define a relation R_L on strings by rule xR_Ly if there is no distinguishing extension z with the property that exactly one of the strings xz and yz is in L.

The Myhill - Nerode theorem is useful for indistinguishability of strings i.e. Strings x and y in Σ^* are indistingusihable with respect to a language L if and only if for any string z in Σ^*, either xz and yz are both in L or they are both not in L.

For example, a and aa are indistinguishable with respect to the language a^n over alphabet {a}, where n is a positive integer, because aa^k and aaa^k are in the language a^n for any positive integer k. However, with respect to the language a^nb^n, a and aa are not indistinguishable (hence distinguishable), because ab is in the language a^nb^n while aab is not in the language. Using this concept of indistinguishability, the theorem by **Myhill and Nerode** gives a criterion for non-regularity of a language. Hence we can state the theorem in other words as,

"A language L over alphabet Σ is nonregular if and only if there is an infinite subset of Σ^*, whose strings are pairwise distinguishable with respect to L".

Example 2.12.1 *Prove that* $L = \left\{a^n b^n \,\middle|\, n > 0\right\}$ *is non regular using Myhill - Nerode theorem.*

Solution : We will assume a set of strings S = $\left\{a^n \,\middle|\, n > 0\right\}$. The set S is over alphabet $\{a, b\}$ and it is infinite. We have to show that its strings are pairwise distinguishable with respect to language L.

Assume that a^i and a^j be arbitrary two different members of the set S, where i and j are positive integers and i ≠ j.

Let b^j as a string to be appended to a^i and a^j. It then becomes $a^i\ b^j$ and $a^j\ b^j$. From these two string $a^i\ b^j$ is not L while $a^j\ b^j$ is in L. Hence we can say that a^i and a^j are arbitrary strings of S and are distinguishable with respect to L. Thus S satisfies the conditions of Myhill-Nerode theorem. This proves that L is non-regular.

Example 2.12.2 $L = \left\{ ww \,\middle|\, w \varepsilon \{a, b\}^* \right\}$ *is non-regular.*

Solution : We will assume a set of strings S = $\left\{ a^n \,\middle|\, n > 0 \right\}$. The set S is over alphabet {a, b} and it is infinite. We have to show that its strings are pairwise distinguishable with respect to language L.

Assume that a^i and a^j be arbitrary two different members of the set S, where i and j are positive integers and i ≠ j. Select $ba^i\ b$ as a string to be appended to a^i and a^j. Then $a^i\ b\ a^i\ b$ is in L while $a^j\ b\ a^i\ b$ is not in L. Here a^i and a^j are arbitrary strings of S and are distinguishable with respect to L. Thus S satisfies the conditions of Myhill-Nerode theorem. Hence L is non-regular.

2.13 Case Study

SPPU : Dec.-05, 06, 10, 13, 14, May-13, 14, Oct.-19, Marks 6

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 + .. + 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 bit 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 -

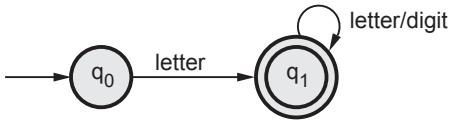


Fig. 2.13.1

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.

Review Questions

1. *Explain the following applications of regular expression.*
   i) GREP utilities in unix. ii) Finding pattern in text.

SPPU : Dec.-05, 06, May-14, Marks 6

2. *Explain the use of regular expressions in unix with any one example.*

SPPU : Dec.-10, Marks 4

3. *Explain the applications of regular expressions in lexical analysis phase of compiler.*

SPPU : Dec.-13, May-13, Marks 6

4. *State applications of Regular Expression* **SPPU : Dec.-14 ,End Sem, Marks 4**

5. *Explain the application of regular expression in GREP utilities in Unix*

SPPU : Oct.-19, In Sem, Marks 5

2.14 Fill in the Blanks

Q.1 The zero or more occurrences of a letter is denoted using ___ operator.

Q.2 The one or more occurrences of a letter is denoted using ___ operator.

Q.3 The set of strings that end in 0 is denoted as ____.

Q.4 The theorem for checking non regularity of a language is ____.

Q.5 If P, Q, R are three regular expressions and if R = Q + RP then it has a unique solution given by _____.

Q.6 The RE in which any number of 0's is followed by any number of 1's followed by any number 2's is ____.

Q.7 In regular expression for adjoining two symbols we use ___ operator.

Q.8 In regular expression the null or empty string is denoted by ____.

Q.9 Concatenation of R with Ø gives ____.

Q.10 Ø* = ____.

Q.11 If A and B are regular languages then A* is ____.

2.15 Multiple Choice Questions

Q.1 Regular expressions are used to represent which language _____.

a recursive language
b context free language
c regular language
d all of these

Q.2 The set of all strings over ∑ = {a,b} in which all strings of a's and b's ending in bb is _____.

a ab
b a*bbb
c (a+b)* bb
d all of these

Q.3 Which of the following is not a prefix of abc ?

a ε
b a
c ab
d bc

Q.4 What is R+R _____.

a R
b 0
c R+
d None

Q.5 Any transition graph has an equivalent _____.

a RE
b DFA
c NFA
d DFA, NFA, RE

Q.6 Which of the following is not a regular expression ?

a (a+b)*abb
b abba*
c 0 + 1
d (a+b)*-(aa)

Q.7 Regular expressions are _____.

a type 0 language
b type 1 language
c type 2 language
d type 3 language

Q.8 Regular expressions are closed under _____.

a union
b intersection
c kleen star
d all of these

Q.9 Regular expression(a/b)(a/b) denotes the set of _____ .

a {ab,ab}
b {aa,bb,ab,ba}
c {aa,bb}
d {a,b}

Q.10 Which of the following identity is wrong ?

a R + R = R
b (R*)* = R*
c εR = Rε = R
d ØR = RØ = RR*

Q.11 Concatenation operation refers to which of the following set operations :

a Union
b Dot
c Kleene
d a and c

Answer Keys for Fill in the Blanks :

Q.1	*	**Q.2**	+
Q.3	(0 + 1)*0	**Q.4**	pumping lemma
Q.5	QP*	**Q.6**	0*1*2*
Q.7	dot or	**Q.8**	*
Q.9	Ø	**Q.10**	Ø
Q.11	regular		

Answer Keys for Multiple Choice Questions :

Q.1	a	**Q.2**	c	**Q.3**	d	**Q.4**	a
Q.5	d	**Q.6**	d	**Q.7**	a	**Q.8**	d
Q.9	b	**Q.10**	d	**Q.11**	b		

❑❑❑

UNIT - III

3 Context Free Grammar (CFG) and Context Free Language (CFL)

Syllabus

Basic Elements of Grammar, Formal Definition of Context Free Grammar, Sentential form, Derivation and Derivation Tree/ Parse Tree, Context Free Language (CFL), Ambiguous Grammar, writing grammar for language. ***Simplification of CFG :*** *Eliminating ε - productions, unit productions, useless production and useless symbols.* ***Normal Forms :*** *Chomsky Normal Form, Greibach Normal Form, Pumping Lemma for CFG, Closure properties of CFL, Decision properties of CFL, Chomsky Hierarchy, Cock-Younger-Kasami Algorithm.*

Contents

3.1 Basic Elements of Grammar
3.2 Formal Definition of Context Free Grammar
3.3 Sentential Form . . . . . **Aug.-17,** . . . . . Marks 3
3.4 Context Free Language (CFL) **Dec.-05, 06, 07, 08, 09, 10, 13, 14, 17,**
. . . . . **May-08, 09, 10, 13, 14, 19, Aug.-17,**
. . . . . **Oct.-16, 18, 19,** . . . . . Marks 12
3.5 Derivation . . . . . **May-15,** . . . . . Marks 4
3.6 Parse Tree
3.7 Ambiguous Grammar **May-07, 11, 12, 14,**
. . . . . **Dec.-05, 12, 13, 14,** . . . . . Marks 8
3.8 Simplification of CFG **Dec.-16, 18, 19, Aug.-17,**
. . . . . **Oct.-18, 19,** . . . . . Marks 8
3.9 Normal Forms **Dec.-05, 10, 13, 14, 15, 16, 17, 19,**
. . . . . **May-08, 10, 13, 14, 18,**
. . . . . **Oct.-18,** . . . . . Marks 12
3.10 Pumping Lemma for CFG
3.11 Closure Properties of CFL . . . . . **May 09, Dec.-18,** . . . . . Marks 6
3.12 Decision Properties of CFL. . . . . **Dec.-11,** . . . . . Marks 5
3.13 Chomsky Hierarchy . . . . . **May-06, Dec.-11,** . . . . . Marks 8
3.14 Cock-Younger-Kasami Algorithm
3.15 Applications of CFG **May-06, 11, Dec.-06, 13,**
. . . . . **Oct.-16, 19,** . . . . . Marks 6
3.16 Case Studies
3.17 Fill in the Blanks
3.18 Multiple Choice Questions

3.1 Basic Elements of Grammar

A grammar is a 4 tuple G = (V,T,P,S) where V is a finite set of **non terminal symbols**, T is a finite set of **terminal symbols**, P is a set of **production rules** and S is a **start symbol**.

The P is of the form A->B where A and B ∈ (VUT)*.

The grammar is normally represented using derivation tree.

For example

The production rules for defining natural language are as follows -

Sentence → Noun verb

Noun → Rama | Sita | Gopal

Verb → sings | eats | dances

A derivation tree can also be built using the production rules.

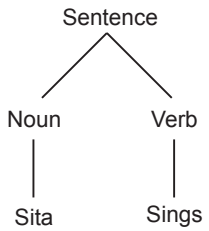


Fig. 3.1.1 Phrase tree for the string Sita Sings

3.2 Formal 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 → Non terminals

or Non terminal → Terminals

S is a start symbol.

For example

$$
\begin{aligned}
P = \{ & S \rightarrow S + S \\
& S \rightarrow S * S \\
& S \rightarrow (S) \\
& S \rightarrow 4 \}
\end{aligned}
$$

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 → NOUN VERB

NOUN → Rama | Sita | Gopal

VERB → goes | writes | sings

Thus if want to derive a string "Rama sings" then we can follow the above rules.

3.3 Sentential Form

SPPU : Aug.-17, Marks 3

Consider G = (V, T, P, S) be a context free grammar then, we can derive a string w from it. This w can 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 $S \overset{*}{\Rightarrow} w$ which is called as **sentential form**. If $S \underset{lm}{\overset{*}{\Rightarrow}} w$ then w is a **left-sentential form.**

If $S \underset{rm}{\overset{*}{\Rightarrow}} w$ then w is a **right-sentential form.**

For example

Consider the grammar,

$$S \rightarrow S + S \mid S*S \mid 4$$

Then for deriving the string w = 4+4*4 we use above grammar as -

$$S \underset{lm}{\Rightarrow} S*S \underset{lm}{\Rightarrow} S+S*S \underset{lm}{\Rightarrow} 4+S*S \underset{lm}{\Rightarrow} 4+4*S \underset{lm}{\Rightarrow} 4+4*4$$

Here S+S*S is called left-sentential form.

$$S \underset{rm}{\Rightarrow} S+S \underset{rm}{\Rightarrow} S+S*S \underset{rm}{\Rightarrow} S+S*4 \underset{rm}{\Rightarrow} S+4*4 \underset{rm}{\Rightarrow} 4+4*4$$

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.

Review Question

1. *Write in brief about "Sentential form" with reference to context free grammar.*

SPPU : Aug.-17, In Sem, Marks 3

3.4 Context Free Language (CFL)

SPPU : Dec.-05, 06, 07, 08, 09, 10, 13, 14, 17, May-08, 09, 10, 13, 14, 19, Aug.-17, Oct.-16, 18, 19,Marks 12

The language of grammar can be defined by creating the production rules for the given condition. The language generated by context free grammar is called as Context Free Language (CFL).

Let us understand the CFL with the help of various examples.

Example 3.4.1 *Construct the CFG for the language having any number of a's over the set* $\Sigma = \{a\}$.

SPPU : Oct.-19, In Sem, Marks 3

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,

$$S \rightarrow a\,S \qquad \text{rule 1}$$

$$S \rightarrow \varepsilon \qquad \text{rule 1}$$

Now if want "*aaaaa*" string to be derived we can start with start symbols.

S

a S

$a\ a$ S $\quad \because \quad$ rule 1

$a\ a\ a$ S $\quad$ rule 1

$a\ a\ a\ a$ S $\quad$ rule 1

$a\ a\ a\ a\ a$ S $\quad$ rule 1

$a\ a\ a\ a\ a\ \varepsilon$ $\quad$ rule 2

$= a\ a\ a\ a\ a$

The r.e. = a^* suggest a set of {ε, *a*, *aa*, *aaa*, ...}. We can have a null string simply because S is a start symbol, and rule 2 gives S → ε.

Example 3.4.2 *Try to recognize the language L for given CFG.*

$$G = \left[\{S\}, \{a, b\}, P, \{S\}\right]$$

$$\text{where } P = \begin{Bmatrix} S \rightarrow aSb \\ S \rightarrow a\,b \end{Bmatrix}$$

SPPU : Oct.-19, In Sem, Marks 4

Solution : Since S → *a* S b | *a* b is a rule. | indicates the 'or' operator.

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

If this rule can be recursively applied then,

$$\begin{array}{c} S \\ a\ S\ b \\ \downarrow \\ a\ a\ S\ b\ b \\ \downarrow \\ a\ a\ a\ S\ b\ b\ b \end{array}$$

and if finally we can put S → *a*b 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 $\{L = a^n\ b^n \quad \text{where} \quad n \geq 1\}$. 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 3.4.3 *Construct the CFG for the regular expression* $(0+1)^*$.

SPPU : Dec.-05, Marks 8

Solution : The CFG can be given by,

$$\begin{aligned} P = \{ & S \rightarrow 0S \mid 1S \\ & S \rightarrow \varepsilon \} \end{aligned}$$

The rules are in combination of 0's and 1's with the start symbol. Since $(0+1)^*$ indicates {ε, 0, 1, 01, 10, 00, 11, ...} in this set ε is a string. So in the rules we can set the rule S → ε.

Example 3.4.4 *Construct a grammar for the language containing strings of at least two a's.*

Solution : Let G = (V, T, P, S)

where

$$\begin{aligned} V &= \{S, A\} \\ T &= \{a, b\} \\ P &= \{S \rightarrow Aa\ Aa\ A \\ & \quad A \rightarrow aA \mid bA \mid \varepsilon \} \end{aligned}$$

The rule S → A*a*A*a*A is something in which the two *a*'s are maintained since at least two *a*'s should be there in the strings. And A → *a*A |bA|ε gives any combination of *a*'s and b's i.e. this rules 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 3.4.5 *Construct a grammar generating* $L = w\,c\,w^T$ *where* $w \in \{a, b\}^*$

Solution : The strings which can be generated for given L is
{*aa*c*aa*, bcb, *a*bcb*a*, b*aca*b ...}

The grammar could be

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

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

$$S \rightarrow c$$

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 *a*bcb*a* can be generated from given production rules as

S

a S *a*

a b S b

a b c b *a*

a	b	c	b	*a*

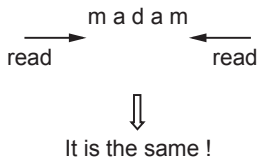
a	b	c	b	*a*

a	b	c	b	*a*

Thus any of this kind of string could be derived from the given production rules.

Example 3.4.6 *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 Σ = {*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 $S \rightarrow a\ S\ a$

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

$$S \rightarrow a$$

$$S \rightarrow b$$

$$S \rightarrow \varepsilon$$

The string abaaba can be derived as

S	
a S *a*	*a* b a a b *a*
a b S b *a*	*a* b *a* *a* b *a*
a b *a* S *a* b	*a* *a* b *a* *a* b *a*
a b *a* ε *a* b	*a* *a* b *a* *a* b *a*
a b *a* *a* b *a*	*a* b *a* *a* b *a*

which is a palindrome.

Example 3.4.7 *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)

Thus r.e. = $(0+1)^*\,000\,(0+1)^*$

Let us build the production rules as

$$S \rightarrow ATA$$
$$A \rightarrow 0A \mid 1A \mid \varepsilon$$
$$T \rightarrow 000$$

Example 3.4.8 *Construct CFG for the language in which there are no consecutive b's, the strings may or may not have consecutive a's.*

Solution : $S \rightarrow a\,S \mid bA \mid a \mid b \mid \varepsilon$

$A \rightarrow a\,S \mid a \mid \varepsilon$

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 → bA, and A gives all the strings which are starting with letter *a*.

Thus G = ({S, A}, {*a*, b}, P, S)

Let us derive the string

Derivation	Input	Productions
S		
a S	*a* *a* b *a* b	S → *a* S
a *a* S	*a* *a* b *a* b	S → *a* S
a *a* b A	*a* *a* b *a* b	S → b A
a *a* b *a* S	*a* *a* b *a* b	A → *a* S
a *a* b *a* b	*a* *a* b *a* b	S → b

Example 3.4.9 *Recognize the context free language for the given CFG.*

$S \rightarrow a\,B \mid b\,A$

$A \rightarrow a \mid a\,S \mid bAA$

$B \rightarrow b \mid b\,S \mid aBB$

SPPU : Aug.-17, In Sem, Marks 3

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 → *a* B	rule 1
S → b A	rule 2
A → *a*	rule 3
A → *a* S	rule 4
A → b A A	rule 5
B → b	rule 6
B → b S	rule 7
B → *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 "*aa*bb*a*bb*a*". 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 "bb*aa*b*a*" 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 3.4.10 *Construct CFG for the language containing at least one occurrence of double a.* **SPPU : Dec.-17, End Sem, Marks 4**

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. $B \rightarrow aB|bB|\varepsilon$ 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.

$$S \rightarrow BAB \quad \Rightarrow \quad (\text{anything}) \begin{pmatrix} \text{one occurrence} \\ \text{of double } a \end{pmatrix} (\text{anything})$$

$$A \rightarrow aa$$

$$B \rightarrow aB|bB|\varepsilon$$

Thus the CFG contains V = {A, B, S} T = {*a*, b}. The start symbol is S.

Let us derive "*abaa*b"

S		
BAB		
*a*BAB	*abaa*b	$B \rightarrow a\ B$
*a*bBAB	*abbaa*b	$B \rightarrow bB$
*a*bAB	*abbaa*b	$B \rightarrow \varepsilon$
*abaa*B	*abbaa*b	$A \rightarrow a\ a$
*abaa*bB	*abbaab*	$B \rightarrow bB$
*abaa*bε	*abbaab*	$B \rightarrow \varepsilon$
= *abaa*b	*abbaab*	

Example 3.4.11 *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.

$$S \rightarrow 0\ A\ 1 \mid 1\ A\ 0$$

$$A \rightarrow 0\ A \mid 1\ A \mid \varepsilon$$

Thus clearly, in the above CFG different start and end symbols are maintained. The non terminal $A \rightarrow 0A|1A|\varepsilon$ indicates (0 + 1)*. Thus the given CFG is equivalent to the regular expression $\left[0(0+1)^*1+1(0+1)^*0\right]$.

Example 3.4.12 *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 3.4.13 *Construct the production rules for defining a language $L = \left\{a^x b^y \mid x \neq y\right\}$.*

SPPU : Dec.-06, Marks 8

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 -

$$S \rightarrow aSb \mid R1 \mid R2$$

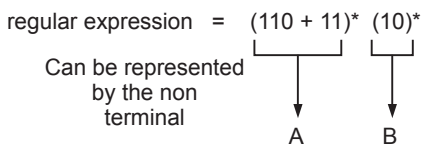
$$R1 \rightarrow aR1 \mid a$$

$$R2 \rightarrow bR2 \mid b$$

where S is a start symbol.

Example 3.4.14 *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

$A \rightarrow 110A \mid 11\ A \mid \varepsilon$

$B \rightarrow 10\ B \mid \varepsilon$

Hence, finally the CFG for given r.e. can be

$S \rightarrow AB$

$A \rightarrow 110A \mid 11A \mid \varepsilon$

$B \rightarrow 10B \mid \varepsilon$

Example 3.4.15 *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 -

$S \rightarrow GI$

$G \rightarrow + \mid -$

$I \rightarrow DI \mid D$

$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \ldots \mid 9$

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

$S \rightarrow GI$	
$-\ I$	$G \rightarrow -$
$-\ DI$	$I \rightarrow DI$
$-\ 2I$	$D \rightarrow 2$
$-\ 2D$	$I \rightarrow D$
$-\ 21$	$D \rightarrow 1$

Example 3.4.16 *Build a CFG for the language*

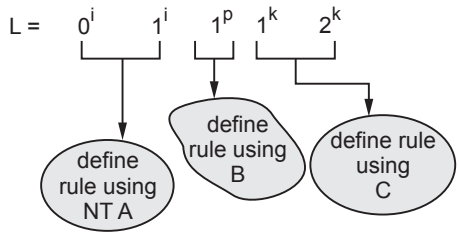
$$L = \left\{0^i\ 1^j\ 2^k \mid j > i + k\right\}.$$

Solution : As the language $L = 0^i\ 1^j 2^k$ such that $j > i + k$.

Hence we can rewrite L as

$$L = 0^i\ \underbrace{1^i\ 1^k\ 1^p}_{\text{This will be greater than } i+k}\ 2^k$$

We can again rewrite L for simplification as -



$A \rightarrow 0\ A\ 1 \mid \varepsilon$

$B \rightarrow 1\ B \mid 1$

$C \rightarrow 1\ C\ 2 \mid \varepsilon$

Hence the CFG can be

$S \rightarrow A\ B\ C$

$A \rightarrow 0\ A\ 1 \mid \varepsilon$

$B \rightarrow 1\ B \mid 1$

$C \rightarrow 1\ C\ 2 \mid \varepsilon$

Example 3.4.17 *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

$\therefore \quad S \rightarrow A\ B$

$A \rightarrow 0\ A\ 1 \mid \varepsilon$

$B \rightarrow 2\ B \mid \varepsilon$

Example 3.4.18 *Obtain CFG for the language* $L = \{0^i\ 1^j\ 2^k \mid j \le k\}$.

Solution : Here $j \le k$. That means the language L has two variations.

$L_1 = 0^i\ 1^j\ 2^j$

$L_2 = 0^i\ 1^j\ 2^k$ where $k > j$.

Hence $L_2 = 0^i\ 1^j\ 2^j \cdot 2^k$

Hence the required CFG can be

$S \rightarrow A\ B$

$A \rightarrow 0\ A \mid \varepsilon$

$B \rightarrow 1\ B\ 2\ C \mid 12$

$C \rightarrow 2\ C \mid \varepsilon$

Example 3.4.19 *Give CFG for set of odd length settings in {0, 1}* with middle symbol '1'.*

SPPU : May-14, Marks 4

Solution : S → 0S0 | 0S1| 1S0| 1S1|1

Consider the string 01110 for derivation

S

0S0

01S10

01110

Example 3.4.20 *If the grammar G is given by production*

S → aSa|bSb|aa|bb|ε show that

- *Any string in L(G) is of length 2n, n > = 0*
- *The number of strings of length 2n is* 2^n

SPPU : Dec.-13, Marks 2

Solution : The CFG denotes the language L of even length in which the two middle symbols equal. Hence it will generate the strings such as

L(G) = { ε, aa, bb, aaaa, aaab, baab, baaa, abba, abbb, bbbb, bbba, …}

- If n = 0, the string is ε.
- If n = 1, the string of length 2 (i.e. 2 × 1) are aa and bb (i.e. 2^n = 2).

Example 3.4.21 *Describe the language generated by each of these grammars. Justify your answer with an example.*

1) S → a S a |b S b | ε *2) S → a S a |b S b | a | b*

3) S → a S b |b S a | ε *4) S → SS |bS| a*

SPPU : Dec.-10, Marks 12

Solution :

1) L = The language containing all strings of even length palindrome.

S	S → aSa
aSa	S → bSb
abSba	S → ε
abba	

2) L = The language containing all strings of odd length palindrome.

S	$S \rightarrow aSab$
aSb	$S \rightarrow bSb$
abSba	$S \rightarrow a$
ababa	

3) L = The language containing all strings which contains equal number of a's and b's. But having different first and last symbol.

S	$S \rightarrow aSb$
aSb	$S \rightarrow bSa$
abSab	$S \rightarrow \varepsilon$
abab	

4) L = The language containing all strings that end with lettter a.

S	$S \rightarrow SS$
SS	$S \rightarrow bS$
bSS	$S \rightarrow a$
baS	$S \rightarrow a$
baa	

Example 3.4.22 *Write a CFG for generating identifiers in higher-level languages such as 'C'. Identifiers can be defined by the regular expression (letter). (letter | digit)*.*

SPPU : May-13, Marks 4

Solution :

$$S \rightarrow LA$$

$$A \rightarrow LA \mid DA \mid \in$$

$$L \rightarrow a|b|c \ldots z$$

$$D \rightarrow 0|1|2 \ldots 9$$

Example 3.4.23 *In each case find context free grammar generating given language*

a) The set of odd length strings in {a, b} with middle symbol a.*

b) The set of even length strings {a, b} with the two middle symbols equal.*

SPPU : Dec.-07, May-08, Marks 8

Solution :

a) Required CFG is -

$S \rightarrow aSa$

$S \rightarrow aSb$

$S \rightarrow bSb$

$S \rightarrow bSb$

$S \rightarrow a$

b) This language can be denoted by regular expression.

regular expression = (a + b)* (aa + bb) (a + b)*

Hence CFG can be given as

$S \rightarrow aSa|aSb|bSb|bSa$

$S \rightarrow aa$

$S \rightarrow bb$

Example 3.4.24 *Describe the language generated by each of these grammars*

a) The regular grammar with production :

$S \rightarrow aA|bC|b$

$A \rightarrow aS|bB$

$B \rightarrow aC|bA|a$

$C \rightarrow aB|bS$

b) The regular grammar with production :

$S \rightarrow bS|aA|\varepsilon$

$A \rightarrow aA|bB|b$

$B \rightarrow bS$

SPPU : Dec.-07, 09, May-08, 09, Marks 8

Solution :

a) For obtaining the language generated by given CFG we will first draw FA for it.

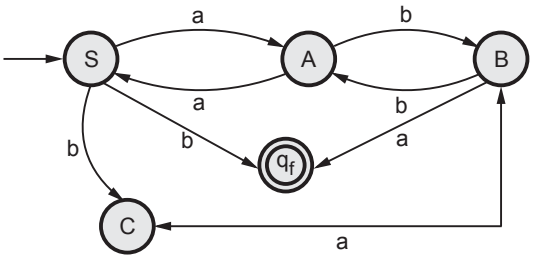


Fig. 3.4.1

We can merge q_f and state C. Hence the FA will be.

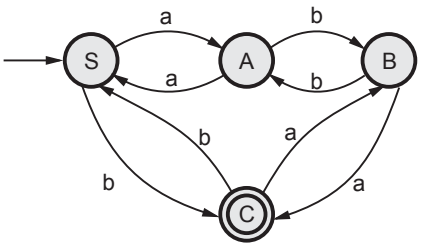


Fig. 3.4.2

We can derive some strings from above drawn FA. Those are {aab, bbb, b, aba, ...}

That means given language L accepts the strings containing even number of a's and odd number of b's.

b) For obtaining the language generated by given CFG we will first draw FA for it.

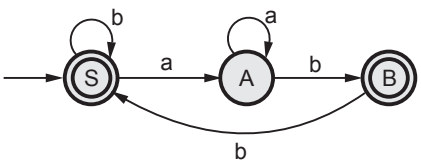


Fig. 3.4.3

The r.e. $= \left(a^* aa^* b\ (\varepsilon + b)\right)^* + b^*$

This regular expression denotes the strings made up of either any number of b's or the strings that contain the substring 'ab' but not contain the substrings 'aba' at all.

Example 3.4.25 *Give the context free grammars for the following languages*

a) $L = \left\{0^m\ 1^n\ 0^{m+n} \mid m, n >= 0\right\}$ *b)* $L = \left\{0^a\ 1^b\ 2^c \mid |a - c| = b\right\}$

SPPU : Dec.-08, Marks 12

Solution :

a) Let, $L = \left\{0^m\ 1^n\ 0^{m+n}\right\}$

We will rewrite L as

$= 0^m\ 1^n\ 0^n\ 0^m \qquad \because n + m = m + n$

Hence P = {

$S \rightarrow 0S0 \mid 0100$

$S \rightarrow 01A00$

$A \rightarrow 1A0 \mid \varepsilon$

}

Consider the derivation for string 001000 as

S	
0S0	$S \rightarrow 0S0$
001000	$S \rightarrow 0100$

b) Let, $L = \{0^a 1^b\ 2^c \mid a - c \mid = b\}$

If we arrange $a = |b+c|$ then we can rewrite L as

$$= 0^{b+c}\ 1^b\ 2^c$$

$$= 0^c 0^b 1^b 2^c$$

Hence CFG can be

$$S \rightarrow 0S2 | 0012$$

$$S \rightarrow 00A12$$

$$A \rightarrow 0A1 | \varepsilon$$

Let us derive the string 000000112222 as follows -

S	$S \rightarrow 0S2$
0S2	$S \rightarrow 0S2$
00S22	$S \rightarrow 0S2$
000S222	$S \rightarrow 00A12$
00000A12222	$A \rightarrow 0A1$
000000A112222	$A \rightarrow \varepsilon$
000000112222	

i.e. $0^6 1^2 2^4 \in \{L = 0^a 1^b 2^c \mid |a - c| = b\}$

Example 3.4.26 *Give the context free grammar for the following languages*

a) $L = \{a^n b^{2n} | n>1\}$ *b)* $L = \{a^m b^n | n>m\}$

SPPU : Dec.-08, Marks 12

Solution :

a) Let, $L = \{a^n b^{2n} | n>1\}$

The production rules for the CFG of above language will be -

$$S \rightarrow aAbb$$

$$A \rightarrow aAbb | abb$$

b) Let, $L = \{a^m b^n \mid n > m\}$

The L can be written as $L = a^m b^m b^K$ where $K \geq 1$.

Hence required CFG will be

$$S \rightarrow aSbB \mid \varepsilon$$

$$B \rightarrow bB \mid b$$

Example 3.4.27 *Give the context free grammars for the following languages.*

a) (011 + 1) (01)**

b) $0^i 1^{i+k} 0^k$ *where i, k > = 0.* **SPPU : May-09, Marks 12, May-14, Marks 8**

Solution : a) The CFG can be denoted by,

$$G = (\{S, A, B, C, D\}, \{0, 1\}, P, S)$$

where $P = \{$

$$S \rightarrow AB$$

$$A \rightarrow CA \mid \varepsilon$$

$$C \rightarrow 011 \mid 1$$

$$B \rightarrow DB \mid \varepsilon$$

$$D \rightarrow 01$$

$\}$

Let us derive a valid string 01110101 $\in$ (011 + 1)* (01)*

S

AB

CAB

CCAB

CCADB

CCADDB

011CADDB

0111ADDB

0111DDB

011101DB

01110101B

0111101

b) Let us rewrite the language L as

$$L = 0^i 1^{i+k} 0^k$$

$$L = 0^i 1^i 1^k 0^k$$

Hence required CFG G = ({S, A, B}, {0, 1}, P, S)

Hence

$$P = \{S \rightarrow AB$$
$$= A \rightarrow 0A1 \mid \varepsilon$$
$$= B \rightarrow 1B0 \mid \varepsilon$$
$$\}$$

Example 3.4.28 *Describe the language generated by each of these grammar and justify your answer with the example string derive from the grammar of the productions given below.*

1) $S \rightarrow aSa \mid bSb \mid aAb \mid bAa$

$A \rightarrow aAa \mid bAb \mid a \mid b \mid \varepsilon$

2) $S \rightarrow bT \mid aT \mid \varepsilon$

$T \rightarrow aS \mid bS$

SPPU : May-10, Marks 12

Solution :

1) This is a language which is a combination of two language L_1 and L_2, where L_1 denotes all the strings that are palindrome.

L_2 denotes all the strings containing the substrings as palindrome but their first and last letter are different.

i.e. L_2 = [a (Palindrome string) b + b (palindrome string) a]

2) To derive the string for this CFG we will draw a FA

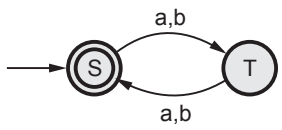


Fig. 3.4.4

Clearly, this FA suggest a language containing all the strings having even length.

Example 3.4.29 *Find the context free grammar for the following languages.*

i) $L = \{a^n b^m \mid n \neq m\}$

ii) $\{a^n b^m c^k \mid n = m \text{ or } m \leq k\}$

SPPU : Dec.-06, Marks 8

Solution : i) $\mathbf{L = \{a^n b^m \mid n \neq m\}}$ **:** Refer example 3.4.26 (b).

ii) $\mathbf{\{a^n b^m c^k \mid n = m \text{ or } m \leq k\}}$:

Here L is a combination of L_1 and L_2 where

$$L_1 = \{a^n b^m c^k \mid n = m\}$$
$$L_2 = \{a^n b^m c^k \mid m \leq k\}$$

The CFG for L_1 can be

$$S \rightarrow AB$$

$$A \rightarrow a\,A\,b \mid \varepsilon$$

$$B \rightarrow c\,B \mid \varepsilon$$

The CFG for L_2 can be

$$S \rightarrow PQ$$

$$P \rightarrow aP \mid \varepsilon$$

$$Q \rightarrow b\,Q\,c\,R \mid b\,c$$

$$R \rightarrow c\,R \mid \varepsilon$$

Hence the CFG for L will be

$$S \rightarrow XY$$

$$X \rightarrow AB$$

$$A \rightarrow a\,A\,b \mid \varepsilon$$

$$B \rightarrow c\,B \mid \varepsilon$$

$$Y \rightarrow PQ$$

$$P \rightarrow a\,P \mid \varepsilon$$

$$Q \rightarrow b\,Q\,c\,R \mid b\,c$$

$$R \rightarrow c\,R \mid \varepsilon$$

Example 3.4.30 *Give context free grammars for the following languages.*

i) L = {S → aAb, A → aA|bA|ε, x|x ε {a, b} with strings of starting with 'a' and ending 'b'}*

ii) L = {x|x ε {a, b} with strings of even length palindrome}.*

SPPU : Dec.-14, In Sem, Marks 4

Solution : i) S → aAb, A → aA|bA|ε.

ii)

Step 1 : As we know the strings are palindrome if they possess same alphabets from forward as well as from backward.

Step 2 : Let G = ({S}, {*a*, b}, P, S)

Step 3 : The string abaaba can be derived as

P can be $S \rightarrow a\,S\,a$ S

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

S → *a*	*a* b S b *a*	*a* b *a a* b *a*
S → b	*a* b *a* S *a* b	*a a* b *a a* b *a*
S → ε	*a* b *a* ε *a* b	*a a* b *a a* b *a*
	a b *a a* b *a*	*a* b *a a* b *a*

Example 3.4.31 *Give context free grammars for the following languages :*

i) $L = \{x \mid x \in (,)\}^*$ *with strings having well-formed parentheses (WFP)*

ii) $L = \{a^m b^n c^{m+n} \mid m, n >= 0\}$ **SPPU : Oct.-16, In sem, Marks 6**

Solution :

i) The production rule are

P = {

S → ( ) S | ( ) S | (S) | ( )

}

ii) We will first rewrite the given L as

$L = a^m b^n c^n c^m$

Hence P = {

S → aSc | abcc

S → abAcc

A → bAc | ε

}

Consider the derivation for string

aabccc

S

aSc

aabccc

Example 3.4.32 *Write context free grammar for the following language 0 (0 + 1)* 01 (0 + 1)* 1*

SPPU : Aug.-17, In sem, Marks 4

Solution :

P → {

S → 0A01A1

A → 0A | 1A | ε

}

Example 3.4.33 *Write the CFG for following language.*

$L = \{a^{m+n} b^m c^n \mid n, m >= 0\}$ **SPPU : Oct.-18, In Sem, Marks 4**

Solution : We will rewrite language L as

$$L = a^n a^m b^m c^n$$

Hence $P = \{$

$S \rightarrow aSc \mid abcc$

$S \rightarrow abAcc$

$A \rightarrow bAc \mid \varepsilon$

$\}$

Example 3.4.34 *Write the grammar generating all strings consisting of a's and b's with at least two a's.* **SPPU : Oct.-18, In Sem, Marks 2**

Solution : We will write regular expression for this language

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

The production rules $(a + b)^*$

$A \rightarrow aA \mid bA \mid \varepsilon$

Hence production rules for regular expression

$S \rightarrow A \times A \times A$

$X \rightarrow a$

$A \rightarrow aA \mid bA \mid \varepsilon$

Example 3.4.35 *Write CFG for following language.*

$L = \{a^n b^m a^n \mid n >= 0, \; m >= 1\}$ **SPPU : Oct.-18, In Sem, Marks 4**

Solution : The CFG can be given by following production rule.

$P = \{$

$S \rightarrow aSa \mid bB$

$B \rightarrow bB \mid \varepsilon$

$\}$

Simulation :

The string aabbbaa can be derived as

S

aSa

aaSaa

aabBaa

aabbBaa

aabbbBaa

aabbbaa

Example 3.4.36 *Write CFG for the following languages.*

i) $L = \{0^i 1^j 0^k \mid j > i+k\}$ **(Refer example 3.4.16)**

ii) $L = \{0^i 1^j 2^k \mid i = j+k\}$ **SPPU : May-19, End Sem, Marks 4**

Solution : ii) Let,

$$L = \{0^i\ 1^j\ 2^k \mid i = j+k\}$$

We can rewrite L as

$$L = 0^j\ 0^k\ 1^j\ 2^k$$
$$= 0^k\ 0^j\ 1^j\ 2^k$$

The production rules are

$$S \rightarrow 0X_2 \mid 0\,2$$
$$X \rightarrow 0X_1 \mid \varepsilon$$

3.5 Derivation

SPPU : May-15, Marks 4

- 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

$S \rightarrow XYX$	$S \rightarrow XYX$
$S \rightarrow aYX$	$S \rightarrow XYa$
$S \rightarrow abX$	$S \rightarrow Xba$
$S \rightarrow aba$	$S \rightarrow aba$
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.

Example 3.5.1 *Derive the string "aabbabba" for leftmost derivation and rightmost derivation using a CFG given by,*

$$S \rightarrow aB \mid bA$$
$$A \rightarrow a \mid aS \mid bAA$$
$$B \rightarrow b \mid bS \mid aBB$$

Solution : Let us see the leftmost derivation first,

S	
*a*B	S → *a*B
*aa*BB	B → *a*BB
*aa*bB	B → b
*aa*bbS	B → bS
*aa*bb*a*B	S → *a*B
*aa*bb*a*bS	B → bS
*aa*bb*a*bbA	S → bA
*aa*bb*a*bb*a*	A → *a*

is derived. Now let us solve using rightmost derivation.

S	
*a*B	S → *a*B
*aa*BB	B → *a*BB
*aa*BbS	B → bS
*aa*BbbA	S → bA
*aa*Bbb*a*	A → *a*
*aa*bSbb*a*	B → bS
*aa*bbAbb*a*	S → bA
*aa*bb*a*bb*a*	A → *a*

is a derivation.

Example 3.5.2 *Derive the string 1000111 for leftmost and rightmost derivation using CFG.*

$G = (V, T, P, S)$ *where*

$V = \{S, T\}$

$T = \{0, 1\}$

$P = \{ S \rightarrow T00T$

$T \rightarrow 0T \mid 1T \mid \varepsilon \}$

Solution : The leftmost derivation can be

S	
T00T	$S \rightarrow T00T$
1T00T	$T \rightarrow 1T$
10T00T	$T \rightarrow 0T$
10 ε 00T	$T \rightarrow \varepsilon$
1000T	
10001T	$T \rightarrow 1T$
100011T	$T \rightarrow 1T$
1000111T	$T \rightarrow 1T$
1000111 ε	$T \rightarrow \varepsilon$
1000111	

Rightmost derivation -

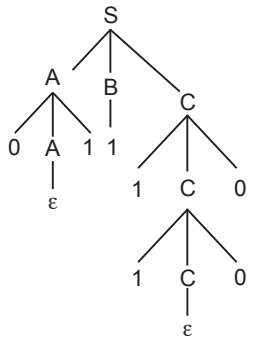
S	
T00T	
T001T	$T \rightarrow 1T$
T0011T	$T \rightarrow 1T$
T00111T	$T \rightarrow 1T$
T00111 ε	$T \rightarrow \varepsilon$
T00111	
1T00111	$T \rightarrow 1T$
10T00111	$T \rightarrow 0T$
10 ε 00111	$T \rightarrow \varepsilon$
1000111	

Example 3.5.3 *Write the CFG for language* $L = \{0^i 1^j 0^k \mid j > i + k\}$. *Show the derivation of the string '0111100'.* **SPPU : May-15, End Sem, Marks 4**

Solution : The CFG can be

$$S \rightarrow ABC$$
$$A \rightarrow 0A1 \mid \varepsilon$$
$$B \rightarrow 1B \mid 1$$
$$C \rightarrow 1C0 \mid \varepsilon$$

Derivation of 0111100



3.6 Parse Tree

Parse 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 parse tree is also called derivation tree.

Properties of Parse Tree

Following are properties of any parse 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

$S \rightarrow bSb|a|b$ is a production rule. The S is a start symbol.

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,

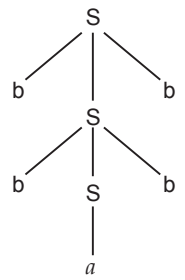
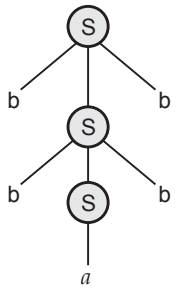


Fig. 3.6.1 Parse tree



Problems on parse tree

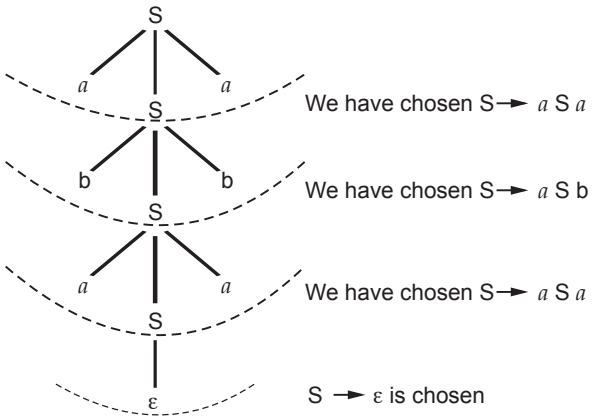
Example 3.6.1 *Draw a parse tree for the string abaaba for the CFG given by,*

G where $P = \{S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow a|b|\varepsilon\}$

Solution :



Example 3.6.2 *Construct the parse tree for the string aabbabba from the CFG given by*

$S \rightarrow aB|bA$

$A \rightarrow a|aS|bAA$

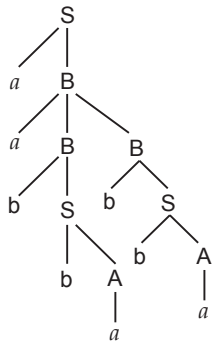
$B \rightarrow b|bS|aBB$

Solution : To draw a tree we will first try to obtain derivation for the string *aa*bb*a*bb*a*

S	
*a*B	S → *a*B
a $\boxed{a\text{BB}}$	S → *a*BB
aa $\boxed{\text{bS}}$ B	B → bS
*aa*b $\boxed{\text{bA}}$ B	S → bA

*aa*bb *a* $\boxed{\text{B}}$	A → *a*
*aa*bb*a* $\boxed{\text{bS}}$	B → bS
*aa*bb*a*b $\boxed{\text{bA}}$	S → bA
*aa*Bb*a*bb $\boxed{a}$	A → *a*

Now let us draw a tree.



3.7 Ambiguous Grammar

SPPU : May-07, 11, 12, 14, Dec.-05, 12, 13, 14, Marks 8

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.

- **Definition**

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.

- **Example**

For example : The CFG given by G = (V, T, P, S)

where V = {E}

T = {id}

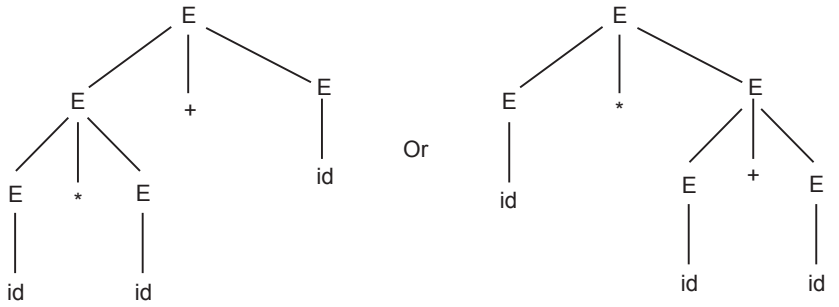
P = {E → E + E

E → E * E

E → id }

S = {E}

Now if the string is id * id + id then we can draw the two different parse trees indicating our id * id + id.



Thus the above grammar is an ambiguous grammar. Let us solve some exercise on it.

Problems on ambiguity of grammar

Example 3.7.1 *Check whether the given grammar is ambiguous or not.*

$S \rightarrow iC\ t\ S$

$S \rightarrow iC\ t\ S\ e\ S$

$S \rightarrow a$

$C \rightarrow b$

Solution : To check whether given grammar is ambiguous or not we will have some string for derivation tree such as ibtibtibt*a*e*a*.

Let us draw the derivation tree, for the string ibtibtibt*a*e*a*

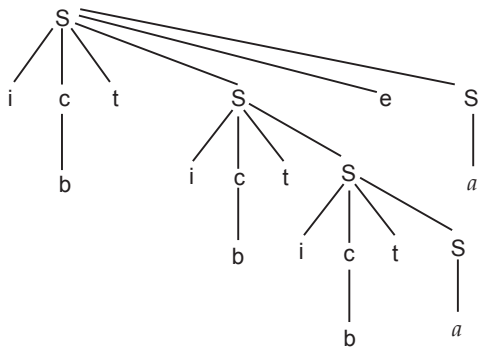


Fig. 3.7.1 (a) For example 3.7.1

or

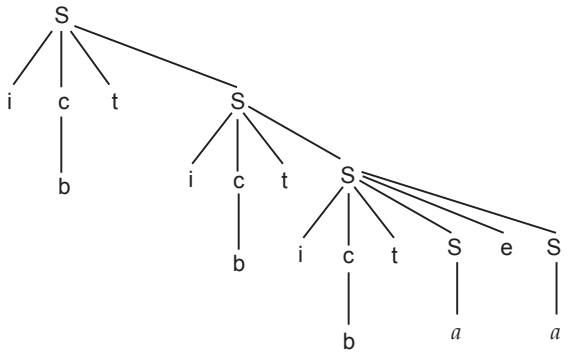


Fig. 3.7.1 (b) For example 3.7.1

Thus we have got more than two parse trees. Hence the given grammar is ambiguous.

Example 3.7.2 *Consider the grammar* $G = (V, \Sigma, R, S)$.

where $V = \{S, A\}$

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

$R = \{ S \rightarrow AA$

$A \rightarrow AAA$

$A \rightarrow a$

$A \rightarrow bA$

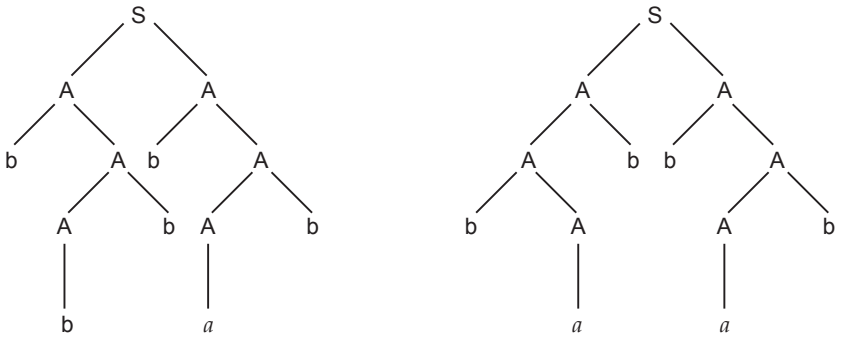
$A \rightarrow Ab \}$

Show that this is an ambiguous grammar

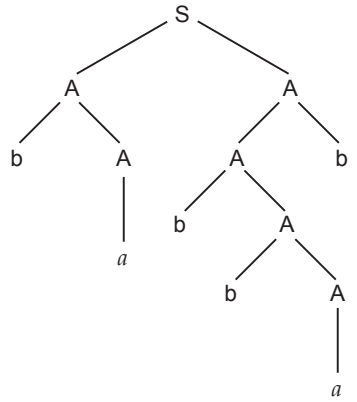
Solution : Consider a string b*a*bb*a*b.

Refer Fig. 3.7.2.

Thus there are more than one parse tree getting generated. Hence the grammar is ambiguous.



(a) **(b)**



(c)

Fig. 3.7.2 For example 3.7.2

3.7.1 Inherent Ambiguity

- **Definition :**

A context free grammar is called inherently ambiguous if all the production rules in this grammar are ambiguous.

- **Example :**

For example -

$S \rightarrow XYZ \mid aaYbb$

$X \rightarrow aaY \mid aa$

$Y \rightarrow baZ \mid ba$

$Z \rightarrow bZb \mid bb$

Then the parse trees for the string "aababb" can be as shown below -

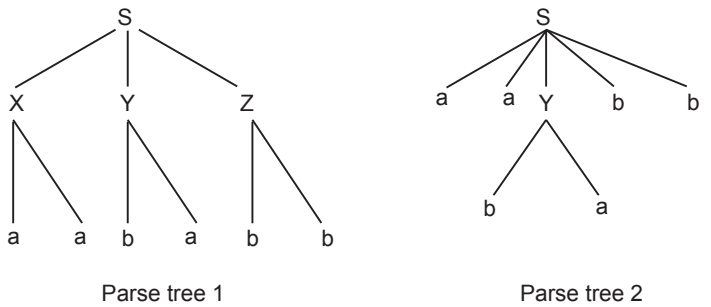


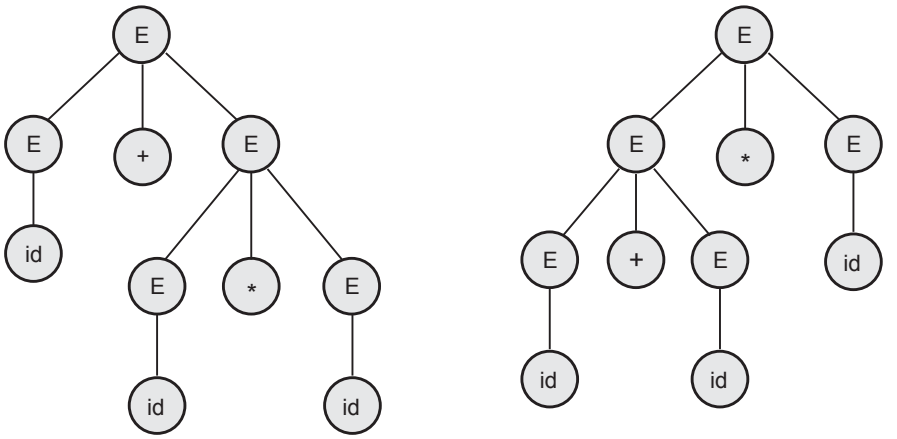
Fig. 3.7.3 Derivation trees for inherent grammar

Note that in above context free grammar there are two productions for each rule this makes every rule of the grammar as ambiguous.

3.7.2 Removal of Ambiguity

Consider the ambiguous grammar as -

$E \rightarrow E + E \mid E * E \mid id$



(a) Parse tree 1 (b) Parse tree 2

Fig. 3.7.4 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 3.7.3 *Give an ambiguous grammar for if then else statement and then re-write an equivalent unambiguous grammar.* **SPPU : May-14, Marks 8**

Solution : The ambiguous grammar for **if then else** statement is as follows -

$S \rightarrow iCtS$

$S \rightarrow iCtSeS$

$S \rightarrow a$

$S \rightarrow b$

Where i stands for **if**, C stands for **condition**, S stands for statement, e stands for **else**, and t stands for **then**. For making the given grammar unambiguous we match **else** with closest previous **unmatched then**. Hence on removing ambiguity we will get following rules -

$$S \rightarrow M \mid U$$

$$M \rightarrow iCtMeM \mid a$$

$$U \rightarrow iCtS \mid iCtMeU$$

$$C \rightarrow b$$

Here M and U non-terminals for matched and unmatched statments.

Example 3.7.4 *Consider the grammar* $G = (V = \{E, F\}, T = \{a, b, -\}, E, P)$

Where P consist of rules :

$E \rightarrow F - E, F \rightarrow a, E \rightarrow E - F, F \rightarrow b, E \rightarrow F$

1) Show that G is ambiguous

2) Remove the ambiguity, if possible of G. **SPPU : May-11, Marks 6**

Solution :

Let,

$$E \rightarrow F - E$$

$$F \rightarrow a$$

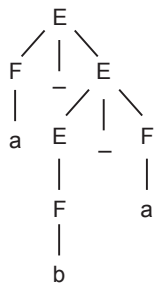
$$E \rightarrow E - F$$

$$F \rightarrow b$$

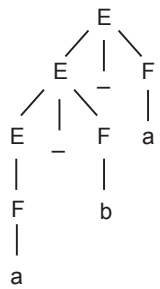
$$E \rightarrow F$$

are the production rules.

1) Consider the string a-b-a



Parse tree 1



Parse tree 2

2) The unambiguous grammar will be

$$E \rightarrow E - F \mid F$$

$$F \rightarrow a \mid b$$

Example 3.7.5 *Given a CFG as :*

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid a\ BB$

Is ab, baba, abbbaa in L (G) ? What is the language of this CFG ? Is the CFG ambiguous ? **SPPU : May-12, Marks 8**

Solution : The {ab, baba, abbbaa}∈ L (G). The language of this CFG is L which contains equal number of a's and b's. The given CFG is ambiguous because consider the string bbabaa.

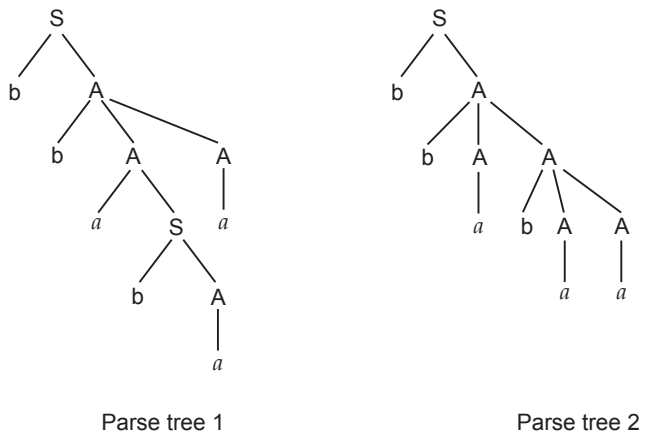


Fig. 3.7.5

Example 3.7.6 *Show that following grammar is ambiguous :*

$S \rightarrow a \mid absb \mid a\ A\ b$

$A \rightarrow bs \mid a\ A\ A\ b$ **SPPU : Dec.-12, Marks 4**

Solution : Consider the string absb. Try to find left and right derivation tree.

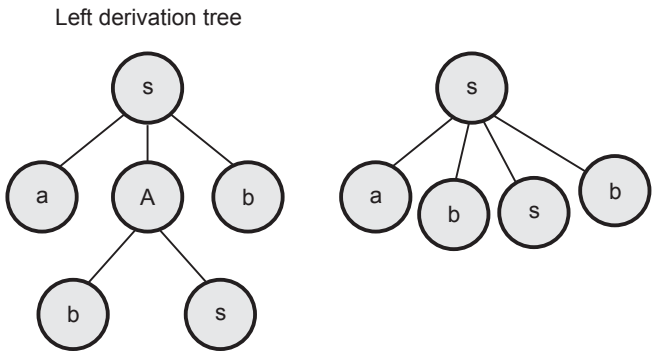


Fig. 3.7.6

For same string we have two different derivation tree so given grammar is ambiguous.

Example 3.7.7 *Consider the grammar having production.*

$S \rightarrow aS \mid \varepsilon$

$S \rightarrow aSbS$

This grammar is ambiguous.

i) Show in particular that the string aab has two parse trees.

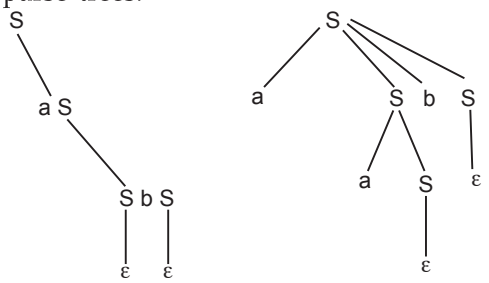
ii) Find an unambiguous grammar for the same. **SPPU : Dec.-05, Marks 8**

Solution : Given grammar with production

$S \rightarrow aS \mid \varepsilon$

$S \rightarrow aSbS.$

i) a a b has following parse trees.



(a) **(b)**

Fig. 3.7.7

unambiguous grammar for above is

$S \rightarrow aS \mid XS \mid \varepsilon$

$X \rightarrow aXXb \mid \varepsilon$

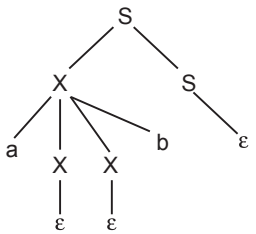


Fig. 3.7.7 (c)

Example 3.7.8 *Let G be the grammar*

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

For the string aaabbabbba find

1) Leftmost derivation *2) Rightmost derivation*

3) Parse tree *4) Is the grammar unambiguous*

SPPU : May-07, Marks 8; Dec.-14, End Sem, Marks 4

Solution : 1) Leftmost derivation :

$$
\begin{aligned}
S &\rightarrow aB \\
&\rightarrow aaBB \\
&\rightarrow aaaBBB \\
&\rightarrow aaabSBB \\
&\rightarrow aaabbABB \\
&\rightarrow aaabbaBB \\
&\rightarrow aaabbababB \\
&\rightarrow aaabbabbS \\
&\rightarrow aaabbabbbA \\
&\rightarrow aaabbabbba.
\end{aligned}
$$

2) Rightmost derivation :

$$
\begin{aligned}
S &\rightarrow aB \\
&\rightarrow aaBB \\
&\rightarrow aaBbS \\
&\rightarrow aaBbbA \\
&\rightarrow aaBbba \\
&\rightarrow aaaBBbba \\
&\rightarrow aaaBbbba \\
&\rightarrow aaabSbbba \\
&\rightarrow aaabbAbbba \\
&\rightarrow aaabbabbba.
\end{aligned}
$$

3) Parse tree :

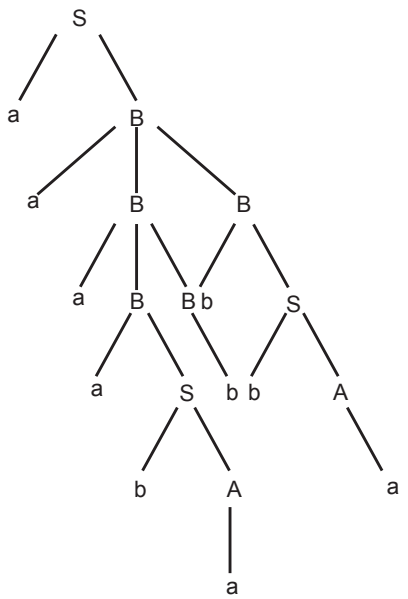
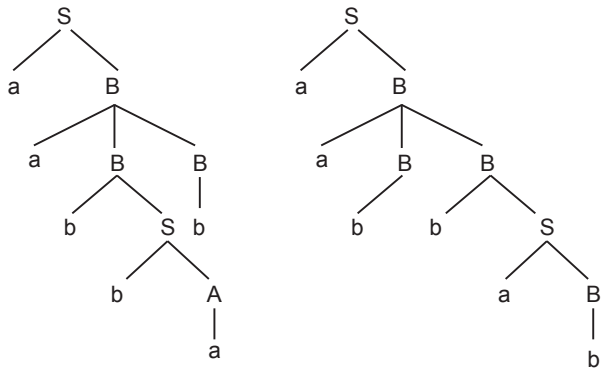


Fig. 3.7.8

4) No, the grammar is ambiguous. Following derivations for **aabbab** shows -



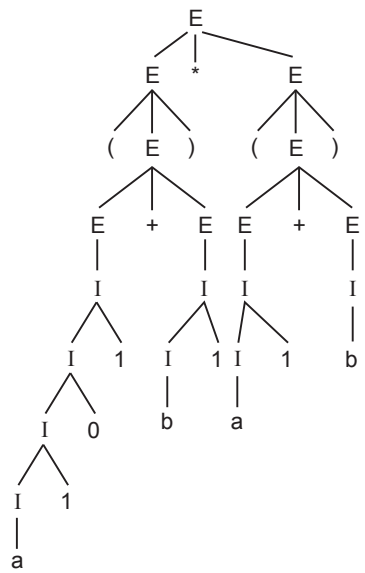
Example 3.7.9 *Derive (a101 + b1) * (a1 + b) using leftmost and rightmost derivation where grammar is given as*
*E → I, E → E + E, E → E * E, E → (E), I → a,*
I → b, I → Ia, I → Ib, I → I0, I → I1.
Test whether the grammar is ambiguous. **SPPU : Dec.-13, Marks 6**

Solution :

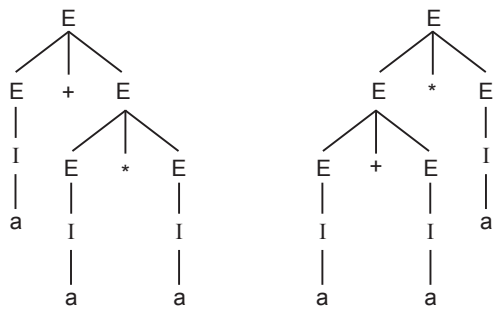
Leftmost derivation	Rightmost derivation
E	E
E ∗ E	E ∗ E
(E) ∗ E	E ∗ (E)
(E + E) ∗ E	E ∗ (E + E)
(I + E) ∗ E	E ∗ (E + I)
(I1 + E) ∗ E	E ∗ (E + b)
(I01 + E) ∗ E	E ∗ (I + b)
(I101 + E) ∗ E	E ∗ (I1 + b)
(a101 + E) ∗ E	E ∗ (a1 + b)
(a101 + I) ∗ E	(E) ∗ (a1 + b)
(a101 + I1) ∗ E	(E + E) ∗ (a1 + b)
(a101 + b1) ∗ E	(E + I) ∗ (a1 + b)
(a101 + b1) ∗ (E)	(E + I1) ∗ (a1 + b)
(a101 + b1) ∗ (E + E)	(E + b1) ∗ (a1 + b)

(a101 + b1) ∗ (I + E)	(I + b1) ∗ (a1 + b)
(a101 + b1) ∗ (I1 + E)	(I1 + b1) ∗ (a1 + b)
(a101 + b1) ∗ (a1 + E)	(I01 + b1) ∗ (a1 + b)
(a101 + b1) ∗ (a1 + I)	(I101 + b1) ∗ (a1 + b)
(a101 + b1) ∗ (a1 + b)	**(a101 + b1) ∗ (a1 + b)**

The derivation is as follows -



The grammar is ambiguous. Consider the derivation for the string **a + a ∗ a**.



Parse tree 1 **Parse tree 2**

As there are more than one parse trees for deriving the given string the grammar is said to be ambiguous.

3.8 Simplification of CFG

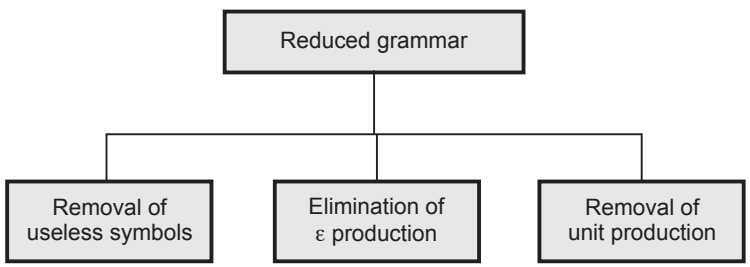
SPPU : Dec.-16, 18, 19, Aug.-17, Oct.-18, 19, Marks 8

As we have seen various languages can effectively be represented by context tree grammar. All the grammars are not always optimized. That means grammar may consists 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.

Properties of reduced grammar -

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 $X \rightarrow \varepsilon$.

We see the reduction of grammar as below.



Let us study the reduction process in detail.

3.8.1 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.

$$S \overset{*}{\Rightarrow} \alpha p \beta$$

and $$\alpha p \beta \overset{*}{\Rightarrow} w$$

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. Let us see what exactly means the useless symbol with some example.

For example, G = (V, T, P, S) where V = {S, T, X}, T = {0, 1}

$S \rightarrow 0T|1T|X|0|1$ rule 1

$T \rightarrow 00$ rule 2

Now, in the above CFG, the non terminals are S, T and X.

To derive some string we have to start from start symbol S.

S

0T $S \rightarrow 0T$

000 $T \rightarrow 00$

Thus we can reach to certain string after following these rules.

But if $S \rightarrow X$ then there is no further rule as a definition to X. That means there is no point in the rule $S \rightarrow X$. Hence we can declare that X is a useless symbol. And we can remove this so after removal of this useless symbol CFG becomes

G = (V, T, P, S) where V = {S, T}

T = {0, 1} and P = { $S \rightarrow 0T|1T|0|1$

$T \rightarrow 00$ }

S is a start symbol.

Problems on removal of useless symbol

Example 3.8.1 *Consider the CFG G = (V, T, P, S)*

where V = {S, A, B} T = {0, 1}

P = { $S \rightarrow A\ 11\ B|\ 11A$

$S \rightarrow 1\ B|\ 11$

$A \rightarrow 0$

$B \rightarrow BB$ } 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,

$S \rightarrow 11\ A\ |\ 11$

$A \rightarrow 0$

Example 3.8.2 *Find CFG with no useless symbols equivalent to*

$S \rightarrow AB\ |\ CA$ $B \rightarrow BC\ |\ AB$

$A \rightarrow a$ $C \rightarrow aB\ |\ b$

Solution : Consider, the rule

$$S \rightarrow AB \qquad 1$$

$$S \rightarrow CA \qquad 2$$

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. : $S \rightarrow AB \rightarrow aAB \rightarrow aaAB \rightarrow aaaAB$ and so on.

Thus we come to know as B is an useless symbol. Removing B the rules are now,

$$S \rightarrow CA$$

$$A \rightarrow a$$

$$C \rightarrow b$$

Example 3.8.3 *Consider the following CFG*

$G = (V, \Sigma, R, S)$

where $V = \{S, X, Y\}$

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

$R = \{S \rightarrow XY \mid 0$

$X \rightarrow 1\}$

Remove the useless symbols from it.

Solution : As

$$S \rightarrow XY$$

$$S \rightarrow 0$$

$$X \rightarrow 1$$

It is easy to recognize that there is no derivation for Y. Hence we can eliminate the production with Y, and the rules are

$$S \rightarrow 0$$

$$X \rightarrow 1$$

Example 3.8.4 *Remove useless symbols from the following grammar.*

$S \rightarrow aA \mid bB$

$A \rightarrow aA \mid a$

$B \rightarrow bB$

$D \rightarrow ab \mid Ea$

$E \rightarrow aC \mid d$

SPPU : Dec.-18, End Sem, Marks 6

Solution : We will first find all the productions which are giving terminal symbols.

$$A \rightarrow a$$

$$D \rightarrow ab$$

$$E \rightarrow d$$

Now consider start symbol

$$S \rightarrow aA \mid bB$$

Now since B → bB we will be continuously in a loop with B productions. Thus B is an useless symbol. So we will eliminate it. Then the rules are

$$S \rightarrow aA$$
$$A \rightarrow a \mid aA$$
$$D \rightarrow ab \mid Ea$$
$$E \rightarrow aC \mid d$$

Again if we look at E productions there is a production E → aC. But there is no rule for C. Hence we will remove the production
E → aC.

Now,

$$S \rightarrow aA$$
$$A \rightarrow aA \mid a$$
$$D \rightarrow ab \mid Ea$$
$$E \rightarrow a$$

Now the rules S and A indicates that there is no D and E in the derivation. Again we get D and E as useless symbols. So we will eliminate D and E. Finally after removal of all useless symbols the production rules are ,

$$S \rightarrow aA$$

$$A \rightarrow aA \mid a$$

Example 3.8.5 *Eliminate the useless symbols from the following grammar.*

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

$$A \rightarrow a$$

$$B \rightarrow aa$$

$$C \rightarrow aCb$$

Solution : We will first write all the productions that are giving terminal symbols.

$$A \rightarrow a$$

$$B \rightarrow aa$$

But if we look at start symbol S then,

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

There is no production for B from start symbol. Thus B becomes a useless symbol.

Similarly,

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

There is no terminating symbol for C. Therefore we will eliminate C. Then set of rules are

$$S \rightarrow aS \mid A$$

$$A \rightarrow a$$

Thus we obtain the grammar having these two rules after removal of useless symbols B, C.

Example 3.8.6 *Eliminate the useless symbols from following grammar.*

$S \rightarrow aA \mid a \mid Bb \mid cC$

$A \rightarrow aB$

$B \rightarrow a \mid Aa$

$C \rightarrow cCD$

$D \rightarrow ddd$

Solution : Consider all the productions that are giving terminal symbols.

$$S \rightarrow a$$

$$B \rightarrow a$$

$$D \rightarrow ddd$$

Now consider

$$S \rightarrow cC$$

$$C \rightarrow cCD$$

$$D \rightarrow ddd.$$

When we try to derive the string using production rule for C we get,

$S \rightarrow cC \rightarrow ccCD \rightarrow ccCddd \rightarrow cccCDddd \rightarrow cccCdddddd \rightarrow \ldots$

We will not get any terminal symbol for C. Thus we get a useless symbol C. To reach to D the only rule available is by using C. But as C gets eliminated, there is no point in keeping D. Hence D also will be removed.

$\therefore$

$$S \rightarrow aA \mid a \mid Bb$$

$$A \rightarrow aB$$

$$B \rightarrow a \mid Aa$$

is the reduced grammar.

3.8.2 Elimination of ε Productions from Grammar

As we have seen in finite automata and regular expression that ε or a null string indicates a string with no value. We have also seen that even though some NFA contains ε moves we can convert that NFA without ε moves. Even 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,

If $S \rightarrow 0S|1S|\varepsilon$

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 $S \rightarrow \varepsilon$ in other rules we get $S \rightarrow 0$ when $S \rightarrow 0\ S$ and $S \rightarrow \varepsilon$ as well as $S \rightarrow 1$ when $S \rightarrow 1\ S$ and $S \rightarrow \varepsilon$

Hence we can rewrite the rules as

$$S \rightarrow 0S|1S|0|1$$

Thus ε production is removed.

Problems on removal of ε production

Example 3.8.7 *Remove the ε production from following CFG by preserving meaning of it.*

$S \rightarrow XYX$

$X \rightarrow 0X|\varepsilon$

$Y \rightarrow 1\ Y|\varepsilon$

Solution : Now, while removing ε production we are deleting the rules $X \rightarrow \varepsilon$ and $Y \rightarrow \varepsilon$. To preserve the meaning of CFG we are actually placing ε at right hand side wherever X and Y have appeared.

Let us take

$$S \rightarrow XYX$$

if first X at right hand side is ε.

Then $S \rightarrow YX$

Similarly if last X in R.H.S. = ε.

Then $S \rightarrow XY$

If $Y = \varepsilon$ then

$$S \rightarrow XX$$

If Y and X are ε then,

$$S \rightarrow X \quad \text{also}$$

$$S \rightarrow Y \quad \text{when both X are replaced by } \varepsilon$$

$$\therefore \quad S \rightarrow XY|YX|XX|X|Y$$

Now let us consider

$$X \rightarrow 0X$$

If we place ε at right hand side for X then,

$$X \rightarrow 0$$

$$X \rightarrow 0X|0$$

Similarly $Y \rightarrow 1\ Y|1$

Collectively we can rewrite the CFG with removed ε productions as

$$S \rightarrow XY|YX|XX|X|Y$$

$$X \rightarrow 0X|0$$

$$Y \rightarrow 1Y|1$$

Example 3.8.8 *For the CFG given below remove the ε production*

$S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow \varepsilon$

Solution : According to the replacement procedure, we will place ε at S in the sentential form.

$$S \rightarrow a\ S\ a \quad \text{if } S = \varepsilon$$

$$S \rightarrow a\ a$$

Similarly if $S \rightarrow b\ S\ b$ if $S = \varepsilon$

$$S \rightarrow b\ b$$

Thus finally the rules are

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

Example 3.8.9 *Eliminate the ε productions from the CFG given below.*

$A \rightarrow 0\ B\ 1\ |\ 1\ B\ 1$

$B \rightarrow 0\ B|1\ B|\varepsilon$

Solution : Now the ε production is $B \rightarrow \varepsilon$ we will delete this production. And then add the productions having B replaced by ε.

$A \rightarrow 0\ B\ 1$

if $B = \varepsilon$

$A \rightarrow 0\ 1$

Similarly $A \rightarrow 1\ B\ 1 \rightarrow 11$

$\therefore$ $A \rightarrow 0\ B\ 1 | 1\ B\ 1 | 01 | 11$

Similarly, $B \rightarrow 0\ B$

if $B = \varepsilon$

$B \rightarrow 0$

as well as $B \rightarrow 1$

$B \rightarrow 0\ B\ |\ 1\ B\ | 0 | 1$

Collectively, we can write

$A \rightarrow 0\ B\ 1 | 1\ B\ 1 | 01 | 11$

$B \rightarrow 0\ B | 1\ B | 0 | 1$

Example 3.8.10 *Construct CFG without ε production from the one which is given below.*

$S \rightarrow a | Ab | aBa$

$A \rightarrow b | \varepsilon$

$B \rightarrow b | A$

Solution : If you observe carefully, then not only A have ε production but even B also indicates ε production i.e. $A \rightarrow \varepsilon$ straight forward but $B \rightarrow A \rightarrow \varepsilon$. Let us apply the method of replacement.

$S \rightarrow A\ b$

if $A = \varepsilon$ then

$S \rightarrow b$

if $B = \varepsilon$

$S \rightarrow a\ a$

$S \rightarrow a | A\ b | b | a\ a | a\ B\ a$

$A \rightarrow b$

$B \rightarrow b$

Finally the rules are

$S \rightarrow a | A\ b | b | a\ a | a\ B\ a$

$A \rightarrow b$

$B \rightarrow b$

Example 3.8.11 *Consider the CFG given below.*

$S \rightarrow Y$

$X \rightarrow Zb$

$Y \rightarrow BW$

$Z \rightarrow AB$

$W \rightarrow Z$

$A \rightarrow aA \mid bA \mid \varepsilon$

$B \rightarrow Ba \mid Bb \mid \varepsilon$

Solution : In this example only A and B shows ε productions.

If we put A = ε in A production then

$$A \rightarrow a \mid b$$

$$B \rightarrow a \mid b$$

Also

$$Z \rightarrow A\ B \qquad \text{Putting } A = \varepsilon,\ B = \varepsilon$$

$$Z \rightarrow \varepsilon\varepsilon \rightarrow \varepsilon$$

So wherever Z appears we place ε

$$X \rightarrow b$$

Since $W \rightarrow Z \rightarrow \varepsilon$ we can straightaway delete it.

$\therefore \qquad Y \rightarrow b$

Collectively let us rewrite the rules

$$S \rightarrow XY$$

$$X \rightarrow b$$

$$Y \rightarrow b$$

$$A \rightarrow a\ A \mid b\ A \mid a \mid b$$

$$B \rightarrow B\ a \mid B\ b \mid a \mid b$$

Since S is a start symbol and it defines XY only whereas X and Y yields b as a terminal symbol. There is no chance for producing rules by A or B. Hence A and B are useless symbols and we can remove it. Finally rules are

$$S \rightarrow XY$$

$$X \rightarrow b$$

$$Y \rightarrow b$$

which ultimately

$$S \rightarrow bb$$

Example 3.8.12 *Consider the CFG given below,*

For eliminating ε productions

$S \rightarrow P\ 0\ Q \mid Q\ Q \mid 0\ R \mid R\ Q\ P$

$P \rightarrow R\ 0 \mid 1\ R \mid R\ R \mid R\ Q\ P$

$Q \rightarrow Q\ 0 \mid P\ Q \mid \varepsilon$

$R \rightarrow 0\ P \mid Q\ Q\ Q$

Solution : As we can see that $Q \rightarrow \varepsilon$.

Similarly $R \rightarrow QQQ \rightarrow \varepsilon$.

Hence we will replace Q and R by ε and accordingly add the production rules.

Similarly $P \rightarrow R\ R \rightarrow \varepsilon$

Let us modify the rules if P or Q or R becomes ε.

$$S \rightarrow P\ 0\ Q \mid 0 \mid 0\ R \mid Q\ Q \mid R\ Q\ P$$

$$P \rightarrow R\ 0 \mid 0 \mid 1 \mid 1\ R \mid RR \mid RQP$$

$$Q \rightarrow Q\ 0 \mid 0 \mid PQ$$

$$R \rightarrow 0\ P \mid 0 \mid Q\ Q\ Q$$

Note that with P, Q, R other meanings are associated, they are not purely giving ε to us. Therefore we cannot remove P, Q and R symbols.

Example 3.8.13 *Eliminate ∈ - productions from the grammar G*

$A \rightarrow aBb \mid bBa$

$B \rightarrow aB \mid bB \mid \in$ **SPPU : Aug.-17, In Sem, Marks 3**

Solution : $B \rightarrow \varepsilon$ is used at RHS of each production rule. On eliminating ε production we get

$$A \rightarrow aBb \mid bBa \mid ab \mid ba$$

$$B \rightarrow aB \mid bB \mid a \mid b$$

3.8.3 Removing Unit Productions

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

For example if

$$X \rightarrow Y$$

$$Y \rightarrow Z \qquad Z \rightarrow X$$

Then X, Y and Z are unit productions. To optimize the grammar we need to remove the unit productions.

If $A \overset{*}{\Rightarrow} B$ is a unit production and $B \rightarrow X_1 X_2 X_3 \ldots X_n$ then while removing $A \rightarrow B$ production we should add a rule

$A \rightarrow X_1 X_2 X_3 \ldots X_n$.

Let us solve something.

Example 3.8.14 *If the CFG is as below.*

$S \rightarrow 0A \mid 1B \mid C$

$A \rightarrow 0S \mid 00$

$B \rightarrow 1 \mid A$

$C \rightarrow 01$

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.

$$S \rightarrow 0A \mid 1B \mid 01$$

Similarly $B \rightarrow A$ is also a unit production so we can modify it as

$$B \rightarrow 1 \mid 0S \mid 00$$

Thus finally we can write CFG without unit production as

$$S \rightarrow 0A \mid 1B \mid 01$$

$$A \rightarrow 0S \mid 00$$

$$B \rightarrow 1 \mid 0S \mid 00$$

$$C \rightarrow 01$$

Example 3.8.15 *Optimize the CFG given below by reducing the grammar.*

S is a start symbol.

$S \rightarrow A \mid 0C1$

$A \rightarrow B \mid 01 \mid 10$

$C \rightarrow \varepsilon \mid CD$

Solution : $S \rightarrow A \rightarrow B$ is a unit production

$C \rightarrow \varepsilon$ is a null production.

$C \rightarrow CD$ B and D are useless symbol.

Reducing a grammar we have to avoid all the above conditions.

Let $S \rightarrow A$

i.e. $A \rightarrow B$ is a useless symbol because B is not defined further more.

$$S \rightarrow 01|10$$

i.e. $$S \rightarrow 01|10|0C1$$

But $$C \rightarrow \varepsilon$$

Hence ultimately $S \rightarrow 01|10$

$A \rightarrow B$ but we can remove this production since B is a useless symbol.

Hence $A \rightarrow 01|10$

But the start symbol $S \rightarrow 01|10$

There is no A in the derivation of A so by considering A also as a useless symbol we get final CFG as

$$S \rightarrow 01|10$$

Example 3.8.16 *Eliminate the unit productions from following grammar*

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow C|b$

$C \rightarrow D$

$D \rightarrow E|bC$

$E \rightarrow d|Ab$

Solution : As,

$S \rightarrow AB$ and

$A \rightarrow a$

There is only one rule with A and that too giving terminal symbol. Hence there is no question of getting unit production with A. Now consider

$B \rightarrow C$

$C \rightarrow D$

$D \rightarrow E$

It is clear that B, C and D are unit productions. As $E \rightarrow d|Ab$, we will replace value of D. Then,

$$D \rightarrow d \mid Ab \mid bC$$

Similarly as $C \rightarrow D$ we can write

$$C \rightarrow d|Ab|bC$$

Hence B becomes

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

Thus the grammar after removing unit productions

$$S \rightarrow AB$$

$$A \rightarrow a$$

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

$$C \rightarrow d \mid Ab \mid bC$$

$$D \rightarrow d \mid Ab \mid bC$$

$$E \rightarrow d \mid Ab.$$

Now there is no path for D and E. From start state we can remove them by considering useless symbols. The optimized grammar will be

$$S \rightarrow AB$$

$$A \rightarrow a$$

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

$$C \rightarrow d \mid Ab \mid bC$$

Example 3.8.17 *Simplify the following grammar :*

i) $S \rightarrow Ab,\ A \rightarrow a,\ B \rightarrow C \mid b,\ \ C \rightarrow D,\ D \rightarrow E,\ E \rightarrow a$

ii) $S \rightarrow 0A0 \mid 1B1 \mid BB,\ A \rightarrow C,\ B \rightarrow S \mid A,\ C \rightarrow S \mid \varepsilon$

SPPU : Dec.-16, End Sem, Marks 8, Oct.-19, In Sem, Marks 4

Solution :

i) Let,

$$S \rightarrow Ab$$

$$A \rightarrow a$$

$$B \rightarrow C \mid b$$

$$C \rightarrow D$$

$$D \rightarrow E$$

$$E \rightarrow a$$

Step 1 : Here $B \rightarrow C$, $C \rightarrow D$, $D \rightarrow E$ and $E \rightarrow a$, are unit productions. These can be eliminated and we get simply $B \rightarrow a \mid b$.

Step 2 : Now if we observe the start state S, we can clearly say that non terminal B is non-reachable. Hence eliminate it as useless symbol.

Step 3 : Thus we get the simplified grammar, on removing B, C, D and E as

$$S \rightarrow Ab$$

$$A \rightarrow a$$

ii) Let,

$$S \rightarrow 0A0$$

$$S \rightarrow 1B1$$

$$S \rightarrow BB$$

$$A \rightarrow C$$

$$B \rightarrow S \mid A$$

$$C \rightarrow S \mid \varepsilon$$

Step 1 : The $A \rightarrow C$, $C \rightarrow S \mid \varepsilon$, are unit productions. These can be eliminated $A \rightarrow S \mid \varepsilon$.

Step 2 :

$B \rightarrow A$, $A \rightarrow S \mid \varepsilon$ are unit productions, these can be eliminated by $B \rightarrow \varepsilon$

Step 3 : As we have

$$S \rightarrow 0A0 \mid 1B1$$

Also $A \rightarrow S$ and $B \rightarrow S$

$\therefore$ We can reduce it as

$$S \rightarrow 0S0 \mid 1S1$$

Step 4 : Similarly $S \rightarrow BB$ can be reduced.

Step 5 : Finally we get

$$S \rightarrow 0S0 \mid 1S1 \mid \varepsilon$$

as simplified productions.

Example 3.8.18 *Consider the grammar G = {(A,B), (a,b),P,A} where P consists of*

$A \rightarrow B$

$B \rightarrow a \mid b$

Eliminate unit productions. **SPPU : Oct.-18, In Sem, Marks 3**

Solution : $A \rightarrow B$ is a unit production. We eliminate it and rewrite the production rule as - $A \rightarrow a \mid b$.

Example 3.8.19 *Optimize the CFG given below by reducing the grammar where S is a start symbol.*

$S \rightarrow A \mid 0C1$

$A \rightarrow B \mid 01 \mid 10$

$C \rightarrow \varepsilon \mid CD$ **SPPU : Oct.-19, In Sem, Marks 4**

Solution :

1) In given production rules, B and D as useless symbol,
2) $S \rightarrow A \rightarrow B$ is unit production
3) $C \rightarrow \varepsilon$ is null production
4) To optimize given production, we must eliminate useless symbol, unit and null production. Hence optimized production is $S \rightarrow 01 \mid 10$

Example 3.8.20 *Simplify the following Context Free Grammar (CFG) :*

$S \rightarrow ASB \mid \varepsilon$

$A \rightarrow aAS \mid \varepsilon$

$B \rightarrow SbS \mid A \mid bb$

G = {(S,A,B), (a,b), S,P : Productions are given above. **SPPU : Dec.-19, End Sem, Marks 6**

Solution : Let,

$S \rightarrow ASB \mid \varepsilon$

$A \rightarrow aAS \mid \varepsilon$

$B \rightarrow SbS \mid A \mid bb$

We will eliminate unit productions

$S \rightarrow A$

Hence, $S \rightarrow ASB \mid \varepsilon$

$A \rightarrow aAS \mid \varepsilon$

$B \rightarrow SbS \mid aAS \mid bb \mid \varepsilon$

Now we will eliminate ε productions

Hence, $S \rightarrow ASB \mid AB \mid AS \mid SB$

$A \rightarrow aAS \mid aA \mid a$

$B \rightarrow SbS \mid aAS \mid bb \mid Sb \mid bS \mid b \mid aA \mid aS \mid bb$

3.9 Normal Forms

SPPU : Dec.-05, 10, 13, 14, 15, 16, 17, 19, May-08, 10, 13, 14, 18, Oct.-18, Marks 12

There are two important normal forms - **Chomsky's Normal Form** and **Greibach Normal Form**

Let us study these normal forms with the help of examples -

3.9.1 Chomsky Normal Form

The Chomsky's Normal Form can be defined as

Non terminal → Non terminal · Non terminal
Non terminal → 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.

Problems on CNF

Example 3.9.1 *Convert the following CFG into CNF*

$S \rightarrow aaaaS$

$S \rightarrow aaaa$

Solution : As we know the rule for Chomsky's normal form is

Non terminal → Non terminal · Non terminal

Non terminal → Terminal

But the CFG given is

$S \rightarrow aaaaS$ rule 1

$S \rightarrow aaaa$ rule 2

If we add a rule

Then out 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 CFG).

$$\boxed{\begin{array}{c} S \rightarrow A\ P_1 \\ P_1 \rightarrow A\ P_2 \\ P_2 \rightarrow A\ P_3 \\ P_3 \rightarrow A\ S \end{array}}$$

Now consider rule 2

$S \rightarrow AAAA$

If we break the rule 2 as

$$S \rightarrow \underset{\substack{\downarrow \\ P_4}}{\boxed{AA}}\ \underset{\substack{\downarrow \\ P_5}}{\boxed{AA}} \quad \text{indicated by}$$

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 3.9.2 *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

$$S \rightarrow B\ A_2 \quad \text{both are in CNF}$$
$$A_2 \rightarrow SB$$

Hence we can write the rule in CNF as

$$S \rightarrow A\ A_1$$
$$A_1 \rightarrow SA$$
$$A \rightarrow a$$
$$S \rightarrow B\ A_2$$

$$A_2 \rightarrow SB$$
$$B \rightarrow b$$
$$S \rightarrow a$$
$$S \rightarrow b$$

Example 3.9.3 *Consider* $a = (\{S, A\}, \{a, b\}, P, S)$

Where P consists of

$S \rightarrow aAS \mid a$

$A \rightarrow SbA \mid SS \mid ba$

Convert it to its equivalent CNF.

Solution : Let us consider,

$$S \rightarrow aAS$$

If we put $R_1 \rightarrow a$

then $S \rightarrow R_1AS$

and if $R_2 \rightarrow AS$

then S becomes

$S \rightarrow R_1 R_2$ is now in CNF

$S \rightarrow a$ is already in CNF

$A \rightarrow SbA$

Let $R_3 \rightarrow b$

Hence $A \rightarrow S R_3A$

Add $R_4 \rightarrow R_3A$

then $A \rightarrow S R_4$

$A \rightarrow SS$ is already in CNF

Now as we have defined $R_1 \rightarrow a$ and $R_3 \rightarrow b$.

$A \rightarrow R_1 R_3$ will be a conversion in CNF of $A \rightarrow ab$.

Finally, we can write

$S \rightarrow R_1 R_2$	$A \rightarrow S R_4$
$S \rightarrow a$	$R_4 \rightarrow R_3A$
$R_1 \rightarrow a$	$R_3 \rightarrow b$
$R_2 \rightarrow AS$	$A \rightarrow SS$
	$A \rightarrow R_1 R_3$

Example 3.9.4 *Convert the given CFG to CNF.*

Consider $G = (V, T, P, S)$

Where $V = \{S, A, B\}$

$T = \{a, b\}$

P consists of

$S \to aB \qquad A \to bAA$

$S \to bA \qquad B \to b$

$A \to a \qquad B \to bS$

$A \to aS \qquad B \to aBB$

SPPU : Dec.-05, Marks 12; Dec.-14, In Sem, Marks 6; Dec.-16, End Sem, Marks 8

Solution : Let us start with first rule.

$R_1 \to a$

$S \to R_1B$ is in CNF for $S \to aB$

$R_2 \to b$

$S \to R_2A$ is in CNF for $S \to bA$

$A \to a$

$A \to a\,S$ can be written as

$A \to R_1S$

Now $A \to bAA$ can be written as

$A \to R_2\,\boxed{AA}$ replace it by R_3

i.e. $R_3 \to AA$

then $A \to R_2R_3$

$B \to a$

$B \to bS$ can be written as

$B \to R_2S$ is in CNF

Now, $B \to aBB$

i.e. $B \to R_1BB \qquad \because \; R_1 \to a$

Let $R_4 \to BB$

then $B \to R_1\,R_4$

Finally we can write,

$S \to R_1B$

$$S \rightarrow R_2A$$
$$A \rightarrow R_1S$$
$$A \rightarrow R_2\ R_3$$
$$B \rightarrow b$$
$$B \rightarrow R_2S$$
$$B \rightarrow R_1\ R_4$$
$$R_1 \rightarrow a$$
$$R_2 \rightarrow b$$
$$R_3 \rightarrow AA$$
$$R_4 \rightarrow BB$$

is a Chomsky's normal form.

Example 3.9.5 *Convert the following CFG to CNF*

$S \rightarrow ABA$

$A \rightarrow aA \mid \varepsilon$

$B \rightarrow bB \mid \varepsilon$ **SPPU : May-13, 14, Marks 6**

Solution : In the Chomsky's normal form the ε production is not allowed. So first we will eliminate ε productions.

$$A \rightarrow \varepsilon \quad \text{and} \quad B \rightarrow \varepsilon$$

If put ε instead of A and B

We can get,

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

Similarly, $A \rightarrow aA \mid a$

$$B \rightarrow bB \mid b$$

Now S → A and S → B is unit production getting introduced in the grammar after removal of ε production. So we will remove unit production also,

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

Now let us convert this grammar to Chomsky's normal form.

Let

$$S \rightarrow ABA$$

$$\boxed{\mathbf{S \rightarrow ABA}}$$

$$\boxed{\mathbf{R_1} \rightarrow \mathrm{BA}}$$

then $S \rightarrow AR_1$

Now

$\mathbf{S \rightarrow AB}$	$\mathbf{S \rightarrow BA}$	$\mathbf{S \rightarrow AA}$

already in CNF

Let $S \rightarrow aA$

We will define $R_2 \rightarrow a$

$\therefore$

$$\boxed{\begin{array}{l} \mathbf{S \rightarrow R_2 A} \\ \mathbf{R_2} \rightarrow a \end{array}}$$

$$\boxed{\mathbf{S \rightarrow a}}$$ already in CNF

$S \rightarrow bB$

Let

$$\boxed{\begin{array}{l} \mathbf{R_3 \rightarrow b} \\ \mathbf{S \rightarrow R_3 B} \end{array}}$$ then

$S \rightarrow b$ already in CNF

Now consider the rules,

$$\boxed{\mathbf{A \rightarrow aA}}$$

We can rewrite it as

$$\boxed{\begin{array}{l} \mathbf{A \rightarrow R_2 A} \\ \mathbf{A \rightarrow a} \end{array}}$$

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 3.9.6 *Convert the following grammar to Chomsky's normal form*
$S \rightarrow a \mid b \mid a\ S\ S.$

Solution : The Chomsky's normal form is

$NT \rightarrow NT \cdot NT$

$NT \rightarrow$ Terminal

We will convert the given grammar in this form

$S \rightarrow a$ already in CNF

$S \rightarrow b$ already in CNF

$S \rightarrow a\ S\ S$

Now we will convert

$S \rightarrow a\ S\ S$ into equivalent CNF

$S \rightarrow R_1\ R_2$

$R_1 \rightarrow a$

$R_2 \rightarrow S\ S$

$\therefore$ The equivalent CNF of given grammar is -

$S \rightarrow a$

$S \rightarrow b$

$S \rightarrow R_1\ R_2$

$R_1 \rightarrow a$

$R_2 \rightarrow S\ S.$

Example 3.9.7 *Convert the following grammar to Chomsky's normal form*
$S \rightarrow \sim S \mid [S \supset S] \mid p \mid q.$ **SPPU : Dec.-13, Marks 6**

Solution : Chomsky's normal form is -

Non terminal $\rightarrow$ Non terminal $\cdot$ Non terminal

Non terminal $\rightarrow$ Terminal

Consider given grammar rule by rule and let us convert it into CNF

$S \rightarrow \sim S$

Can be written as

$$S \rightarrow AS$$

$$A \rightarrow \sim$$

Similarly $S \rightarrow [S \supset S]$

Can be written as

$$S \rightarrow BC$$

$$B \rightarrow DE$$

$$E \rightarrow SF$$

$$D \rightarrow [$$

$$F \rightarrow \supset$$

$$C \rightarrow SG$$

$$G \rightarrow]$$

And, $S \rightarrow p$

$$S \rightarrow q$$

is already in CNF.

Hence the complete grammar written in CNF is -

$$S \rightarrow AS$$

$$A \rightarrow \sim$$

$$S \rightarrow BC$$

$$B \rightarrow DE$$

$$E \rightarrow SF$$

$$D \rightarrow [$$

$$F \rightarrow \supset$$

$$C \rightarrow SG$$

$$G \rightarrow]$$

$$S \rightarrow p$$

$$S \rightarrow q$$

Example 3.9.8 *Convert the following grammar to CNF*

$S \rightarrow AACD$

$A \rightarrow aAb \mid \varepsilon$

$C \rightarrow aC \mid a$

$A \rightarrow aDa \mid bDb \mid \varepsilon$ **SPPU : May-10, Dec.-10, Marks 6, Oct.-18, In Sem, Marks 4**

Solution : Before converting any grammar to its normal form we must first simplify it by removing ε production and by eliminating useless symbols.

Let S → A A A D

But there is no production rule defined for given non-terminal D. Hence S and D are useless symbols. But S is a start symbol and if we eliminate it then the grammar becomes **invalid**. Hence **conversion** of such grammar to CNF is **not possible**.

Example 3.9.9 *Reduce the following grammar to Chomsky Normal Form :*

$S \rightarrow 1A \mid 0B$

$A \rightarrow 1AA \mid 0S \mid 0$

$B \rightarrow 0BB \mid 1S \mid 1$ **SPPU : Dec.-13, Marks 6**

Solution : Refer example 3.9.4 by assuming a = 0 and b = 1.

Example 3.9.10 *Convert the following CFG into Chomsky Normal Form (CNF) :*

$S \text{ - > } AB$

$A \text{ - > } CA \mid \wedge$

$B \text{ - > } DB \mid \wedge$

$C \text{ - > } 011 \mid 1$

$D \text{ - > } 01$ **SPPU : Dec.-15, End Sem, Marks 6**

Solution : Before converting the given grammar to CNF we will eliminate ∧ production.

$A \rightarrow C\,A \mid \wedge$ becomes

$A \rightarrow CA \mid 011 \mid 1$

Similarly

$B \rightarrow DB \mid \wedge$ becomes

$B \rightarrow DB \mid 01$

The CNF conversion is as follows :

Production Rule	CNF
$S \rightarrow AB$	already is in CNF
$A \rightarrow CA$	is already in CNF

$A \rightarrow 011$	$A \rightarrow XY$ $X \rightarrow 0$ $Y \rightarrow ZZ$ $Z \rightarrow 1$
$B \rightarrow DB$	is already in CNF
$B \rightarrow XZ$	

3.9.2 Greibach Normal Form

Now we will discuss one more interesting normal form called **Greibach Normal Form**.

The rule for GNF is

Non - terminal → One terminal. Any number of non-terminals

In short,

$$\boxed{NT \rightarrow t.\ NT}$$

For example :

$$\left.\begin{array}{l} S \rightarrow aA \\ S \rightarrow a \end{array}\right\} \text{ is in GNF}$$

But $S \rightarrow AA$

Or $S \rightarrow Aa$

$$\left.\begin{array}{l} S \rightarrow AA \\ S \rightarrow Aa \end{array}\right\} \text{ is not in GNF}$$

To convert given CFG into GNF very interesting procedure is followed.

Step 1 : Convert the given grammar into simplified form by eliminationg ε productions, unit productions and useless symbols.

Step 2 : Bring the grammar to Chomsky's Normal Form (CNF). This step can be optional.

Step 3 : Number the non-terminal symbols in ascending order i.e. A_1, A_2, A_3 and so on.

Step 4 : In the rule of the form

$$A_i \rightarrow A_j A_k \ldots$$

There must be $i < j$. If $i > j$ or $i = j$ then process the rule to convert it to GNF.

When $i = j$ i.e.

$$A_i \rightarrow A_j A_k \ldots$$

then that grammar is said to be **left recursive**. Remove the left recursion using following formula

If $A \rightarrow A\alpha | \beta$

then convert above grammar in the following form

$$A \rightarrow \beta A' | \beta$$

$$A' \rightarrow \alpha A' | \alpha$$

Step 5 : Then convert each rule in the form

Non-terminal $\rightarrow$ one terminal $\cdot$ Any number of non-terminals

Let us solve some problems based on conversion to GNF.

Example 3.9.11 *Convert the following grammar to Greibach Normal Form*

S → abSb

S → aa

Solution : The Greibach Normal Form is

NT $\rightarrow$ t $\cdot$(set of NT)

Step 1 : There is no ε, or unit production. So there is n need to simplify the grammar.

Step 2 : Convert the grammar to CNF.

S → abSb	S → XY X → AB Y → SB A → a B → b
S → aa	S → AA

Step 3 : Now rename non-terminals in ascending order.

S $\rightarrow$ XY can be converted to $\mathbf{A_1 \rightarrow A_2 A_3}$

$$X \rightarrow AB \Rightarrow \mathbf{A_2 \rightarrow A_4 A_5}$$

$$Y \rightarrow SB \Rightarrow \mathbf{A_3 \rightarrow A_1 A_5}$$

$$A \rightarrow a \Rightarrow \mathbf{A_4 \rightarrow a}$$

$$B \rightarrow b \Rightarrow \mathbf{A_5 \rightarrow b}$$

Step 4 : i) Now we will verity the condition of A_i, A_j as if $A_i < A_j$?

The rules are

$A_1 \rightarrow A_2A_3$

$A_2 \rightarrow A_4A_5$

$\boxed{A_3 \rightarrow A_1A_5} \longrightarrow \underset{\underset{A_i}{\uparrow}}{A_3} \rightarrow \underset{\underset{A_j}{\uparrow}}{A_1} A_5 \longrightarrow$ Here i > j i.e. 3 > 1

$A_4 \rightarrow a$

$A_5 \rightarrow b$

Therefore we will replace A_1 on R.H.S

$$A_3 \rightarrow \boxed{A_1} A_5$$
$$\downarrow$$
$$A_3 \rightarrow \boxed{A_2} A_3A_5 \qquad \because A_1 \rightarrow A_2A_3$$
$$\downarrow$$
$$A_3 \rightarrow A_4A_5A_3A_5$$

Now we get i < j to be satisfied for $A_i \rightarrow A_jA_k \ldots$

To summerize,

1) $A_1 \rightarrow A_2A_3$
2) $A_2 \rightarrow A_4A_5$
3) $A_3 \rightarrow A_4A_5A_3A_5$
4) $A_4 \rightarrow a$
5) $A_5 \rightarrow b$

ii) Now consider rules for non terminals A_2 and A_3. The A_4 on RHS is replaced by terminal symbol **a**. The rules then become

1) $A_1 \rightarrow A_2A_3$
2) $A_2 \rightarrow aA_5$
3) $A_3 \rightarrow aA_5A_3A_5$
4) $A_4 \rightarrow a$
5) $A_5 \rightarrow b$

iii) If we observe all the above rules we find that all the rules are in GNF except rule (1). So we will consider it.

$A_1 \rightarrow A_2A_3$

Replace A_2 by its RHS.

$\therefore \quad A_1 \rightarrow aA_5A_3$

To summerise all, we get following rules that are all in GNF.

$A_1 \rightarrow aA_5A_3$

$A_2 \rightarrow aA_5$

$A_3 \rightarrow aA_5A_3A_5$

$A_4 \rightarrow a$

$A_5 \rightarrow b$

Example 3.9.12 *Convert the given CNF to CFG*

$S \rightarrow ABA$

$A \rightarrow aA|\varepsilon$

$B \rightarrow bB|\varepsilon$

Solution :

Step 1 : • There is ε - Production, so we will simplify the given grammar as follows

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

$A \rightarrow aA|a$

$B \rightarrow bB|b$

- Now eliminate unit productions

$S \rightarrow A|B$

so,

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

$A \rightarrow aA|a$

$B \rightarrow bB|b$

Step 2 :

- In above grammar the rules for A and B non-terminals are already in GNF.
- In fact, some of the rules of S are also in GNF.
- Hence we will handle,

$S \rightarrow ABA|BA|AB|AA$

by simply substituting the values of A and B. Here we need not have to bring grammar in ascending order of $A_i, A_j, A_k \ldots$

Step 3 : We will then get -

$S \rightarrow ABA$	$S \rightarrow aABA\|aBA$
$S \rightarrow BA$	$S \rightarrow bBA\|bA$
$S \rightarrow AB$	$S \rightarrow aAB\|aB$
$S \rightarrow AA$	$S \rightarrow aAA\|aA$

Step 4 : Thus now all the rules are in GNF. To summerize

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

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

$$A \rightarrow aA \mid a$$

$$B \rightarrow bB \mid b$$

Example 3.9.13 *Convert given CFG to GNF where V = {S, A}, T = {0, 1} and P is*

$S \rightarrow AA \mid 0$

$A \rightarrow SS \mid 1$

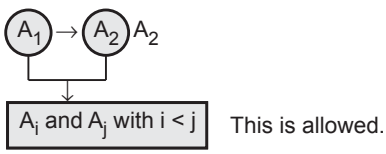
Solution :

Step 1 : The grammar is simplified and it is already in Chomsky's normal form. So we will number out the non-terminal symbols.

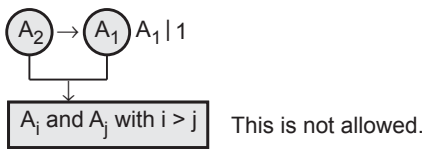
$$S \rightarrow AA \mid 0 \quad \text{becomes } A_1 \rightarrow A_2A_2 \mid 0$$

$$A \rightarrow SS \mid 1 \quad \text{becomes } A_2 \rightarrow A_1A_1 \mid 1$$

Step 2 : Consider



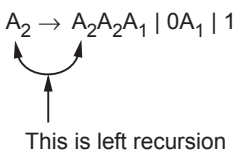
But



Hence we will replace A_1 on RHS by its definition. So we get

$$A_2 \rightarrow A_2A_2A_1 \mid 0A_1 \mid 1$$

Step 3 : Now consider the newly obtained rule



The left recursion can be removed using the following formula

$A \rightarrow A\alpha \mid \beta$ gives $A \rightarrow \beta A' \mid \beta$

$$A' \rightarrow \alpha A' \mid \alpha$$

Hence,

$$A_2 \rightarrow \underbrace{A_2A_2A_1}_{\alpha} \mid \underbrace{0A_1}_{\beta_1} \mid \underbrace{1}_{\beta_2}$$

We get,

$$A_2 \rightarrow 0A1A' \mid 1A' \mid 0A_1 \mid 1 \quad (1)$$

$$A' \rightarrow A_2A_1A' \mid A_2A_1$$

Thus now if we observe A_2 then it is there in GNF.

Step 4 : Now consider rule for A_1.

$$A_1 \rightarrow A_2A_2 \mid 0 \quad (2)$$

We will use equation (1) for equation (2) and replace A_2 on R.H.S.

$$\therefore \quad A_1 \rightarrow 0A_1A'A_2 \mid 1A'A_2 \mid 0A_1A_2 \mid 1A_2 \mid 0$$

Step 5 : Now both A_1 and A_2 are in GNF. The only remaining non-terminal which is not in GNE is our newly introduced A'. The A' rule is

$$A' \rightarrow A_2A_1A' \mid A_2A_1 \quad (3)$$

Again use equation (1) and replace A_2 of equation (3)

$$\therefore \quad A' \rightarrow 0A_1A'A_1A' \mid 1A'A_1A' \mid 0A_1A_1A' \mid 1A_1A'$$

$$A' \rightarrow 0A1A'A_1 \mid 1A'A_1 \mid 0A_1A_1 \mid 1A_1$$

Step 6 : Thus now all the rules are in GNF. The rules can be summerized as follows

$$A_1 \rightarrow 0A_1A'A_2 \mid 1A'A_2 \mid 0A_1A_2 \mid 1A_2 \mid 0$$

$$A_2 \rightarrow 0A_1A' \mid 1A' \mid 0A_1 \mid 1$$

$$A' \rightarrow 0A_1A'A_1A' \mid 1A'A_1A' \mid 0A_1A_1A' \mid 1A_1A'$$

$$A' \rightarrow 0A_1A'A_1 \mid 1A'A_1 \mid 0A_1A_1 \mid 1A_1$$

Example 3.9.14 *Convert the grammar $S \rightarrow AB, A \rightarrow BS \mid b, B \rightarrow SA \mid a$ into Greibach normal form.*

Solution :

Step 1 : The grammar is simplified and is already in CNF. So we will number out the non terminals as $A_1, A_2, A_3, \ldots$

$$S = A_1, \quad A = A_2, \quad B = A_3$$

$$A_1 \rightarrow A_2A_3$$

$$A_2 \rightarrow A_3A_1 \mid b$$

$$A_3 \rightarrow A_1A_2 \mid a$$

Step 2 : For the rules

$$A_1 \rightarrow A_2A_3$$

$$A_2 \rightarrow A_3A_1|b$$

with $A_i \rightarrow A_jA_k \ldots,$ the $i < j$

But for

$$A_3 \rightarrow A_1A_2|a \qquad i > j \text{ i.e. } 3 > 1$$

Hence we will process A_3 rule first

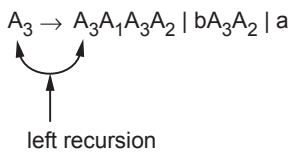
$$A_3 \rightarrow A_1A_2|a$$

$$A_3 \rightarrow A_2A_3A_2|a \qquad \because A_1 \rightarrow A_2A_3$$

$$A_3 \rightarrow A_3A_1A_3A_2|bA_3A_2|a \qquad \because A_2 \rightarrow A_3A_1|b$$

Step 3 : Consider newly obtained rule

The left recursion can be eliminated as



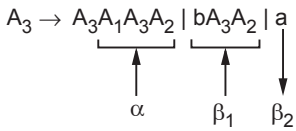
$$A \rightarrow A\alpha|\beta \quad \text{then} \quad A \rightarrow \beta A'|\beta$$

$$A' \rightarrow \alpha A'|\alpha$$

Hence

$$\therefore \qquad A_3 \rightarrow bA_3A_2A'|aA' \tag{1}$$

$$A' \rightarrow A_1A_3A_2A'|A_1A_3A_2$$



Now rule for A_3 is in GNF

Step 4 : Now consider rule for A_2

$$A_2 \rightarrow A_3A_1|b \tag{2}$$

Substitute equation 1 in equation 2

$$A_2 \rightarrow bA_3A_2A'A_1|aA'A_1|b \tag{3}$$

Note that now A_2 is also in GNF

Step 5 : Now we will consider rule for A_1

$$A_1 \rightarrow A_2A_3 \tag{4}$$

Substitute equation (3) in equation (4),

$$A_1 \rightarrow bA_3A_2A'A_1A_3 | aA'A_1A_3 | bA_3 \tag{5}$$

Now A_1 is also in GNF.

Step 6 : Now the only remaining non terminal in A'

$$A' \rightarrow A_1A_3A_2A' | A_1A_3A_2 \tag{6}$$

Substitute equation (5) in equation (6)

We will get

$$A' \rightarrow bA_3A_2A'A_1A_3A_3A_2A' | aA'A_1A_3A_1A_3A_2A' | bA_3A_3A_2A' | bA_3A_2A'A_1A_3A_3A_2 | aA'A_1A_3A_3A_2 | bA_3A_3A_2$$

Thus A' is now in GNF.

Step 7 : Thus all the rules are in GNF. The rules can be summerized as

$$A_1 \rightarrow bA_3A_2A'A_1A_3 | aA'A_1A_3 | bA_3$$

$$A_2 \rightarrow bA_3A_2A'A_1 | aA'A_1 | b$$

$$A_3 \rightarrow bA_3A_2A' | aA'$$

$$A' \rightarrow bA_3A_2A'A_1A_3A_3A_2A' | aA'A_1A_3A_1A_3A_2A' | bA_3A_3A_2A' | bA_3A_2A'A_1A_3A_3A_2 | aA'A_1A_3A_3A_2 | bA_3A_3A_2$$

Review Question

1. *What is a normal form ? Explain Chomsky Normal Form (CNF) and Greibach Normal Form (GNF) with suitable example.*

SPPU : May-08, Marks 8, Dec.-17, 19, End Sem, May-18, End Sem, Marks 8

3.10 Pumping Lemma for CFG

- 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 $|w| = |pqrst| = 1$.

Fig. 3.10.1

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.

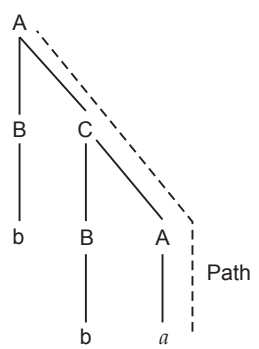


Fig. 3.10.2

Let us consider a grammar,

$$G = (\{A, B, C\}, \{a\}, \{A \rightarrow BC$$
$$B \rightarrow BA$$
$$C \rightarrow BA$$

$$A \rightarrow a$$

$$B \rightarrow b\}, A)$$

Thus

$$A \overset{*}{\Rightarrow} bba = w$$

i.e. path length i = 3

$|w| \leq 2^{i-1}$ 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 3.10.1 *Using the CFL pumping lemma, show that the following language is not context free.* $L = \{a^i b^j c^k | i < j < k\}$.

Solution : Let us prove this statement by method of contradiction. Assume L is a context free language. And w be any string such that $w \in L$.

Let, w be

$$w = p\ q\ r\ s\ t$$

Let, $|qs| \geq 1$ and $|p\ r\ s| \leq n$ and the p r s can contain at the most strings of *a*'s and c's. The string $p\,q^i\,r\,s^i\,t$ contains additional occurrences of q's and s's.

As per the condition mentioned in a lemma, $|qs| \geq 1$ this means that either q or s contains atleast one a then qs can not contain any c's and as per the language L the number of c's has to be more than number of *a*'s. Therefore the $w \notin L$. Similarly if $w = p\,q^0 r\,s^0\,t$ then also there are fewer occurrences of c's than *a*'s which can not be our L. Hence the assumption which we had made that L is a context free language is wrong.

Example 3.10.2 *Show that* $L = \{a^n b^n c^n | n \geq 0\}$ *is not a context free language.*

Solution : Let us assume that

$$L = a^n\ b^n\ c^n \text{ is a context free language.}$$

Let, w be any string such that $w \in L$.

Let,

$$w = pqrst$$

Let, $|qs| \geq 1$ $|prs| \leq n$

Now let us consider

$$w = pq^i rs^i t$$

i.e. we have additional occurrences of q and s. Consider various cases.

Case 1 : Consider i = 0 then,

$$w = pq^i rs^i t$$

if i = 0 then q^0 and s^0. That means q and s are absent then,

$$w = prt$$

```
w = aa bb cc
    ⊔ ⊔⊔ ⊔⊔
    p q r s t
```

$$= a^2bc$$

i.e $= aabc \notin L$

Hence our assumption of L being CFG is wrong.

Case 2 : Consider i = 2 then,

$$w = pq^i rs^i t$$

$$= pqqrsst$$

Consider,

```
w = aa bb cc
    ⊔ ⊔⊔ ⊔⊔
    p q r s t
```

Now with i = 2 we get,

$$= a^2 b^3 c^3 \notin L$$

```
w = aa bbb ccc
    ⊔ ⊔⊔⊔ ⊔⊔⊔
    p qq r ss t
```

Thus our assumption of L being CFG is wrong. This proves that the given language L is not a context free.

3.11 Closure Properties of CFL

SPPU : May 09, Dec.-18, Marks 6

The regular languages are closed under union, concatenation and kleen closure. Now we will discuss closure properties of context free languages. In the sense, we will, check which are those properties for them CFLs are closed under. The context free languages are closed under some operation means after performing that particular operation on those CFLs the resultant language is context free language. These properties are as below.

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

$P_1 = \{$

$S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S B_1 \mid \varepsilon$

$A_1 \rightarrow a$

$B_1 \rightarrow b$

$\}$

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

$P = \{ \quad S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow A_1\ S_1\ A_1 \mid B_1\ S\ B_1 \mid \varepsilon$

$A_1 \rightarrow a$

$B_1 \rightarrow b$

$S_2 \rightarrow aA_2A_2 \mid bB_2B_2$

$A_2 \rightarrow b$

$B_2 \rightarrow a$

}

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_1L_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,

$P_1 = \{ \quad S_1 \rightarrow A_1\ S_1\ A_1 \mid B_1\ S_1\ B_1 \mid \varepsilon$

$A_1 \rightarrow a$

$B_1 \rightarrow b$

}

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,

$P_2 = \{ S_2 \rightarrow aA_2A_2 \mid bB_2B_2$

$A_2 \rightarrow b$

$B_2 \rightarrow a$

}

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,

$P = \{ \; S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow A_1 \, S_1 \, A_1 \mid B_1 \, S_1 \, B_1 \mid \varepsilon$

$A_1 \rightarrow a$

$B_1 \rightarrow b$

$S_2 \rightarrow aA_2A_2 \mid bB_2B_2$

$A_2 \rightarrow b$

$B_2 \rightarrow a$

$\}$

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 $G_1 \rightarrow \varepsilon \; L_1$.

The CFG G_1 can be given as

$G_1 = \{V_1, \Sigma, P_1, S_1\}$ where S_1 is a start symbol

$P_1 = \{ \; S_1 \rightarrow A_1S_1A_1 \mid B_1S_1B_1 \mid \varepsilon$

$A_1 \rightarrow a$

$B_1 \rightarrow b$

$\}$

Now $L = L_1^*$ can be represented by a grammar G such that

$G = \{ \; (V, \Sigma, P, S)$

$V = \{S, S_1, A_1, B_1\}$

and $P = S \rightarrow S_1 \, S \mid \varepsilon$

$S_1 \rightarrow A_1 \, S_1 \, A_1 \mid B_1 \, S_1 \, B_1$

$A_1 \rightarrow a$

$B_1 \rightarrow b$

$\}$

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, $L_1 = \{0^n \, 1^n \, 2^i \mid n \geq 1 \, , \, i \geq 1\}$

$$L_2 = \{0^n\ 1^n\ 2^n | n \geq , i \geq 1\}$$

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.

Theorem 6 : Context Free Languages are closed under homomorphism.

Proof : Let, G = (V,T,P,S) be the context free grammar for generating the language L. The input set is $\sum$ and let Γ be the set of substitution symbols. The $\sum^* \rightarrow \Gamma^*$ be a homomorphism.

Let $a \in L$ and w be the substitution that replaces every $a \in \sum$.

Then $h(w) = h(a_1)h(a_2)h(a_3)$ which ultimately generates h(L). Thus $h(a_i)$ is homomorphism for every $a_i \in \sum$.

For example : Consider

$$P = S \rightarrow aSb \mid bSb \mid \varepsilon$$

Let $h(a) = 00$

$h(b) = 11$

Then $h(w) = S \rightarrow 00S11 \mid 11S00 \mid \varepsilon$

is again a GFG.

Hence it is proved that context free languages are closed under homomorphism.

Review Questions

1. *Write a note on closure properties of CFL.* **SPPU : May 09, Marks 4**
2. *Show that CFLs are not closed under intersection and complementation.* **SPPU : Dec.-18, End Sem, Marks 6**
3. *Prove that CFLs are closed under union, concatenation and Kleen's closure.* **SPPU : Dec.-18, End Sem, Marks 6**

3.12 Decision Properties of CFL

SPPU : Dec.-11, Marks 5

Various decision properties of CFLs are

1. Emptiness
2. Finiteness
3. Membership

These properties are discussed in detail as follows.

1. Emptiness

Theorem : There exists an algorithm which can determine whether or not the given context free grammar can generate any word at all.

Proof : The algorithm to determine whether or not a grammar generates any word is given below -

i) Put the dot (•) above every terminal symbol. If there is any null string ε then put dot over ε as well. These symbols occur at right hand side of production rule.

ii) If all the symbols of right hand side of the production rule are dotted then dot the corresponding non terminal (which is at left hand side of production rule) throughout the grammar. Repeat this step as long as possible.

iii) If the stating symbol gets dotted then answer is yes otherwise answer is no.

Let us apply this algorithm on some grammar.

Let,

$$S \rightarrow PQ$$

$$P \rightarrow AP \mid AA$$

$$A \rightarrow a$$

$$Q \rightarrow BQ \mid BB$$

$$B \rightarrow b$$

Now if we want to put dots we will put over terminals at RHS of rule.

Step 1 :

1) $S \rightarrow PQ$

2) $P \rightarrow AP \mid AA$

3) $A \rightarrow \dot{a}$

4) $Q \rightarrow BQ \mid BB$

5) $B \rightarrow \dot{b}$

Step 2 :

As we RHS of rule 3) and RHS of rule 5) is dotted we will dot A and B throughout the grammar.

1) $S \rightarrow PQ$

2) $P \rightarrow \dot{A}P \mid \dot{A}\dot{A}$

3) $\dot{A} \rightarrow \dot{a}$

4) $Q \rightarrow \dot{B}Q \mid \dot{B}\dot{B}$

5) $\dot{B} \rightarrow \dot{b}$

Now, in rule 2) the alternate production at RHS is dotted $\left(\dot{A}\dot{A}\right)$. Hence dot P. Similarly dot Q.

Hence,

1) $S \rightarrow \dot{P}\dot{Q}$

2) $\dot{P} \rightarrow \dot{A}\dot{P} \mid \dot{A}\dot{A}$

3) $\dot{A} \rightarrow \dot{a}$

4) $\dot{Q} \rightarrow \dot{B}Q \mid \dot{B}\dot{B}$

5) $\dot{B} \rightarrow \dot{b}$

Thus in rule 1) the R.H.S. is dotted. Hence dot S. But since S being dotted is a start symbol, the algorithm generates answer yes. Hence we can prove that an algorithm exists which determines whether or not the given grammar generates any string.

Finiteness

Theorem : There exists an algorithm to determine whether the given context free grammar generates a finite or infinite language.

SPPU : Dec.-11, Marks 5

Proof : We will write following algorithm -

1. Make the grammar simplified by removing unit productions and useless symbols.
2. Check whether there is any self-embedded non terminal or not by following steps.
   a) Underline some non terminal say X which is at LHS of production rule.
   b) Then put dot over all the X which are at RHS throughout the grammar. The dotted X and underlined X are treated as two different symbols.
   c) Put dot over any non terminal at LHS whose RHS contains any dotted symbols. Dot this non terminal throughout the grammar.
   d) If underlined X gets dotted then X is called self embedded otherwise it is not self embedded.
3. If the grammar contains any self embedded non terminal then it generates infinite languages otherwise it generates finite languages.

To support this algorithm, let us take one grammar.

$$S \rightarrow PQa \mid bpz \mid b$$

$$P \rightarrow Xb \mid bZa$$

$$Q \rightarrow bPP$$

$$X \rightarrow aZa \mid aaa$$

$$Z \rightarrow ZPbP$$

The above grammar has Z as useless symbol. Hence by removing it we get,

1) $S \rightarrow PQa \mid b$
2) $P \rightarrow Xb$
3) $Q \rightarrow bPP$
4) $X \rightarrow aaa$

Now we will mark X of rule 4) underlined and put dot over x in rule 2)

1) $S \rightarrow PQa \mid b$

2) $P \rightarrow \dot{X}b$

3) $Q \rightarrow bPP$

4) $\underline{X} \rightarrow aaa$

As R.H.S. of rule 2) contains one dotted symbol dot P throughout the grammar

$$S \rightarrow \dot{P}Qa \mid b$$

$$\dot{P} \rightarrow \dot{X}b$$

$$Q \rightarrow b\dot{P}\dot{P}$$

$$\underline{X} \rightarrow aaa$$

Then Q gets dotted

$$S \rightarrow \dot{P}\dot{Q}a \mid b$$

$$\dot{P} \rightarrow \dot{X}b$$

$$\dot{Q} \rightarrow b\dot{P}\dot{P}$$

$$\underline{X} \rightarrow aaa$$

The S gets dotted

$$\dot{S} \rightarrow \dot{P}\dot{Q}a \mid b$$

$$\dot{P} \rightarrow \dot{X}b$$

$$\dot{Q} \rightarrow b\dot{P}\dot{P}$$

$$\underline{X} \rightarrow aaa$$

Now all the non terminals are dotted but X is not. That means X is not self embedded symbol. Hence the grammar generates finite language.

Membership : There exists an algorithm which tells whether given string belongs to given grammar.

To check whether the given grammar generates desired string the derivation tree can be drawn.

Consider the grammar,

$$S \rightarrow PQa \mid b$$

$$P \rightarrow Xb$$

$Q \rightarrow bPP$

$X \rightarrow aaa$

Check whether it generates string aaabbaaabaaaba.

The leaves generated string aaabbaaabaaaba.

3.13 Chomsky Hierarchy

SPPU : May-06, Dec.-11, Marks 8

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	n!

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.

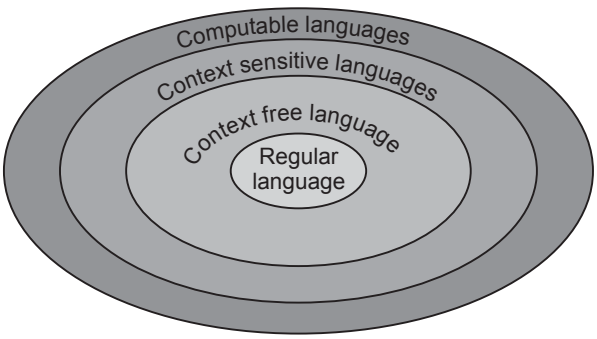


Fig. 3.13.1 Chmosky hierarchy

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

$$A \rightarrow \alpha$$

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 is 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 $A \rightarrow \varepsilon$ 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.

Review Questions

1. *Define Type 0 and Type 1 grammer.* **SPPU : May-06, Marks 4**
2. *Write short note on Chomsky hierarchy.* **SPPU : Dec.-11, Marks 8**

3.14 Cock-Younger-Kasami Algorithm

- Cock-Younger-Kasami or CYK algorithm is used to decide whether a string belongs to language of grammar or not.

- It is also called as **membership algorithm** of context free grammar.
- CYK algorithm works only on CFG which are in Chomsky-Normal-Form (CNF).

Algorithm

```
Begin
for ( i = 1 to n do )
Vi1 { A | A → a is a production where ith symbol of x is a }
for ( j = 2 to n do )
for ( i = 1 to n - j + 1 do )
Begin
Vij = → Φ
For k = 1 to j - 1 do
Vij = Vij → ∪
{A | A → BC is a production where B is in Vik and C is in V(i + k)(j - k) }
End
End
```

Let us understand this algorithm with some suitable example.

Consider the following grammar to check the acceptance of the string **w = baaba**

$$S \rightarrow AB \mid BC$$

$$A \rightarrow BA \mid a$$

$$B \rightarrow CC \mid b$$

$$C \rightarrow AB \mid a$$

The string can be represented in a triangular table as -

The string 'x' has following positions -

x_{15}				
x_{14}	x_{24}			
x_{13}	x_{23}	x_{33}		
x_{12}	x_{22}	x_{32}	x_{42}	
x_{11}	x_{21}	x_{31}	x_{41}	x_{51}

baaba				
baab	aaba			
baa	aab	aba		
ba	aa	ab	ba	
b	a	a	b	a

Notation for Non-terminal V_{ij}

The V_{ij} represents a **set of variables** in the grammar which can drive the substring x_{ij}.

If set of variables contain start symbol then -

1) The substring x_{ij} can be derived from given grammar.
2) The substring x_{ij} is a memeber of the language of the given grammar.

The triangular table for V_{ij} is as given below -

V_{15}				
V_{14}	V_{24}			
V_{13}	V_{23}	V_{33}		
V_{12}	V_{22}	V_{32}	V_{42}	
V_{11}	V_{21}	V_{31}	V_{41}	V_{51}

Step 1 : We will fill up x_{ij} and V_{ij} table as -

- **Finding x_{11} and V_{11} -** The V_{11} represents the set of variables deriving x_{11}. Here x_{11} = b. From given CFG, only non-terminal B derives 'b' in given grammar.

  $\therefore \quad x_{11} = b, \qquad V_{11} = \{B\}$

- **Finding x_{21} and V_{21} -** The V_{21} represents the set of variables deriving x_{21}. Here x_{21} = a. From given CFG, only non-terminals A, C derives 'a' in given grammar.

  $\therefore \quad x_{21} = a, \qquad V_{21} = \{A, C\}$

- **Finding x_{31} and V_{31} -** The V_{31} represents the set of variables deriving x_{31}. Here x_{31} = a. From given CFG, only non-terminals A and C derive 'a'.

  $\therefore \quad x_{31} = a, \qquad V_{31} = \{A, C\}$

- **Finding x_{41} and V_{41} -** The V_{41} represents the set of variables deriving x_{41}. Here x_{41} = b. From given CFG, the non-terminal B derives 'b'.

  $\therefore \quad x_{41} = b, \qquad V_{41} = \{B\}$

- **Finding x_{51} and V_{51} -** The V_{51} represents the set of variables deriving x_{51}. Here x_{51} = a. From given CFG, the non-terminals A and C derive 'a'.

  $\therefore \quad x_{51} = a, \qquad V_{51} = \{A, C\}$

The x_{ij} and V_{ij} table, partially filled up with bottom-most row is -

The x_{ij} table

b	a	a	b	a

The V_{ij} table

{B}	{A, C}	{A, C}	{B}	{A, C}

Step 2 : Now for filling up the table in upward direction from the 2nd row from bottom, we will use following formula -

$$V_{ij} = V_{ik} \; V_{(i+k)\,(j-k)}$$

where k varies from 1 to j – 1.

- **Finding x_{12} and V_{12} -** We have i = 1, j = 2, k = 1

Here x_{12} = 'ba'

$$V_{12} = V_{11} \; V_{(1+1)\,(2-1)}$$

$$= V_{11} \; V_{21}$$

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

$$= \{BA, BC\}$$

$$\therefore \quad V_{12} = \{A, S\}$$

- **Finding x_{22} and V_{22} -** We have i = 2, j = 2, k = 1

Here x_{22} = 'aa'

$$V_{ij} = V_{ik} \; V_{(i+k)\,(j-k)}$$

$$V_{22} = V_{21} \; V_{31}$$

$$V_{22} = \{A, C\} \; \{A, C\}$$

$$= \{AA, AC, CA, CC\}$$

The AA, AC and CA do not exist.

$$\therefore \quad V_{22} = \{CC\}$$

$$\therefore \quad V_{22} = \{B\}$$

- **Finding x_{32} and V_{32} -** We have i = 3, j = 2, k = 1

Here x_{32} = 'ab'

$$V_{ij} = V_{ik} \; V_{(i+k)\,(j-k)}$$

$$V_{32} = V_{31} \; V_{41}$$

$$V_{32} = \{A, C\} \; \{B\}$$

$$= \{AB, CB\}$$

As 'CB' can not be derived from any non-terminal,

$$V_{32} = \{AB\}$$

$$\therefore \quad V_{32} = \{S, C\}$$

- **Finding x_{42} and V_{42} -** We have i = 4, j = 2, k = 1

Here x_{42} = 'ba'

$$V_{ij} = V_{ik} \; V_{(i+k)\,(j-k)}$$

$$V_{42} = V_{41} \; V_{51}$$

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

$$= \{BA, BC\}$$

$$\therefore \quad V_{42} = \{A, S\}$$

The partially filled x_{ij} and V_{ij} tables are

The x_{ij} table

ba	aa	ab	ba	
b	a	a	b	a

The V_{ij} table

{A, S}	{B}	{S, C}	{A, S}	
{B}	{A, C}	{A, C}	{B}	{A, C}

Step 3 :

- **Finding x_{13} and V_{13} -** Here i = 1, j = 3, k = j – 1, (1, 2)

$$V_{ij} = V_{ik} \; V_{(i+k)\,(j-k)}$$

$$V_{13} = V_{11} \; V_{22} \cup V_{12} \; V_{41}$$

$$= \{B\} \; \{B\} \cup \{A, S\} \; \{A, C\}$$

$$= \{BB\} \cup \{AA, SA, AC, SC\}$$

As we can not get BB, AA, SA, AC, SC.

$$\therefore \quad V_{13} = \phi \cup \phi$$

$$V_{13} = 0$$

- **Finding x_{23} and V_{23} -** Here i = 2, j = 3, k = (3 – 1) = (1, 2)

$$x_{23} = aab$$

$$V_{ij} = V_{ik} \; V_{(i+k)\,(j-k)}$$

$$V_{23} = V_{21} \; V_{32} \cup V_{22} \; V_{41}$$

$$= \{A, C\} \; \{S, C\} \cup \{B\} \; \{B\}$$

$$= \{AS, AC, CS, CC\} \cup \{BB\}$$

Out of the above, only CC can be obtained.

$\therefore \quad V_{23} = \{CC\}$

$\therefore \quad V_{23} = \{B\}$

- **Finding x_{33} and V_{33} -**

$x_{33} = aba$

$V_{ij} = V_{ik} \; V_{(i+k)\,(j-k)}$

We have i = 3, j = 3, k = 1 to (3 – 1) = 1, 2

Substituting values in the formula, we get -

$V_{33} = V_{31} \cdot V_{42} \cup V_{32} \cdot V_{51}$

$V_{33} = \{A, C\}\ \{A, S\} \cup \{S, C\}\ \{A, C\}$

$V_{33} = \{AA, AS, CA, CS\} \cup \{SA, SC, CA, CC\}$

Since AA, AS, CA, CS, SA, SC and CA do not exist, so we have -

$V_{33} = \phi \cup \{CC\}$

$V_{33} = \phi \cup \{B\}$

$\therefore \quad V_{33} = \{B\}$

The partially filled table is -

The x_{ij} table

baa	aab	aba		
ba	aa	ab	ba	
b	a	a	b	a

The V_{ij} table

ϕ	{B}	{B}		
{A, S}	{B}	{S, C}	{A, S}	
{B}	{A, C}	{A, C}	{B}	{A, C}

Step 4 :

- **Finding x_{14} and V_{14} -**

$x_{14} = baab$

Here i = 1, j = 4, k = j – 1 = 4 – 1 = 1, 2, 3

$\therefore \quad V_{ij} = V_{ik} \; V_{(i+k)\,(j-k)}$

$V_{14} = V_{11} \; V_{23} \cup V_{12} \; V_{32} \cup V_{13} \; V_{41}$

$= \{B\}\ \{B\} \cup \{A, S\}\ \{S, C\} \cup \{\phi, B\}$

$= \{BB, AS, AC, SS, SC, B\}$

As none of the above production exist,

$\therefore \quad V_{14} = \phi$

- **Finding x_{24} and V_{24} -**

$x_{24} = aaba$

Here i = 2, j = 4, k = j – 1 = 4 – 1 = 1, 2, 3

$V_{ij} = V_{ik}\ V_{(i+k)\,(j-k)}$

$V_{24} = V_{21}\ V_{33} \cup V_{22}\ V_{42} \cup V_{23}\ V_{51}$

$= \{A, C\}\ \{B\} \cup \{B\}\ \{A, S\} \cup \{B\}\ \{A, C\}$

$= \{AB, CB\} \cup \{BA, BS\} \cup \{BA, BC\}$

The CB can not be obtained

$= \{AB, BA, BS, BC\}$

$\therefore \quad V_{24} = \{S, C, A\}$

Step 5 : Finding x_{15} and V_{15} -

$x_{15} = baaba$

Here i = 1, j = 5, k = 5 – 1 = 1, 2, 3, 4

$V_{ij} = V_{ik}\ V_{(i+k)\,(j-k)}$

$V_{15} = V_{11} \cdot V_{24} \cup V_{12} \cdot V_{33} \cup V_{13} \cdot V_{42} \cup V_{14} \cdot V_{51}$

$V_{15} = \{B\}\ \{S, C, A\} \cup \{A, S\}\ \{B\} \cup \{\phi\}\ \{A, S\} \cup \{\phi\}\ \{A, C\}$

$V_{15} = \{BS, BC, BA\} \cup \{AB, SB\} \cup \{A, S\} \cup \{A, C\}$

Since BS, SB, A, S and C do not exist, so we have -

$V_{15} = \{BC, BA\} \cup \{AB\} \cup \phi \cup \phi$

$V_{15} = \{S, A\} \cup \{S, C\} \cup \phi \cup \phi$

$\therefore \quad V_{15} = \{S, A, C\}$

The table is

The x_{ij} table

baaba				
baab	aaba			
baa	aab	aba		
ba	aa	ab	ba	
b	a	a	b	a

The V_{ij} table

{S, A, C}				
ϕ	{S, C, A}			
ϕ	{B}	{B}		
{A, S}	{B}	{S, C}	{A, S}	
{B}	{A, C}	{A, C}	{B}	{A, C}

As we can see, x_{15} = baaba is a string and V_{15} = {S, A, C}

That means V_{15} contains start symbol S. Thus x_{15} = baaba is a member of given grammar. In other words, the given string baaba can be obtained by given grammar.

The value ϕ in V_{ij} table indicates that corresponding string in x_{ij} are not the members of given grammar.

Thus 'baab' and 'baa' can not be obtained from given grammar.

3.15 Applications of CFG

SPPU : May-06, 11, Dec.-06, 13, Oct.-16, 19, Marks 6

When any high level program like C or Pascal is compiled, the compiler checks the syntax of every programming statement by constructing syntax tree. And for building the syntax tree, it is necessary to write context free grammar for each statement in the program.

For example, if in your C program the statement is

x = y + z;

Then the x, y and z are identified as 'id' by lexical analyzer. It will be interpreted as

'id' = 'id' + 'id' ;

The CFG for this will be

$S \rightarrow id = ET$

$E \rightarrow E + F$

$E \rightarrow F$

$F \rightarrow id$

$T \rightarrow ;$

Let us build the parse tree accordingly.

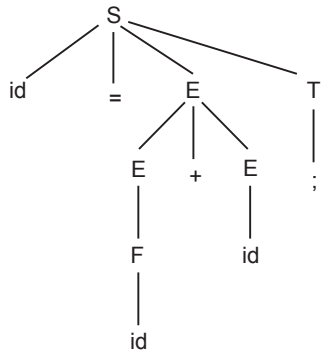


Fig. 3.15.1 Parse tree

Read out the leaf nodes and you will get id = id + id ; If ; is missing in your statement rule won't get matched with T branch and hence it will generate syntax error as 'statement ; missing'.

Thus context free grammar is a powerful tool used in compilers for syntax analysis.

1. Parsing techniques

Context free grammar is used for parsing the programming constructs and finding the syntactical errors from source program. There are two types of parser that are used by compiler top down parser and bottom up parser.

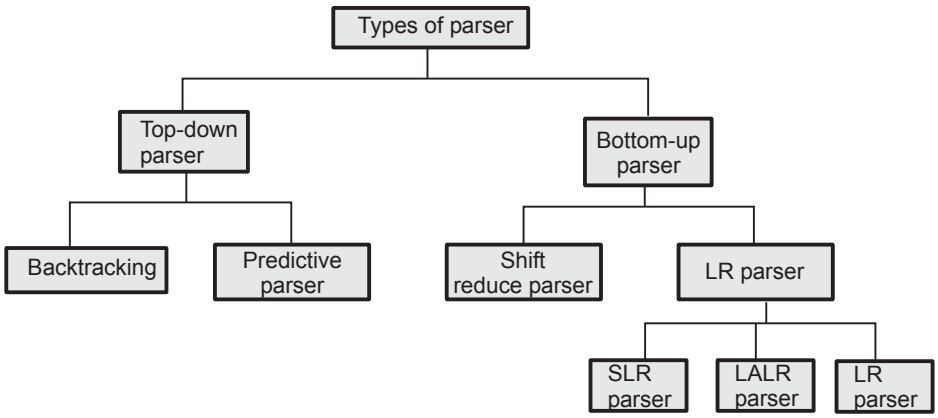


Fig. 3.15.2 Parsing techniques

When parse tree is constructed from root and expanded to leaves then such type of parser is called **Top-down parser.**

When parse tree is constructed from leaves to root, then such type of parser is called **Bottom-up parser.**

2. Markup languages

Markup languages is a family of languages in which the certain strings have special meaning. These strings are referred as **tags**. Thus tags tell us about the semantics of various strings within the document. The most commonly used example of markup language is **H**yper **T**ext **M**arkup **L**anguage (HTML). The context free language is used to describe the structure of HTML document. Consider following HTML document -

* Wonderful colors *

<h1>Red</h1>

<h2>Blue</h2>

<h3>Green</h3>

The above scripting document will help in displaying the text on the web page with special formatting. In order to check this HTML document syntactically, a context free grammar is used. Following are some variables/non terminals are used to parse this HTML document -

1. Text

It represents the collection of strings.

2. Char

It represents a single character which is allowed in the HTML document. The collection of various characters lead to a text.

3. Doc

It represents the sequence of elements.

4. Element

The element can be simple text or a text enclosed within the tags.

The production rules defining above HTML document will be -

Char→ *a* | *A* |…

Text→ *Char Text* | ε

Doc→ *Element Doc*| ε

Element→*Text* |

<h1>Doc</h1> |

<h2>Doc</h2> |

<h3>Doc</h3>|…

Thus CFG is used to describe the structure of HTML document.

3. XML and Document Type Definitions

The HTML is a markup language which makes use of some standard tags. There are some special meanings associated with these tags. But Extensible Markup Language (XML) is a special kind of language using which user can define his own tags. The purpose of XML is not to describe the formatting of HTML but to describe the semantics of the text. For example, consider following XML document.

```
<Person>
    <Personal-Info>
        <Name>My Name is Jaya</Name>
        <City>I live in Pune</City>
    </Personal-Info>
    <Hobby>
        <first>I like reading</first>
        <second>I like programming</second>
        <third>I like singing</third>
    </Hobby>
</Person>
```

Note that the XML document allows user to use user-defined tags which are useful for describing the semantics of the text. For instance : if we want to know the city in which the person lives then it becomes convenient to find out the corresponding information. The **Document Type Definition (DTD)** is a kind of context free grammar used to define the structure of the XML document. The DTD has its own notations and variables.

Using DTD we can specify the various elements types, attributes and their relationship with one another. Following example shows how DTD is useful for describing the structure of XML document -

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE student [
<!ELEMENT student (name,address,std,marks)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT std (#PCDATA)>
<!ELEMENT marks (#PCDATA)>
]>
<student>
```

Here we are defining the DTD

```
    <name>Anand</name>
    <address>Pune</address>
    <std>Second</std>
    <marks>70 percent</marks>
</student>
```

In DTD, The basic entity is **ELEMENT**. The elements are used for defining the tags. The elements typically consist of opening and closing tag. Mostly only one element is used to define a single tag. The PCDATA is used in above given XML document. It stands for **P**arsed **C**haracter **Data** (i.e. text). Any Parsable character data should not contain the markup characters. The markup characters are < or > or &. If we want to use less than, greater than or ampersand characters then make use of < , > or & The notations of context free grammar can also be used in DTD. For instance :

hobby → programming|reading|singing

can be denoted as :

<!ELEMENT hobby(programming | reading |singing)

In order to denote more than one occurrences the + sign can also be used in DTD.

Review Questions

1. *Discuss the applications of CFG in XML.*
   SPPU : May-06, Dec.-06, Marks 4, May-11, Marks 6, Oct.-19, In Sem, Marks 3
2. *Explain the application of context free grammar to syntax analysis phase of compiler with suitable example.* **SPPU : Dec.-13, Marks 6, Oct.-19, In Sem, Marks 3**
3. *Explain with examples any two applications of contest free grammar.*
   SPPU : Oct.-16, In Sem, Marks 4

3.16 Case Studies

CFG for Palindrome

- The palindrome is a condition in which we get the same string when read it from left to right or from right to left.
- Let G be CFG such that

  G = (V, T, P, S) where

  V = {S}

  T = {a, b}

S is a start symbol.

P - The production rules are

$S \rightarrow aSb \mid bSa$

$S \rightarrow a \mid b \mid \varepsilon$

- The simulation of above CFG can be shown for string abba.

S

a S a

a b S b a

a b ε b a

a b b a

CFG for Parenthesis Match

- Parenthesis match is a match made between opening and closing parenthesis.
- Let G be CFG such that

G = (V, T, P, S) where

V = {S}

T = {(,)}

S is a start symbol.

P is a set of production rules.

The rules are

$S \rightarrow SS \mid (S) \mid ( )$

- The simulation above CFG can be shown for the string **(( )) ( )**

S

SS

S( )

(S)( )

(( ))( )

3.17 Fill in the Blanks

Q.1 The CFG stands for ____.

Q.2 In CFG normally the terminals are denoted by ___ case and non terminals are denoted by ___ case.

Q.3 Ambiguous grammar is context free grammar. (true/false)

Q.4 In derivation tree the derivation is always read from _____.

Q.5 Ambiguous if grammar is _____.

Q.6 For every context free language we can construct ___ to accept the language.

Q.7 The process in which we eliminate the products which yield no terminal is called removal of ___ symbols.

Q.8 The intersection of context free language and regular language is ___.

Q.9 ___ is used to prove a language is not context free.

Q.10 $a^i b^i c^i \mid i >= 0$ is context free language (True/False)

Q.11 The pumping lemma is often used to prove that language is ____.

Q.12 ___ is the pumping length of string of length x _____.

Q.13 Expand CNF ____.

Q.14 Expand GNF ___.

3.18 Multiple Choice Questions

Q.1 Context free grammar is _____.

a compiler
b language expression
c language translator
d none of these

Q.2 Which of the following does not belong to context free grammar ?

a Non terminal symbol
b Terminal symbol
c Start symbol
d End symbol

Q.3 Regular grammar is _____.

a context free grammar
b non context free grammar
c simple english grammar
d none of theses

Q.4 S -> 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

Q.5 The CFG S---> 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

Q.6 Context free grammar is closed under _____.

a union

b kleen star

c concatenation

d all of them

Q.7 The language {} is _____.

a regular

b context free but non regular

c context sensitive but not context free

d none of these

Q.8 A grammar is ____ device.

a generative

b congnitive

c acceptor

d can't say

Q.9 The context free grammar S SS | 0S1 | 1S0 | 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

Q.10 The Chomsky normal form is _____ .

a nonterminal -> nonterminal nonterminal

b nonterminal -> terminal nonterminal

c nonterminal -> nonterminal

d none of these

Q.11 The Chomsky normal form is _____.

a nonterminal -> nonterminal non-terminal

b nonterminal -> terminal nonterminal

c nonterminal -> nonterminal

d none of these

Q.12 Given a grammar in GNF and a derivable string in the grammar with the length n, any _____will halt at depth n.

a top-down parser

b bottom-up parser

c multitape turing machine

d none of the mentioned

Q.13 In order to simplify a context free grammar, we can skip the following operation :

a Removal of null production

b Removal of useless symbols

c Removal of unit productions

d None of the mentioned

Q.14 Which among the following is a unit production ?

a A->BB

b A->B

c A->BC

d All of the above

Answer Keys for Fill in the Blanks :

Q.1	Context Free Grammar	**Q.2**	lower, upper
Q.3	True	**Q.4**	top to bottom
Q.5	having more than one derivation tree for the given input string	**Q.6**	push down automata
Q.7	useless	**Q.8**	context free language
Q.9	Regular closure	**Q.10**	False
Q.11	non context free language	**Q.12**	x+1
Q.13	Context Free Language	**Q.14**	Greibach Normal Form

Answer Keys for Multiple Choice Questions :

Q.1	b	**Q.2**	d	**Q.3**	a	**Q.4**	d
Q.5	b	**Q.6**	d	**Q.7**	b	**Q.8**	a
Q.9	a	**Q.10**	a	**Q.11**	b	**Q.12**	a
Q.13	d	**Q.14**	b				

Multiple Choice Questions Explanation

Explanation : The rules S->as|bS generates (a+b)*. In the given list of production rules S->a|b are the two rules. Hence the given CFG represents the regular expression (a+b)(a+b)*.

❑❑❑

UNIT - IV

4 Pushdown Automata (PDA)

Syllabus

Introduction, Formal definition of PDA, Equivalence of Acceptance by Final State and Empty stack, Non-deterministic PDA (NPDA), PDA and Context Free Language, Equivalence of PDA and CFG, PDA vs CFLs. Deterministic CFLs.

Contents

4.1 Formal Definition of PDA . . . . . **Dec.-17,** · · · · · · · · · · · · · · Marks 4

4.2 Instantaneous Description **May-07,11,14,15,16,18,19,**
. . . . . **Dec.-13,14,16,18,19,** · · · · Marks 16

4.3 Acceptance by Final State

4.4 Acceptance by Empty Stack. . . . . **Dec.-18, 19,** · · · · · · · · · · · Marks 4

4.5 Equivalence of Acceptance by Final State and Empty Stack
. . . . . **Dec.-09,13,** · · · · · · · · · · · Marks 18

4.6 Non-deterministic PDA (NPDA) . . . . . **Dec.-17, May-18,** · · · · · · · · Marks 6

4.7 PDA and Context Free Language **Dec.-05,08,11,14,16,17,18,19,**
. . . . . **May-14,15,16,18,**· · · · · · Marks 12

4.8 Deterministic CFLs

4.9 Fill in the Blanks

4.10 Multiple Choice Questions

4.1 Formal Definition of PDA

SPPU : Dec.-17, Marks 4

The push down automata will have input tape, finite control and stack.

The input tape is divided in many cells. At each cell only one input symbol is placed thus certain input string is placed on tape. The finite control has some pointer which point the current symbol which is to be read. At the end of input $ or Δ (blank) ƀ (blank) symbol is placed which indicates end of input.

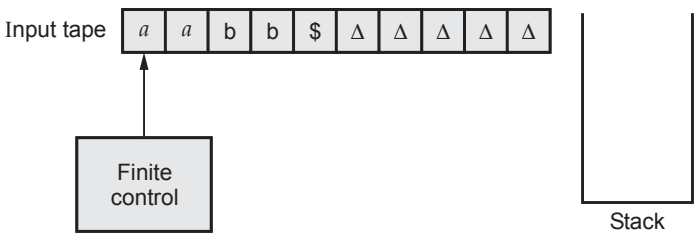


Fig. 4.1.1 Push down automata

The stack is such a structure in which you can push and remove the items from one end only. For example, if you place some coins one above the other, then a stack of coins is formed. While removing a coin we can only remove the coin which is at the top. Similarly while adding more coins to the stack we can place a new coin only on the topmost coin. Thus the stack of coin shows clear cut use of only one end of the stack.

In the push down automata, we are using the stack for storing the items temporarily. Inserting the symbol onto the stack is called push operation and removing a symbol from stack is called pop operation.

Let us have a formal 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 *Differentiate between FA and PDA.* **SPPU : Dec.-17, End Sem. Marks 4**

Solution :

Sr. No.	FA	PDA
1.	The Finite Automata (FA) has no memory. Hence it can not remember already read input.	The push-down machine has a capacity of remembering already read input by means of stack.
2.	It accepts regular language	It accepts regular non-regular and context free languages
3.	It makes use of states and transitions to model any language	It makes use of push and pop operations to model any language
4.	It is simple to represent	It is more powerful than FSM

4.2 Instantaneous Description

SPPU : May-07,11,14,15,16,18,19, Dec.-13,14,16,18,19, Marks 16

We can describe this behaviour of PDA by means of instantaneous description. It is given by ID. The moves of ID's are as shown below,

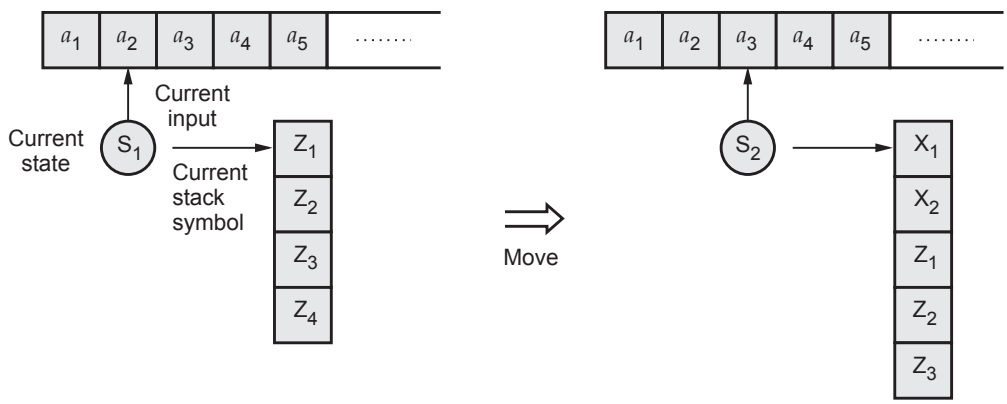
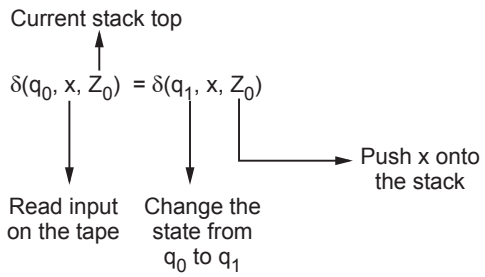
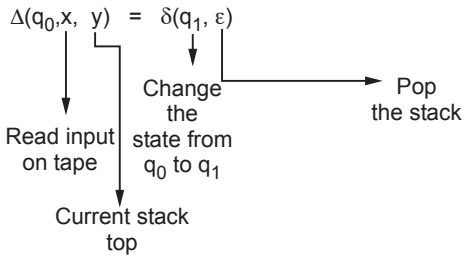


Fig. 4.2.1

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 -



(a) Push operation



(b) Pop operation

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

SPPU : May-16, End Sem. Marks 7, Dec.-16, End Sem. Marks 6, May-18,19, End Sem, Marks 6

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 -

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$

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

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

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

$$\delta(q_1, \varepsilon, Z_0) = (q_2, \varepsilon)$$

where q_0 is a start state and q_2 is accept state.

We will simulate this PDA for following string -

$$(q_0, aaabbb, Z_0) \vdash (q_0, aabbb, aZ_0)$$
$$\vdash (q_0, abbb, aaZ_0)$$
$$\vdash (q_0, bbb, aaaZ_0)$$
$$\vdash (q_1, bb, aaZ_0)$$
$$\vdash (q_1, b, aZ_0)$$
$$\vdash (q_1, \varepsilon, Z_0)$$
$$\vdash (q_2, \varepsilon)$$

ACCEPT state.

Example 4.2.2 *Construct PDA for the language $L = \{a^n b^{2n} \mid n \geq 1\}$.*

SPPU : May-11, Marks 7, Dec.-18, End Sem, Marks 6

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 -

$$\delta(q_0, a, Z_0) = (q_0, aaZ_0)$$
$$\delta(q_0, a, a) = (q_0, aaa)$$

Now when we read b we will change the state from q_0 to q_1 and start popping corresponding '*a*'. Hence,

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

Thus this process of popping will be repeated unless all the symbols are read. Note that popping action occurs in state q_1 only.

$$\therefore \quad \delta(q_1, b, a) = (q_1, \varepsilon)$$

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 -

$$\delta(q_1, \varepsilon, Z_0) = (q_2, \varepsilon)$$

where the PDA $P = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \{q_2\})$

We can summarize the ID as

$$\delta(q_0, a, Z_0) = (q_0, aaZ_0)$$
$$\delta(q_0, a, a) = (q_0, aaa)$$

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

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

$$\delta(q_1, \varepsilon, Z_0) = (q_2, \varepsilon)$$

Let us simulate this PDA for some input string "*aaa*bbbbbb".

$$\begin{aligned}
\delta(q_0, aaabbbbbb, Z_0) &\vdash (q_0, aabbbbbb, aaZ_0) \\
&\vdash (q_0, abbbbbb, aaaaaZ_0) \\
&\vdash (q_0, bbbbbb, aaaaaaZ_0) \\
&\vdash (q_1, bbbbb, aaaaaZ_0) \\
&\vdash (q_1, bbbb, aaaaZ_0) \\
&\vdash (q_1, bbb, aaaZ_0) \\
&\vdash (q_1, bb, aaZ_0) \\
&\vdash (q_1, b, aZ_0) \\
&\vdash (q_1, \varepsilon, Z_0) \\
&\vdash (q_2, \varepsilon)
\end{aligned}$$

Final state or ACCEPT state.

Thus input gets accepted by using the constructed PDA.

Example 4.2.3 *Construct the PDA and then its equivalent CFG for the language L as given below :*

$L = \{XcX^r \mid X \in \{a,b\}^*$ and string X^r is the reverse of string $X\}$

SPPU : May-14, Marks 16, Dec.-19, End Sem, Marks 6

Solution : For a given PDA,

$M = \left(\{q_0, q_1, q_2\}, \{a, b, c\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_2\}\right)$, the mapping function δ is given as

$$\begin{aligned}
R_1 &: \delta(q_0, a, Z_0) = \delta(q_0, aZ_0) \\
R_2 &: \delta(q_0, b, Z_0) = \delta(q_0, bZ_0) \\
R_3 &: \delta(q_0, a, a) = \delta(q_0, aa) \\
R_4 &: \delta(q_0, b, a) = \delta(q_0, ba) \\
R_5 &: \delta(q_0, a, b) = \delta(q_0, ab) \\
R_6 &: \delta(q_0, b, b) = \delta(q_0, bb) \\
R_7 &: \delta(q_0, c, Z_0) = \delta(q_1, Z_0)
\end{aligned}$$

$R_8 : \delta(q_0, c, a) = \delta(q_1, a)$

$R_9 : \delta(q_0, c, b) = \delta(q_1, b)$

$R_{10} : \delta(q_1, a, a) = \delta(q_1, \varepsilon)$

$R_{11} : \delta(q_1, b, b) = \delta(q_1, \varepsilon)$

$R_{12} : \delta(q_1, \varepsilon, Z_0) = \delta(q_2, Z_0)$

We will first draw the transition graph for given δ transitions.

Now we will simulate this PDA for the input bb*aca*bb.

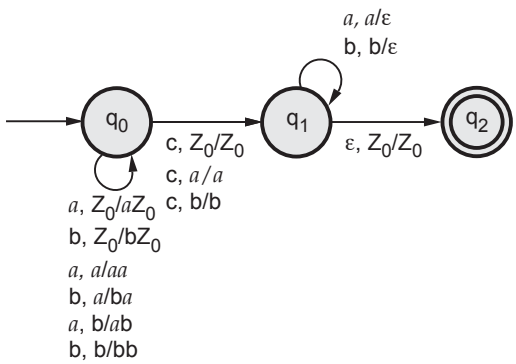


Fig. 4.2.3

From initial ID, we will try to match rules R_1 to R_{12}.

$(q_0, \underline{b}bacabb, \underline{Z}_0) \vdash (q_0, \underline{b}acabb, \underline{b}Z_0)$

$\vdash (q_0, \underline{a}cabb, \underline{b}bZ_0)$

$\vdash (q_0, cabb, abbZ_0)$

$\vdash (q_1, \underline{a}bb, \underline{a}bbZ_0)$

$\vdash (q_1, \underline{b}b, \underline{b}bZ_0)$

$\vdash (q_1, b, bZ_0)$

$\vdash (q_1, \varepsilon, Z_0)$

$\vdash (q_2, Z_0)$

which is ACCEPT state.

Thus the input bb*aca*bb is accepted by given PDA.

Example 4.2.4 *Design PDA to accept the language*

$$L = \{w / w \in (a + b)^* \text{ and } n_a(w) = n_b(w)\}.$$

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.

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$

$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$

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

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

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

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

$$\delta(q_0, \varepsilon, Z_0) = (q_1, Z_0)$$

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 '*aa*b*a*bb'.

$$\begin{aligned}
\delta(q_0, aababb, Z_0) &\vdash (q_0, ababb, aZ_0) \\
&\vdash (q_0, babb, aaZ_0) \\
&\vdash (q_0, abb, aZ_0) \\
&\vdash (q_0, bb, aaZ_0) \\
&\vdash (q_0, b, aZ_0) \\
&\vdash (q_0, \varepsilon, Z_0) \\
&\vdash (q_1, Z_0) \\
&\quad \text{ACCEPT state.}
\end{aligned}$$

Example 4.2.5 *Design a PDA for the language*

$L = \{w \mid w \in (a + b)^* \text{ and } n_a(w) > n_b(w)\}$.

Solution :

- **Logic**

$n_a(w)$ 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,

- **Design**

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$

$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$

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

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

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

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

$$\delta(q_0, \varepsilon, a) = (q_f, a)$$

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 $aabaabab$.

$$\begin{aligned}
\delta(q_0, aabaabab, Z_0) &\vdash (q_0, abaabab, aZ_0) \\
&\vdash (q_0, baabab, aZ_0) \\
&\vdash (q_0, abab, aZ_0) \\
&\vdash (q_0, bab, aaZ_0) \\
&\vdash (q_0, ab, aZ_0) \\
&\vdash (q_0, b, aaZ_0) \\
&\vdash (q_0, \varepsilon, aZ_0) \\
&\vdash (q_f, a) \\
&\quad \text{Accept state.}
\end{aligned}$$

Thus input gets accepted.

Example 4.2.6 *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 -

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$

$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$

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

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

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

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

$$\delta(q_0, \varepsilon, b) = (q_f, b) \leftarrow \text{The b's are more in number.}$$

Now we will **simulate** this PDA for the input "*a*bb*a*b".

$$\begin{aligned}\delta(q_0, abbab, Z_0) &\vdash (q_0, bbab, aZ_0)\\ &\vdash (q_0, bab, Z_0)\\ &\vdash (q_0, ab, bZ_0)\\ &\vdash (q_0, b, Z_0)\\ &\vdash (q_0, \varepsilon, bZ_0)\\ &\vdash (q_f, b)\end{aligned}$$

Accept state.

Example 4.2.7 *Design a PDA that accepts a string of well formed parenthesis. Consider the parenthesis is as (,), [,], { , }.*

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.

$$\therefore \quad \delta(q_0, (, Z_0) \vdash (q_1, (Z_0)$$

$$\delta(q_0, [, Z_0) \vdash (q_1, [Z_0)$$

$$\delta(q_0, \{, Z_0) \vdash (q_1, \{Z_0)$$

Further if we read more left parenthesis then we will simply push them onto the stack.

$$\delta(q_1, (, ( ) = (q_1, (( )$$
$$\delta(q_1, [, [ ) = (q_1, [[ )$$
$$\delta(q_1, \{, \{) = (q_1, \{\{)$$
$$\delta(q_1, (, [ ) = (q_1, ([ )$$
$$\delta(q_1, (, \{) = (q_1, (\{)$$
$$\delta(q_1, [, ( ) = (q_1, [( )$$
$$\delta(q_1, \{, ( ) = (q_1, \{( )$$
$$\delta(q_1, \{, [ ) = (q_1, \{[ )$$
$$\delta(q_1, [, \{) = (q_1, [\{)$$

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.

$$\therefore \quad \delta(q_1,), ( ) = (q_1, \varepsilon)$$
$$\delta(q_1,], [ ) = (q_1, \varepsilon)$$
$$\delta(q_1, \}, \{) = (q_1, \varepsilon)$$

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 -

$$\delta(q_1, \varepsilon, Z_0) = (q_0, Z_0)$$

To summarize these steps we will write ID as -

$$\delta(q_0, (, Z_0) = (q_1, (Z_0)$$
$$\delta(q_0, [, Z_0) = (q_1, [Z_0)$$
$$\delta(q_0, \{, Z_0) = (q_1, \{Z_0)$$
$$\delta(q_1, (, ( ) = (q_1, (( )$$
$$\delta(q_1, [, [ ) = (q_1, [[ )$$
$$\delta(q_1, \{, \{) = (q_1, \{\{)$$
$$\delta(q_1, (, [ ) = (q_1, ([ )$$
$$\delta(q_1, (, \{) = (q_1, (\{)$$
$$\delta(q_1, [, ( ) = (q_1, [( )$$

$$\delta(q_1, \{, ()) = (q_1, \{()$$

$$\delta(q_1, \{, [) = (q_1, \{[)$$

$$\delta(q_1, [, \{) = (q_1, [\{)$$

$$\delta(q_1,), ()) = (q_1, \varepsilon)$$

$$\delta(q_1,], [) = (q_1, \varepsilon)$$

$$\delta(q_1, \}, \{) = (q_1, \varepsilon)$$

$$\delta(q_1, \varepsilon, Z_0) = (q_0, Z_0)$$

The PDA $P = \left(\{q_0, q_1\}, \{(, [, \{,),], \}\}, \{[, (, \{, Z_0\}, \delta, q_0, Z_0, \{q_0\}\right)$

Let us simulate it for some input string.

$$(\{\}[])$$

$$\begin{aligned}
\delta(q_0, (\{\}[]), Z_0) &\vdash (q_1, \{\}[]), (Z_0) \\
&\vdash (q_1, \}[]), \{(Z_0) \\
&\vdash (q_1, []), (Z_0) \\
&\vdash (q_1,]), [(Z_0) \\
&\vdash (q_1,), (Z_0) \\
&\vdash (q_1, \varepsilon, Z_0) \\
&\quad (q_0, Z_0) \\
&\quad \text{ACCEPT state.}
\end{aligned}$$

Example 4.2.8 *Construct PDA for the language* $\left\{L = a^m\ b^m\ c^n \mid m, n \geq 1\right\}$.

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.

$$\delta(q_0, a, z_0) = \delta(q_0, a Z_0)$$

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

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

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

$$\delta(q_1, c, Z_0) = \delta(q_1, Z_0)$$

$\delta(q_1, \varepsilon, Z_0) = \delta(q_2, Z_0)$

Let us derive this PDA for the same example.

$$\begin{aligned}
\delta(q_0, aabbccc, Z_0) &\vdash \delta(q_0, abbccc, aZ_0) \\
&\vdash \delta(q_0, bbccc, aaZ_0) \\
&\vdash \delta(q_1, bccc, aZ_0) \\
&\vdash \delta(q_1, ccc, Z_0) \\
&\vdash \delta(q_1, cc, Z_0) \\
&\vdash \delta(q_1, c, Z_0) \\
&\vdash \delta(q_1, \varepsilon, Z_0) \\
&\vdash \delta(q_2, Z_0) \\
&\quad \text{ACCEPT}
\end{aligned}$$

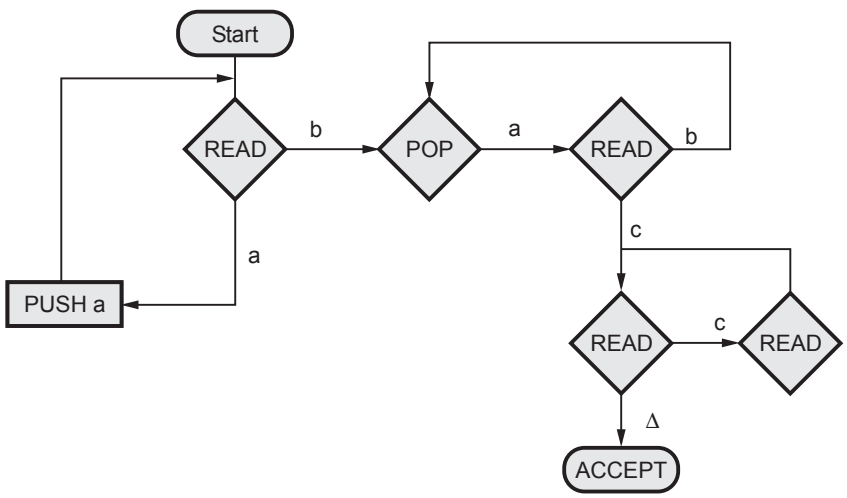


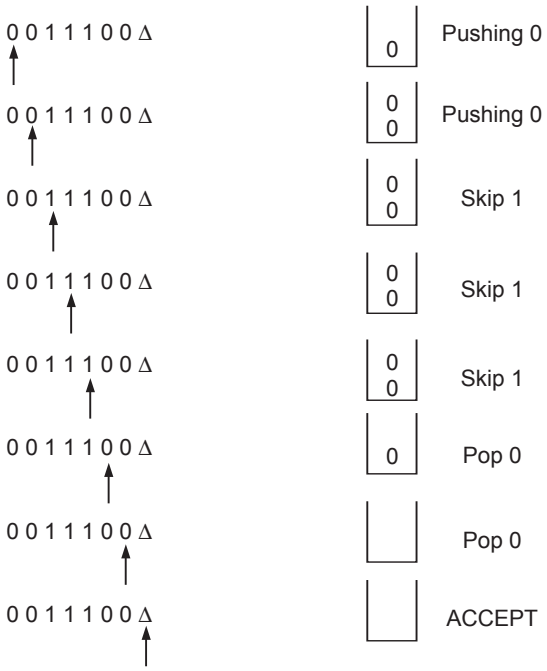
Fig. 4.2.4 PDA for $a^m b^m c^n$

Example 4.2.9 *Build a PDA for the language*

$L = \{0^n\ 1^m\ 0^n \mid m, n \geq 1\}$ *by empty stack.* **SPPU : Dec.-13, Marks 7**

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,



$$\delta(q_0, 0, Z_0) = \delta(q_0, 0Z_0)$$

$$\delta(q_0, 0, 0) = \delta(q_0, 00)$$

$$\delta(q_0, 1, 0) = \delta(q_1, 0)$$

$$\delta(q_0, 1, 0) = \delta(q_1, 0)$$

$$\delta(q_1, 0, 0) = \delta(q_1, \varepsilon)$$

$$\delta(q_0, \varepsilon, Z_0) = \delta(q_2, Z_0) \leftarrow \text{ACCEPT state}$$

For example :

$$\begin{aligned}
\delta(q_0, 0011100, Z_0) &\vdash \delta(q_0, 011100, 0Z_0) \\
&\vdash \delta(q_0, 11100, 00Z_0) \\
&\vdash \delta(q_0, 1100, 00Z_0) \\
&\vdash \delta(q_1, 100, 00Z_0) \\
&\vdash \delta(q_1, 00, 00Z_0) \\
&\vdash \delta(q_1, 0, 0Z_0) \\
&\vdash \delta(q_1, \varepsilon, Z_0) \\
&\vdash \delta(q_2, Z_0) \\
&\quad \text{ACCEPT}
\end{aligned}$$

Example 4.2.10 *Construct pushdown automata for each of the following language*

1) The set of palindromes over alphabet {a, b}

2) The set of all string over alphabet {a, b} with exactly twice many a's as b's

3) $\{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}$

SPPU : May-07, Marks 12, Dec.-16, End Sem. Marks 10, May-18, End Sem, Marks 6

Solution :

1) Refer example 4.2.3.

2) We will apply a simple logic to design this PDA if we read first 'a' then simply change the state from q_0 to q_1. In state q_1 if a is read then push 'A', and for this second 'a', change the state from q_1 to q_0 again.

If we read single b when we are there in state q_0 then push B onto the stack. This state indicates that we are waiting for two a's to read out. On receiving two a's, this B will be popped.

The simple rule in the design of this PDA is : State q_1 indicates that a single 'a' is read so far now. The δ transitions for this PDA are as given below -

1. $\delta(q_0, a, Z_0) = (q_1, Z_0)$
2. $\delta(q_1, a, Z_0) = (q_0, AZ_0)$ i.e. push A on reading second a and change the state to q_0
3. $\delta(q_0, b, Z_0) = (q_0, BZ_0)$ i.e. initially simply push B on reading single B.
4. $\delta(q_1, b, Z_0) = (q_1, BZ_0)$ i.e. push B on reading b
5. $\delta(q_0, a, A) = (q_1, A)$ i.e. simply read the a and change state to q_1.
6. $\delta(q_0, a, B) = (q_1, B)$ i.e. one b is already read and now one a is read. so keep B as it is
7. $\delta(q_0, b, B) = (q_0, BB)$ i.e. push B for reading a
8. $\delta(q_1, a, B) = (q_0, \varepsilon)$ i.e. read two a's for single b.
9. $\delta(q_1, a, A) = (q_0, AA)$ i.e. push A onto the stack.
10. $\delta(q_0, b, A) = (q_0, \varepsilon)$ pop A from stack.
11. $\delta(q_1, b, B) = (q_0, BB)$ push B for reading b.
12. $\delta(q_0, \varepsilon, Z_0) = (q_0, \varepsilon)$ **ACCEPT**.

Simulation for abbaaa

$$\delta(q_0, \underline{a}bbaaa, \underline{Z}_0) = (q_1, \cancel{a}\underline{b}baaa, Z_0) \quad \text{rule 1}$$
$$= (q_1, \cancel{a}\cancel{b}baaa, BZ_0) \quad \text{rule 4}$$
$$= (q_1, \cancel{a}\cancel{b}\underline{b}aaa, BZ_0) \quad \text{rule 11}$$
$$= (q_1, \cancel{a}\cancel{b}\cancel{b}\underline{a}aa, \underline{B}BZ_0) \quad \text{rule 8}$$
$$= (q_0, \cancel{a}\cancel{b}\cancel{b}\cancel{a}\underline{a}a, BZ_0) \quad \text{rule 6}$$
$$= (q_1, \cancel{a}\cancel{b}\cancel{b}\cancel{b}\underline{a}, \underline{B}Z_0) \quad \text{rule 8}$$
$$= (q_0, \cancel{a}\cancel{b}\cancel{b}\cancel{a}\cancel{a}\cancel{a}, Z_0) \quad \text{rule 12}$$

ACCEPT.

3) Let, $L = \left\{a^i b^j c^k \mid i \neq j \quad \text{or} \quad j \neq k\right\}$

Note that this is an or condition. If we consider the first condition i.e. $i \neq j$ then we have to focus on a's and b's only. At the same time any number of c's are allowed. Here $i > j$ or $j > i$ are the two applicable conditions.

Similarly in the second condition $j \neq k$ then we have to focus on b's and c's only. At the same time any number of a's are allowed.

Here $j > k$ or $k > j$ are the two applicable conditions.

We will first write the CFG for these conditions.

$$S \rightarrow P|Q|R|T|M$$

Let, A,B and C are the non terminals that generate a^+, b^+ and c^+ respectively.

$$A \rightarrow a\,A|a$$
$$B \rightarrow b\,B|b$$
$$C \rightarrow c\,C|c$$

i) P generates $a^i\, b^j\, c^k$ such that $i > j$

$$P \rightarrow A\,D|A\,D\,C$$
$$D \rightarrow a\,D\,b|\varepsilon \qquad \text{Here } D = a^n\, b^n \text{ where } n \geq 0.$$

ii) Q generates $a^i\, b^j\, c^k$ such that $j > i$

$$Q \rightarrow D\,B|D\,B\,C$$

iii) R generates $a^i\, b^j\, c^k$ such that $j > k$

$$R \rightarrow B\,E|A\,B\,E$$
$$E \rightarrow b\,E\,c|\varepsilon \qquad \text{Here } E = b^n\, c^n \text{ where } n \geq 0$$

iv) M generates $a^i\, b^j\, c^k$ such that $k > j$

$$M \rightarrow E\ C \mid A\ E\ C$$

From these above given production rules we can write δ for PDA as follows.

$$(q, \varepsilon, S) = \{(q,P),(q,Q),(q,R),(q,T),(q,M)\}$$
$$(q, \varepsilon, A) = \{(q,aA),(q,a)\}$$
$$(q, \varepsilon, B) = \{(q,bB),(q,b)\}$$
$$(q, \varepsilon, C) = \{(q,cC),(q,c)\}$$
$$(q, \varepsilon, P) = \{(q,AD),(q,ADC)\}$$
$$(q, \varepsilon, D) = \{(q,aDb),(q,\varepsilon)\}$$
$$(q, \varepsilon, Q) = \{(q,DB),(q,DBC)\}$$
$$(q, \varepsilon, R) = \{(q,BE),(q,ABE)\}$$
$$(q, \varepsilon, E) = \{(q,bEc),(q,\varepsilon)\}$$
$$(q, \varepsilon, M) = \{(q,E,C),(q,AEC)\}$$
$$(q, a, a) = \{(q,\varepsilon)\}$$
$$(q, b, b) = \{(q,\varepsilon)\}$$
$$(q, c, c) = \{(q,\varepsilon)\}$$

Example 4.2.11 *Design PDA for a language.*

$L = \{w(a + b)w^R \mid w \in (a + b)^*$ and w^R is reverse of $w\}$

SPPU : Dec.-14, End Sem, Marks 8

Solution : The PDA is a nondeterministic PDA. It can be represented as follows -

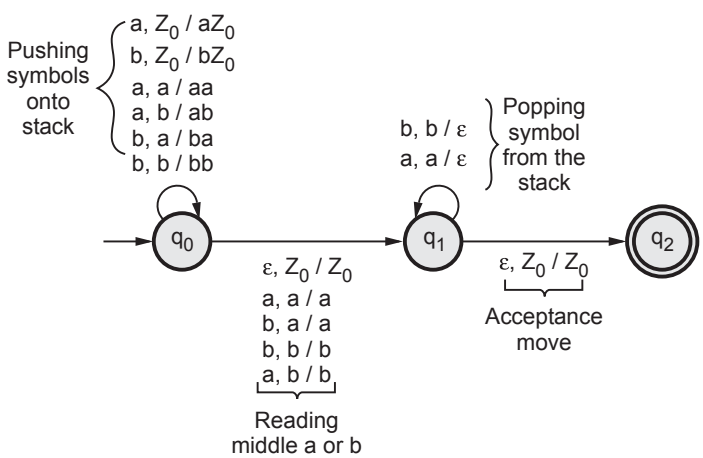


Fig. 4.2.5

Example 4.2.12 *Construct transition table for PDA that accepts the languages $L = \{a^{2n} b^n \mid n >= 1\}$. Trace your PDA for the input with n = 3.*

SPPU : May-15, End Sem, Marks 10

Solution : The logic for building this PDA is as follows :

Just read one a and push and change state. Then read and push next a onto the stack. Repeat this for all strings of a's. On reading each b pop one a from stack.

The δ function can be as given below.

$\delta(q_0, a, Z_0) = (q_1, Z_0)$		Simply read a and change state.
$\delta(q_1, a, Z_0) = (q_0, AZ_0)$		Push a onto the stack as A. Change state to q_0.
$\delta(q_0, a, A) = (q_1, A)$		Simply read a and change state.
$\delta(q_1, a, A) = (q_1, AA)$		Push a as A
$\delta(q_0, b, A) = (q_0, \varepsilon)$		POP A
$\delta(q_0, \varepsilon, Z_0)$		ACCEPT.

Simulation for n = 3 i.e. aaaaaabbb

$$
\begin{aligned}
(q_0, aaaaaabbb, Z_0) &\vdash (q_1, aaaaabbb, Z_0) \\
&\vdash (q_0, aaaa\,bbb, AZ_0) \\
&\vdash (q_1, aaa\,bbb, AZ_0) \\
&\vdash (q_0, aa\,bbb, AAZ_0) \\
&\vdash (q_1, abbb, AAZ_0) \\
&\vdash (q_0, bbb, AAAZ_0) \\
&\vdash (q_0, bb, AAZ_0) \\
&\vdash (q_0, b, AZ_0) \\
&\vdash (q_0, \varepsilon, Z_0) \\
&\vdash \text{ACCEPT}
\end{aligned}
$$

4.3 Acceptance by Final State

SPPU : Dec.-18, 19, Marks 4

The PDA accepts its input by consuming it and then enters in the final state. The formal definition is,

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. Then the language L(P) is called the **language accepted by final state.**

The language L(P) can be defined as,

$$L(P) = \{w \mid (q_0, w, Z_0) \overset{*}{\vdash} (p, \varepsilon, \alpha)\}$$

i.e. the PDA P reads the entire input w and enters in a final state p.

4.4 Acceptance by Empty Stack

On reading the input string from initial configuration for some PDA, the stack of PDA gets empty. The formal definition is -

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be the PDA then the **language accepted by empty stack N(P)** is defined as

$$N(P) = \{w \mid (q_0, w, Z_0) \overset{*}{\vdash} (p, \varepsilon, \varepsilon)\}$$

Review Question

1. *Explain acceptance by PDA - i) By final state ii) By empty state.*

SPPU : Dec.-18, 19, End Sem, Marks 4

4.5 Equivalence of Acceptance by Final State and Empty Stack

SPPU : Dec.-09,13, Marks 18

1. If $L = N(P_1)$ for some PDA P_1 then there is a PDA P_2 such that $L = L(P_2)$. That means the language accepted by empty stack PDA will also be accepted by final state PDA.
2. If there is a language $L = L(P_1)$ for some PDA P_1 then there is a PDA P_2 such that $L = N(P_2)$. That means language accepted by final state PDA is also acceptable by empty stack PDA.

Theorem 1 : The language accepted by empty stack PDA is also acceptable by final state PDA.

Proof : Let $P_1 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \phi)$ be a PDA, accepted by empty stack.

Now, construct a PDA P_2 using P_1.

The construction will be -

$$P_2 = (Q \cup \{q_0', q_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta', q_0', X_0, \{q_f\})$$

Where, the mapping function is given by -

Rule 1) $\delta'(q_0', \varepsilon, X_0) = \{(q_0, Z_0 X_0)\}$

Rule 2) $\delta'(q, a, Z) = \delta(q, a, z)$ for all q in Q and a in $\Sigma \cup \{\varepsilon\}$ and Z in Γ.

Rule 3) $\delta'(q, \varepsilon, X_0) = \{(q_f, \varepsilon)\}$ for all q in Q.

Thus PDA P_2 first enters into the initial configuration of P_1 without using any input symbol (i.e. ε transition). Then using Rule 2 the simulation of P_1 is carried out. When P_1 has an empty stack then P_2 enters in final state q_f (Refer Rule 3).

This shows that for every PDA accepting by empty stack there exists equivalent PDA by final state.

Theorem 2 : The language accepted by final state PDA is also acceptable by empty stack PDA.

Proof : Let, $P_1 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be the PDA which accepts the language by empty stack.

Now, we will construct PDA P_2 using P_1. The construction will be -

$$P_2 = (Q \cup \{q_0', q_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta', q_0', X_0, \phi\}$$

Where,

Rule 1) $\delta'(q_0', \varepsilon, X_0) = \{q_0, Z_0 \ X_0\}$

Rule 2) $\delta'(q, a, Z) = \delta(q, a, Z)$ for all q in Q, *a* in Σ and Z in Γ.

Rule 3) $\delta'(q, \varepsilon, Z) = \delta(q, \varepsilon, Z) \cup \{(d, \varepsilon)\}$ for all $Z \in \Gamma \cup X_0$ and $q \in F$ and d is a new dead state.

Rule 4) $\delta'(d, \varepsilon, Z) = \{d, \varepsilon\}$ for all $Z \in \Gamma \cup X_0$.

Thus PDA P_2 enters into initial configuration of P_1 without consuming any input symbols. When some symbol is read then using Rule 2 the PDA P_1 is simulated and it reaches to final state of PDA P_1. The PDA P_2 at this stage simply makes a guess about the input string whether it is terminated or not.

If input string is not terminated, P_2 again simulates P_1, otherwise P_2 enters the dead state d (Rule 3). Finally using Rule 4 using ε move,

P_2 erases all symbols and empties the stack. This shows that for every PDA accepting by final state there exists equivalent PDA by empty stack.

Example 4.5.1 *Let $L = \{a^j b^j c^k / i, j, k \geq$ and $i + j = k\}$*

a) Find a PDA (which accepts via final state) that recognizes L

b) Find a PDA (which accepts via empty stack) that recognizes L.

SPPU : Dec.-09, Marks 18

Solution : Let, $L = \{a^i b^j c^k \mid i+j = k\}$ be a given language. The simple logic for designing this PDA will be - just read a's and push A for each corresponding a. Then read b's and push A for each corresponding b. Now on reading each c just pop a single A from stack.

a) Acceptance via final state.

$$\delta(q_0, a, Z_0) = (q_1, AZ_0)$$
$$\delta(q_1, a, A) = (q_1, AA)$$
$$\delta(q_0, b, Z_0) = (q_2, AZ_0)$$
$$\delta(q_1, b, A) = (q_2, AA)$$
$$\delta(q_2, b, A) = (q_2, AA)$$

no 'a' is present $b^j c^k$ is a string.

Pushing A onto the stack

$$\left.\begin{aligned}\delta(q_2, c, A) &= (q_3, \varepsilon)\\ \delta(q_3, c, A) &= (q_3, \varepsilon)\end{aligned}\right\}$$ popping A from the stack.

$\delta(q_3, \varepsilon, Z_0) = (q_3, \varepsilon) \rightarrow$ Acceptance by final state q_3.

b) Acceptance via empty stack.

$$\delta(q_0, a, Z_0) = (q_1, A\,Z_0)$$
$$\delta(q_1, a, A) = (q_1, A\ A)$$
$$\delta(q_0, b, Z_0) = (q_2, A\,Z_0)$$
$$\delta(q_1, b, A) = (q_2, A\,A)$$
$$\delta(q_2, b, A) = (q_2, A\,A)$$
$$\delta(q_2, c, A) = (q_3, \varepsilon)$$
$$\delta(q_3, c, A) = (q_3, \varepsilon)$$
$$\delta(q_3, \varepsilon, Z_0) = (q_0, \varepsilon) \rightarrow$$ Acceptance by empty stack.

Review Question

1. *Give formal definition of -*
   i) Pushdown Automata (PDA)
   ii) Acceptance of PDA in terms of final state
   iii) Acceptance of PDA in terms of null store
   iv) Instantaneous description of PDA. **SPPU : Dec.-13, Marks 6**

4.6 Non-deterministic PDA (NPDA)

SPPU : Dec.-17, May-18, Marks 6

The Non deterministic PDA (NPDA) is a 7 tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where Q is a finite set of states, Σ is input set, Γ is a set of stack symbols. The δ is a mapping function. q_0 is a initial state, Z_0 is a stack symbol and F is set of finite states A NPDA is like NFA, except it has stack. There can be more than one moves possible on reading the input symbol.

NPDA is strictly more powerful than deterministic PDA.

Example 4.6.1 *Construct NPDA that accepts the language generated by $S = S + S \mid S * S \mid 4$.*

SPPU : Dec.-17, End Sem, Marks 6

Solution : The PDA can be,

$$A = \{\{q\}, \{*, 4 +\}, \{S, 4, +, *\}, \delta, q, S, \phi\}$$

The δ can be

R1 : $\delta(q, \varepsilon, S) = \{(q, S + S), (q, S * S\}, (q, 4)\}$

R2 : $\delta(q, \varepsilon, 4) = \{(q, \varepsilon)\}$

R3 : $\delta(q, \varepsilon, +) = \{(q, \varepsilon)\}$

R4 : $\delta(q, \varepsilon, *) = \{(q, \varepsilon)\}$

Thus the generated PDA is NPDA because there are more than one moves for same input symbol and from same current state.

Example 4.6.2 *Construct a NPDA that accepts the language $L = \{a^{2n} \mid n > 0\}$*

SPPU : May-18, End Sem, Marks 6

Solution : The ID for PDA is,

$$\left.\begin{array}{l}\delta(q_0, a, Z_0) = (q_1, aZ_0) \\ \delta(q_1, a, a) = (q_1, aa)\end{array}\right\} \text{Push}$$

$$\left.\begin{array}{l}\delta(q_1, a, a) = (q_2, \varepsilon) \\ \delta(q_2, a, a) = (q_3, \varepsilon) \\ \delta(q_2, \varepsilon, Z_0) = (q_3, \varepsilon)\end{array}\right\} \text{POP}$$

$$\delta(q_3, \varepsilon, Z_0) = (q_3, \varepsilon) \quad \text{ACCEPT}$$

Review Question

1. *Explain closure property of CFL with suitable example.*

SPPU : Dec.-17, End Sem, Marks 6

4.7 PDA and Context Free Language

SPPU : Dec.-05,08,11,14,16,17,18,19, May-14,15,16,18, Marks 12

4.7.1 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

   $\delta(q_i, a, Z_0) = (q_{i+1}, \varepsilon)$ can be converted as $(q_i Z_0 q_{i+1}) \rightarrow a$

Example 4.7.1 *Consider the PDA with following moves.*

$\delta(q_0, a, Z_0) = \{(q_0, a, Z_0)\}$

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

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

$\delta(q_1, b, a) = \{(q_1, e)\}$

$\delta(q_0, e, Z_0) = \{(q_1, e)\}$

Obtain CFG equivalent to PDA

SPPU : Dec.-08, Marks 8, May-14, Marks 12, Dec.-19, End Sem, Marks 6

Solution : 1) We will create production rules for start symbol, S. These are as follow :

$S \rightarrow (q_0, Z_0, q_0)$

$S \rightarrow (q_0, Z_0, q_1)$

2) Consider the rule $\delta(q_0, a, Z_0) = (q_0, aZ_0)$

i) $(q_0, Z_0, q_0) \rightarrow a\,(q_0, a, q_0)(q_0, Z_0, q_0)$

ii) $(q_0, Z_0, q_0) \rightarrow a\,(q_0, a, q_1)(q_1, Z_0, q_0)$

iii) $(q_0, Z_0, q_1) \rightarrow a\,(q_0, a, q_0)(q_0, Z_0, q_1)$

iv) $(q_0, Z_0, q_1) \rightarrow a\,(q_0, a, q_1)(q_1, Z_0, q_1)$

3) Consider the rule $\delta(q_0, a, a) = (q_0\ a\ a)$

v) $(q_0, a, q_0) \rightarrow a\,(q_0, a, q_0)(q_0, a, q_0)$

vi) $(q_0, a, q_0) \rightarrow a\,(q_0, a, q_1)(q_1, a, q_0)$

vii) $(q_0, a, q_1) \rightarrow a\,(q_0, a, q_0)(q_0, a, q_1)$

viii) $(q_0, a, q_1) \rightarrow a\,(q_0, a, q_1)\,(q_1, a, q_1)$

4) Consider the rule $(q_0, b, a) = (q_1, \varepsilon)$

ix) $(q_0, a, q_1) \rightarrow b$

5) Consider the rule $(q_1, b, a) = (q_1, \varepsilon)$

x) $(q_1, a, q_1) \rightarrow b$

6) Consider the rule $(q_0, \varepsilon, Z_0) = (q_1, \varepsilon)$

xi) $(q_0, Z_0, q_1) \rightarrow \varepsilon$

If we assume the state names as follows -

$(q_0, Z_0, q_0) = A$		$S \rightarrow A \mid B$
$(q_0, Z_0, q_1) = B$	Above obtained	$A \rightarrow a\,CA \mid a\,DE$
$(q_0, a, q_0) = C$	$\xrightarrow[\text{rules}]{\text{Conversion}}$	$B \rightarrow a\,CB \mid a\,DF$
$(q_1, a, q_1) = D$		$C \rightarrow a\,CC \mid a\,DG$
$(q_1, Z_0, q_0) = E$		$D \rightarrow a\,CD \mid a\,DH$
$(q_1, Z_0, q_1) = F$		$D \rightarrow b$
$(q_1, a, q_0) = G$		$H \rightarrow b$
$(q_1, a, q_1) = H$		$B \rightarrow \varepsilon$

The rules for E, F, G are missing. Hence we will eliminate production rules on RHS containing E, F or G non terminals. Finally the production rules for the equivalent grammar are

$$S \rightarrow A \mid B$$

$$A \rightarrow aCA$$

$$B \rightarrow aCB \mid \varepsilon$$

$$C \rightarrow aCC$$

$$D \rightarrow aCD \mid aDH \mid b$$

$$H \rightarrow b$$

Example 4.7.2 *Obtain CFG for the PDA given below*

$\delta(q_0, 1, Z_0) = \{q_0, XZ_0)$

$\delta(q_0, 1, X) = (q_0, XX)$

$\delta(q_0, 0, X) = (q_1, X)$

$\delta(q_0, e, Z_0) = (q_0, e)$

$\delta(q_1, 1, X) = (q_1, e)$

$\delta(q_0, 1, Z_0) = (q_0, Z_0)$ **SPPU : May-15,16, End Sem, Dec.-16, End Sem. Marks 9**

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 $\delta(\mathbf{q_0, 1, Z_0}) = (\mathbf{q_0, XZ_0})$

i) $(q_0, Z_0, q_0) \rightarrow 1(q_0, X, q_0)(q_0, Z_0, q_0)$

ii) $(q_0, Z_0, q_0) \rightarrow 1(q_0, X, q_0)(q_0, Z_0, q_0)$

iii) $(q_0, Z_0, q_1) \rightarrow 1(q_0, X, q_0)(q_0, Z_0, q_1)$

iv) $(q_0, Z_0, q_1) \rightarrow 1(q_0, X, q_1)(q_1, Z_0, q_1)$

3) Consider the rule $\delta(\mathbf{q_0, 1, X}) = (\mathbf{q_0\ X\ X})$

v) $(q_0, X, q_0) \rightarrow 1\ (q_0, X, q_0)(q_0, X, q_0)$

vi) $(q_0, X, q_0) \rightarrow 1\ (q_0, X, q_1)(q_1, X, q_0)$

vii) $(q_0, X, q_1) \rightarrow 1\ (q_0, X, q_0)(q_0, X, q_1)$

viii) $(q_0, X, q_1) \rightarrow 1\ (q_0, X, q_1)(q_1, X, q_1)$

4) Consider the rule $\delta(q_0, 0, X) = (q_1, X)$

ix) $(q_0, X, q_0) \rightarrow 0(q_0, X, q_0)$

x) $(q_0, X, q_1) \rightarrow 0(q_0, X, q_1)$

5) Consider the rule $\delta(\mathbf{q_0, \varepsilon, Z_0}) = (\mathbf{q_0, \varepsilon})$

xi) $(q_0, Z_0, q_0) \rightarrow \varepsilon$

6) Consider the rule $\delta(\mathbf{q_1, 1, X}) = (\mathbf{q_1, \varepsilon})$

xii) $(q_1, X, q_1) \rightarrow 1$

7) Consider the rule $\delta(\mathbf{q_0, 1, Z_0}) = (\mathbf{q_0, Z_0})$

xiii) $(q_0, Z_0, q_0) \rightarrow 1(q_0, Z_0, q_0)$

xiv) $(q_0, Z_0, q_1) \rightarrow 1\ (q_0, Z_0, q_1)$

If we assume following state names for above generated 1 to 7 production rules then

		$S \rightarrow A \mid B$
		$A \rightarrow 1\ CA \mid 1\ CE$
$(q_0, Z_0, q_0) = A$		$B \rightarrow 1\ CB \mid 1\ DF$
$(q_0, Z_0, q_1) = B$		$C \rightarrow 1\ CC \mid 1\ DG$
$(q_0, X, q_0) = C$	$\xrightarrow[\text{CFG rules}]{}$	$D \rightarrow 1\ CD \mid 1\ DH$
$(q_0, X, q_1) = D$		$C \rightarrow 0C$
$(q_1, Z_0, q_0) = E$		$D \rightarrow 0D$
$(q_1, Z_0, q_1) = F$		$A \rightarrow \varepsilon$
$(q_1, X, q_0) = G$		$H \rightarrow 1$
$(q_1, X, q_1) = H$		$A \rightarrow 1\ A$
		$B \rightarrow 1\ B$

The production rules for E, F and G are not defined. Hence we will eliminate the production rules on RHS containing the non terminals E, F and G. The resultant CFG rules will be,

$$S \rightarrow A \mid B$$

$$A \rightarrow 1\,CA \mid 1A \mid \varepsilon$$

$$B \rightarrow 1\,CB \mid 1\,B$$

$$C \rightarrow 1\,CC \mid 0\,C$$

$$D \rightarrow 1\,CD \mid 1\,DH \mid 0D$$

$$H \rightarrow 1$$

The rules for B, C and D does not lead to any terminal symbols (i.e. the rules generate continuous non terminals). Hence we will eliminate them. Finally we will get -

$$S \rightarrow A \Rightarrow S \rightarrow 1\,S \mid \varepsilon$$

$$A \rightarrow 1\,A \mid \varepsilon$$

Example 4.7.3 *Find the grammar G equivalent to the following PDA.*

$M = (\{q_0, q_1\}, \{0,1\}, \{X, Z_0\}, q_0, Z_0, d, f)$ and δ is

i) $\delta(q_0, 0, Z_0) = (q_0, XZ_0)$ ii) $\delta(q_0, 0, X) = (q_0, XX)$

iii) $\delta(q_0, 1, X) = (q_1, \Delta)$ iv) $\delta(q_1, 1, X) = (q_1, \Delta)$

v) $\delta(q_1, \Delta, X) = (q_1, \Delta)$ vi) $\delta(q_1, \Delta, Z_0) = (q_1, \Delta)$

SPPU : Dec.-14, End Sem, Marks 9

Solution : The corresponding grammar G can be constructed as follows,

1) The production rule for start symbol S is

$$S \rightarrow (q_0, Z_0, q_0)$$

$$S \rightarrow (q_0, Z_0, q_1)$$

2) Consider the rule $\delta(\mathbf{q_0, 0, Z_0}) = (\mathbf{q_0, X\,Z_0})$

i) $(q_0, Z_0, q_0) \rightarrow 0\,(q_0, X, q_0)\,(q_0, Z_0, q_0)$

ii) $(q_0, Z_0, q_0) \rightarrow 0\,(q_0, X, q_1)\,(q_1, Z_0, q_0)$

iii) $(q_0, Z_0, q_1) \rightarrow 0\,(q_0, X, q_0)\,(q_0, Z_0, q_1)$

iv) $(q_0, Z_0, q_1) \rightarrow 0\,(q_0, X, q_1)\,(q_1, Z_0, q_1)$

3) Consider the rule $\delta(\mathbf{q_0, 0, X}) = (\mathbf{q_0, X\,X})$

v) $(q_0, X, q_0) \rightarrow 0\,(q_0, X, q_0)\,(q_0, X, q_0)$

vi) $(q_0, X, q_0) \rightarrow 0\,(q_0, X, q_1)\,(q_1, X, q_0)$

vii) $(q_0, X, q_1) \rightarrow 0\,(q_0, X, q_0)\,(q_0, X, q_1)$

viii) $(q_0, X, q_1) \rightarrow 0\,(q_0, X, q_1)\,(q_1, X, q_1)$

4) Consider the rule $\delta\,(\mathbf{q_0, 1, X}) = (\mathbf{q_1}, \Delta)$

ix) $(q_0, X, q_1) \rightarrow 1$

5) Consider the rule $\delta\,(\mathbf{q_1, 1, X}) = (\mathbf{q_1}, \Delta)$

x) $(q_1, X, q_1) \rightarrow 1$

6) Consider the rule $\delta\,(\mathbf{q_1}, \Delta, \mathbf{X}) = (\mathbf{q_1}, \Delta)$

xi) $(q_1, X, q_1) \rightarrow \varepsilon$

7) Consider the rule $\delta\,(\mathbf{q_1}, \Delta, \mathbf{Z_0}) = (\mathbf{q_1}, \Delta)$

xii) $(q_1, Z_0, q_1) \rightarrow \varepsilon$

If we assume

$(q_1, Z_0, q_0) = A$		$S \rightarrow A \mid B$
$(q_0, Z_0, q_1) = B$		$A \rightarrow 0CA \mid 0DE$
$(q_0, X, q_0) = C$		$B \rightarrow 0CB \mid 0DG$
$(q_0, X, q_1) = D$	$\xrightarrow[\text{CFG will be}]{\text{The}}$	$C \rightarrow 0CC \mid 0DF$
$(q_1, Z_0, q_0) = E$		$D \rightarrow 0CD \mid 0DH$
$(q_1, X, q_0) = F$		$D \rightarrow 1$
$(q_1, Z_0, q_1) = G$		$H \rightarrow 1$
$(q_1, X, q_1) = H$		$H \rightarrow \varepsilon$
		$G \rightarrow \varepsilon$

There is no rule for E and F. Hence

We will eliminate RHS rules containing E and F. Hence the final production rules are,

$$S \rightarrow A \mid B$$

$$A \rightarrow 0CA$$

$$B \rightarrow 0CB \mid 0DG$$

$$C \rightarrow 0CC$$

$$D \rightarrow 0CD \mid 0DH \mid 1$$

$$H \rightarrow 1 \mid \varepsilon$$

$$G \rightarrow \varepsilon$$

The rule for A, C generate infinite sequence of non terminals. Hence we will eliminate these non terminals. Finally we will get simplified rules as,

$$S \rightarrow B$$

$$B \rightarrow 0DG$$

$$D \rightarrow 0DH \mid 1$$

$$H \rightarrow 1 \mid \varepsilon$$

$$G \rightarrow \varepsilon$$

Example 4.7.4 *Construct a PDA accepting the language $L = \{a^n b^m a^n \mid n, m >= 1\}$ by null store. Then convert this PDA to CFG.*

SPPU : Dec.-11, Marks 8+10, Dec.-19, End Sem, Marks 6

Solution : The instantaneous description for above problem is,

$$\delta(q_0, a, Z_0) = \delta(q_0, a, Z_0)$$

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

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

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

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

$$\delta(q_1, \varepsilon, Z_0) = \delta(q_2, \varepsilon)$$

The conversion of above PDA to CFG will be -

The corresponding grammar can be constructed as follows.

Let $G = (N, T, P, S)$ where

$N = \{S, [q_0, a, q_0], [q_0, Z_0, q_0], [q_0, a, q_1], [q_0, Z_0, q_1]$
$[q_1, a, q_1], [q_1, Z_0, q_1], [q_1, a, q_0]$ and $[q_1, Z_0, q_0]\}$

$T = \{a, b\}$

The productions P are constructed as follows.

The s-productions and

1) $S \rightarrow [q_0, Z_0, q_0]$
2) $S \rightarrow [q_0, Z_0, q_1]$ are the two transitions from q_0.

3) For δ $[q_0, a, Z_0] = [q_0, aZ_0]$ we get

$[q_0, Z_0, q_0] \rightarrow a\ [q_0, a, q_0]\ [q_0, Z_0, q_0]$

$[q_0, Z_0, q_0] \rightarrow a\ [q_0, a, q_1]\ [q_1, Z_0, q_0]$

$[q_0, Z_0, q_1] \rightarrow a\ [q_0, a, q_0]\ [q_0, Z_0, q_1]$

$[q_0, Z_0, q_1] \rightarrow a\ [q_0, a, q_1]\ [q_1, Z_0, q_1]$

4) Using rule δ $(q_0, a, a) = \{(q_0, aa)\}$ we get,

$[q_0, a, q_0] \rightarrow a\ [q_0, a, q_0]\ [q_0, a, q_0]$

$[q_0, a, q_0] \rightarrow a\ [q_0, a, q_1]\ [q_1, a, q_0]$

$[q_0, a, q_1] \rightarrow a\ [q_0, a, q_0]\ [q_0, a, q_1]$

$[q_0, a, q_1] \rightarrow a\ [q_0, a, q_1]\ [q_1, a, q_1]$

5) Using δ $(q_0, b, a) = \{(q_1, a)\}$ we get,

$[q_0, a, q_0] \rightarrow b\ [q_1, a, q_0]$

$[q_0, a, q_1] \rightarrow b\ [q_1, a, q_1]$

6) δ $(q_1, b, a) = \{(q_1, a)\}$ we get,

$[q_1, a, q_0] \rightarrow b\ [q_1, a, q_0]$

$[q_1, a, q_1] \rightarrow b\ [q_1, a, q_1]$

7) δ $(q_1, a, a) = \{(q_1, \varepsilon)\}$ gives us

$[q_1, a, q_1] \rightarrow a$

8) δ $(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$ will give

$[q_1, Z_0, q_1] \rightarrow \varepsilon$

Thus the required CFG is constructed.

4.7.2 CFG to PDA

For converting given CFG to PDA, by this method the necessary condition is that the first symbol on R.H.S. production must be a terminal symbol. The rule that can be used to obtain PDA from CFG is,

Rule 1 : For non terminal symbols, add following rule

$$\delta(q, \varepsilon, A) = (q, \alpha)$$

where the production rule is $A \rightarrow \alpha$.

Rule 2 : For each terminal symbols, add following rule -

$$\delta(q, a, a) = (q, \varepsilon) \text{ for every terminal symbol.}$$

Example 4.7.5 *Construct PDA for the given CFG*

$S \rightarrow 0BB$

$B \rightarrow 0S \mid 1S \mid 0.$

Test whether 010^4 is acceptable by this PDA.

Solution : Let PDA $A = \{ \{q\}, \{0,1\}, \{S,B,0,1\}, \delta, q, S, \phi \}$

The production rules δ can be given as -

$$R1 : \delta(q, \varepsilon, S) = \{ (q, 0BB) \}$$

$$R2 : \delta(q, \varepsilon, B) = \{ (q, 0S), (q, 1S), (q, 0) \}$$

$$R3 : \delta(q, 0, 0) = \{ (q, \varepsilon) \}$$

$$R4 : \delta(q, 1, 1) = \{ (q, \varepsilon) \}$$

Testing 010^4 i.e. 010000 against PDA

$$\begin{aligned}
\delta(q,010000,\mathbf{S}) &\vdash \delta(q,010000,0BB) && \because R1 \\
&\vdash \delta(q,10000,BB) && \because R3 \\
&\vdash \delta(q,10000,1SB) && \because R2 \\
&\vdash \delta(q,\ 0000,SB) && \because R4 \\
&\vdash \delta(q,\ 0000,0BBB) && \because R1 \\
&\vdash \delta(q,000,BBB) && \because R3 \\
&\vdash \delta(q,000,0BB) && \because R2 \\
&\vdash \delta(q,00,BB) && \because R3 \\
&\vdash \delta(q,00,0B) && \because R2 \\
&\vdash \delta(q,0,B) && \because R3 \\
&\vdash \delta(q,0,0) && \because R2 \\
&\vdash \delta(q,\varepsilon) && \because R3 \\
&\quad \text{ACCEPT.}
\end{aligned}$$

Thus 010^4 is accepted by the PDA.

Example 4.7.6 *Convert the grammar*

$S \rightarrow 0\ S\ 1 \mid A$

$A \rightarrow 1\ A\ 0 \mid S \mid \varepsilon$

to the PDA that accepts the same language by empty stack.

SPPU : Dec.-05, Marks 8, Dec.-18, End Sem, Marks 6

Solution : The PDA can be

$$A = \{ \{q\}, \{0,1\}, \{S,A,0,1\}, \delta, q, S, \phi \}$$

The δ can be,

$$R1 : \delta(q, \varepsilon, S) = \{(q, 0S1), (q, A)\}$$

$$R2 : \delta(q, \varepsilon, A) = \{(q, 1A0), (q, S), (q, \varepsilon)\}$$

$$R3 : \delta(q, 0, 0) = \{(q, \varepsilon)\}$$

$$R4 : \delta(q, 1, 1) = \{(q, \varepsilon)\}$$

Example 4.7.7 *Design a PDA for the following grammar :*

$S \rightarrow aSb \mid bSa \mid SS \mid \wedge$

Is the resultant PDA deterministic or non-deterministic ? Justify your answer.

SPPU : May-14, Marks 6

Solution : The PDA can be,

$$A = \{\{q\}, \{a, b\}, \{S, a, b\}, \delta, q, S, \phi\}$$

The δ can be -

$$R1 : \delta(q, \varepsilon, S) = \{(q, aSb), (q, bSa), (q, SS), (q, \varepsilon)\}$$

$$R2 : \delta(q, a, a) = \{(q, \varepsilon)\}$$

$$R3 : \delta(q, b, b) = \{q, \varepsilon)\}$$

The generated PDA is nondeterministic PDA because there can be more than one different moves for the same input symbol and from same current state.

Example 4.7.8 *Construct PDA that accepts language generated by following CFG :*

$S \rightarrow SS \mid (S) \mid ( )$

SPPU : Dec.-17, End Sem, Marks 6

Solution : The δ for given CFG for constructing PDA is as follows,

$$\delta(q, \varepsilon, S) = \{(q, SS), (q, (S)), (q, ( ))\}$$

$$\delta(q, (, ()) = \{(q, \varepsilon)\}$$

$$\delta(q,),)) = \{(q, \varepsilon)\}$$

Example 4.7.9 *Construct a PDA that recognizes the language accepted by following DFA.*

SPPU : May-18, End Sem, Marks 6

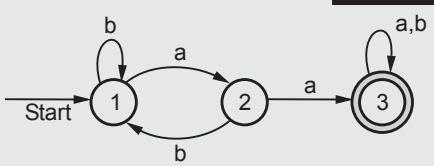


Fig. 4.7.1

Solution : • We will write r.e. for given

DFA r.e. = $b^* a(ba)^* a(a+b)^*$

- Now we will write production rules for obtained r.e.

$$S \rightarrow XaYaZ$$
$$X \rightarrow bX \mid \varepsilon$$
$$Y \rightarrow baY \mid \varepsilon$$
$$Z \rightarrow aZ \mid bZ \mid \varepsilon$$

- The PDA can be,

$$\delta(q, \varepsilon, S) = \{(q, XaYaZ)\}$$
$$\delta(q, \varepsilon, X) = \{(q, bX), (q, \varepsilon)\}$$
$$\delta(q, \varepsilon, Y) = \{(q, baY), (q, \varepsilon)\}$$
$$\delta(q, \varepsilon, Z) = \{(q, aZ), (q, bZ), (q, \varepsilon)\}$$

4.8 Deterministic CFLs

- The Deterministic Context-Free Grammars (DCFGs) are a proper subset of the context-free grammars.
- They are the subset of context-free grammars that can be derived from deterministic pushdown automata, and they generate the deterministic context-free languages.
- DCFGs are always unambiguous.

Theorem 1 : Prove that if L is a regular language, then L = L (P) for some DPDA P.

Proof :

- Fig. 4.8.1 shows the class of various languages.

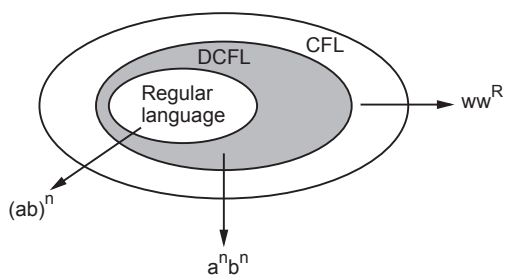


Fig. 4.8.1 Class of various languages

- A language L is a deterministic context free language (**DCFL**) if it is accepted by **DPDA**. Any language L is a subset of DCFL. For example $L = a^*b^*$ is a regular language and $L = a^nb^n$ is a DCFL, and it is accepted by DPDA. And therefore $L = a^*b^*$ is also acceptable by DPDA. Every regular language is accepted by DFA. And every DFA is a DPDA without stack. Hence it is proved that for a regular language L = L(P) there exists some DPDA P.

Theorem 2 : If L is a CFL and R is a regular language then $L \cap R$ is a CFL.

Proof :

- For proving this theorem we will first create a PDA such that it is made up of two machines PDA and a FA. These two models will run parallel by **ANDing** together. The DFA will accept a language L(R) and PDA will accept language L_{CFL}. Hence we can define these two models as -

$$M = (Q_{FA}, \Sigma, \delta_{FA}, q_{FA}, F_{FA}) \quad \text{be FA}$$

$$P = (Q_{PDA}, \Sigma, \Gamma, \delta_{PDA}, q_{PDA}, Z_0, F_{PDA}) \quad \text{be PDA}$$

Then the resultant PDA can be,

$$P_{New} = (Q_{PDA} \times Q_{FA}, \Sigma, \Gamma, \delta_{PM}, (q_{PDA}, q_{FA}), Z_0, F_{FA} \times F_{PDA})$$

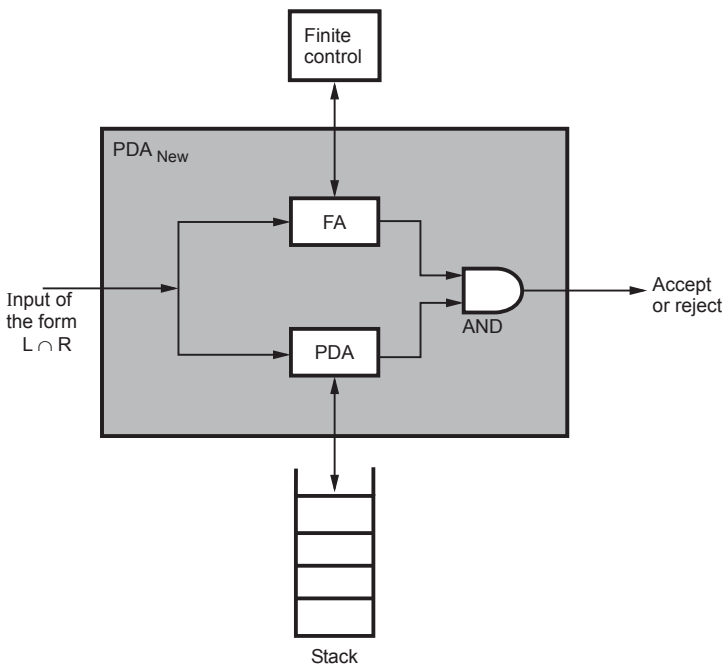


Fig. 4.8.2 L ∩ R is aceeptable by PDA new

where δ is a mapping function it can be defined as -

$\delta((q_1, q_2), a, X)$ is defined as the set of all pairs $((q_3, q_4), Y)$ such that

1. $q_4 = \delta'_{FA}(q_2, a)$ and

2. (q_3, Y) is in $\delta_{PDA}(q_1, a, X)$

Thus for each move of PDA P, we will get the same move in PDA_{New}.

Thus the moves made by the new PDA will be,

$$((q_{PDA}, q_{FA}), W, Z_0) \overset{*}{\underset{P_{New}}{\vdash}} ((q_1, q_2), \varepsilon, Y)$$

Where q_2 will act as q_f. This leads to an accepting configuration.

4.9 Fill in the Blanks

Q.1 ___ is a graphical representation used for modeling the context free grammar.

Q.2 ___ and ___ are types of halting states in push down automata.

Q.3 A pushdown automata can be defined as : (Q, Σ, G, q0, z0, A, d). The z0 represents ____.

Q.4 For every CFL, G, there exists a PDA M such that L(G) = L(M) and vice versa. (True/False)

Q.5 The transition in push down automata depends upon ____ data structure.

Q.6 The PDA accepts a string when ___ is empty.

4.10 Multiple Choice Questions

Q.1 The idea of automation with stack is executed in ___.

a finite automata
b push down automata
c none of these

Q.2 PDA is more powerful than _____.

a turing machine
b finite automata
c both a and b

Q.3 The instantaneous description is PDA shows _____ .

a present state
b stack symbol
c string to be processed
d all of these

Q.4 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

Q.5 Push down machine represents _____ .

a Type-0 grammar
b Type-1 grammar
c Type-2 grammar
d Type-3 grammar

Q.6 Derivation tree is ____.

a parse tree
b syntax tree
c both a and b
d none

Q.7 A PDA behaves like FA when the number of auxiliary memory is _____.

a 0
b 1
c 3
d none

Answer Keys for Fill in the Blanks :

Q.1	Push down automata	**Q.2**	Accept, reject
Q.3	stack symbol	**Q.4**	True
Q.5	stack	**Q.6**	stack

Answer Keys for Multiple Choice Questions :

Q.1	b	**Q.2**	b	**Q.3**	d	**Q.4**	a
Q.5	c	**Q.6**	a	**Q.7**	a		

❑❑❑

Notes

UNIT - V

5 Turing Machines (TM)

Syllabus

Turing Machine Model, Formal definition of Turing Machines, Language Acceptability by Turing Machines, Design of TM, Description of TM, Techniques for TM Construction, Computing function with Turing Machine, Variants of Turing Machines, Halting Problem of TM, Halting vs Looping, A Turing-unrecognizable language, Reducibility, Recursion Theorem. The Model of Linear Bounded Automata.

Contents

5.1 Turing Machine Model

5.2 Design of TM . . . . . **Dec.-13,** . . . . . Marks 3

5.3 Instantaneous Description(ID) of TM . . . . . **Dec.-17, May-18,** . . . . . Marks 4

5.4 Techniques for TM Construction . . . . . **May-19,** . . . . . Marks 8

5.5 Language Acceptability by Turing Machines

. . . . . **Dec.-06,07,09,11,13,14,17,18,19,**

. . . . . **May-06,11,13,14,16,18,19,** Marks 10

5.6 Computing Function with Turing Machine **Dec.-05,07,08,09,10,15,17,18,19,**

. . . . . **May-06,07,10,12,14,15,16,** Marks 12

5.7 Variants of Turing Machines. . . . . **Dec.-17,** . . . . . Marks 4

5.8 Halting Problem of TM . . . . . **May-16,19, Dec.-17,19,** . . . Marks 4

5.9 Halting Vs Looping

5.10 A Turing - unrecognizable Language

5.11 Reducibility

5.12 Recursion Theorem

5.13 The Model of Linear Bounded Automata

5.14 Fill in the Blanks

5.15 Multiple Choice Questions

5.1 Turing Machine Model

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.

5.2 Design of TM

SPPU : Dec.-13, Marks 3

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.

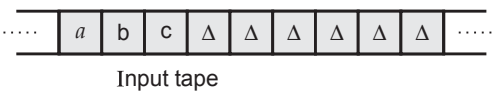


Fig. 5.2.1 Input tape

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.

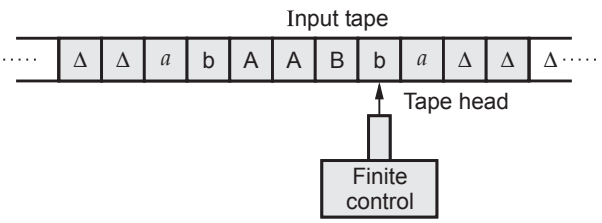


Fig. 5.2.2 Turing machine

Review Question

1. *Give formal definition of move relation of turing machine.* **SPPU : Dec.-13, Marks 3**

5.3 Instantaneous Description(ID) of TM

SPPU : Dec.-17, May-18, Marks 4

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.

Review Question

1. *Explain the representation of TM.* **SPPU : Dec.-17, May-18, End Sem, Marks 4**

5.4 Techniques for TM Construction

SPPU : May-19, Marks 8

Construction of turing machine is a process of writing out the complete set of states and next move function. This is a totally conceptual phenomenon. The turing machine can be designed with the help of some conceptual tools. Let us discuss some of these tools.

5.4.1 Storage in Finite Control

As shown in Fig. 5.4.1, the model of turing machine has a finite control. This finite control can be used to hold some amount of information. The finite automata stores the information in pair of elements such as the current state and the current symbol pointed by the tape head. This is just a conceptual arrangement.

For example - The function δ can be written as follows.

$$\delta([q_0, 0], 1) \rightarrow ([q_1, 1], \Delta, R)$$

This means that if finite control shows the initial state is q_0 and stores the current symbol 0 if it reads the symbol 1 then the machine goes to next state q_1 replace that 1 by Δ and moves to right. This helps in building the transition graph of the language.

Example 5.4.1 *Construct a turing machine M for* $\Sigma = \{a, b\}$ *which will convert lower case letters to upper case.*

Solution : The turing machine is designed such that we will get the equivalent upper case string to the input. The q_0 is a initial state and q_3 is final state. The δ function is written along each edge. For the sake of understanding consider the transition between q_0 and q_1, if the finite control reads on the tape A will be printed there and the move will be in right direction. After converting the given string to upper case we reach to the final state q_3. The q_3 is a halt state.

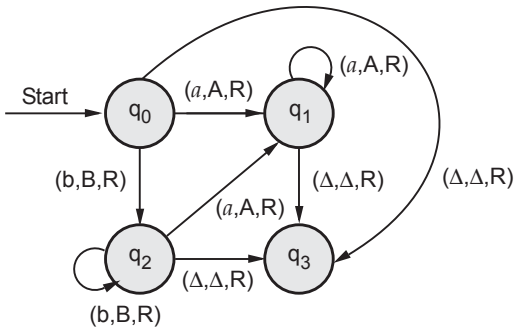


Fig. 5.4.1

5.4.2 Multiple Tracks

If the input tape is divided into multiple tracks then the input tape will be as follows -

For example :

As shown in Fig. 5.4.2 the input tape has multiple tracks on the first track. The input which is placed is surrounded by # and $. The unary number equivalent to 5 is placed on the input tape, on the first track. On the second track unary 2 is placed. If we construct a TM which subtracts 2 from 5 we get the answer on the third track and that is 3, in unary form. Thus this TM is for subtracting two unary numbers with the help of multiple tracks.

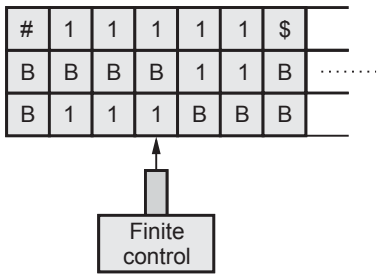


Fig. 5.4.2 Multiple tracks

5.4.3 Checking Off Symbols

Checking off symbols is an effective way of recognizing the language by TM. The symbols are to be placed on the input tape. The symbol which is read is marked by any special character. The tape head can be moved to the right or left. Let us take some example and we will see how to build turing machine by checking of symbols.

Example 5.4.2 *Construct a turing machine* $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ *which recognizes the language* $L = \left\{ wcw \mid w \in (a+b)^+ \right\}$.

Solution : In this language the input set is Σ = {*a*, b}. The string which when will be placed on the input tape it will have two distinct parts separated by letter c, such as

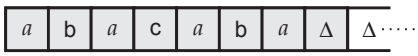


Fig. 5.4.3

In the checking off symbols, each symbol is marked by special character. The simple logic in construction of this TM will be we mark the first letter and then move to the right till we not get c, the first letter after c will be compared with the marked letter. If it is the same as which we have marked then mark this symbol otherwise goto reject state. It can be shown as below,

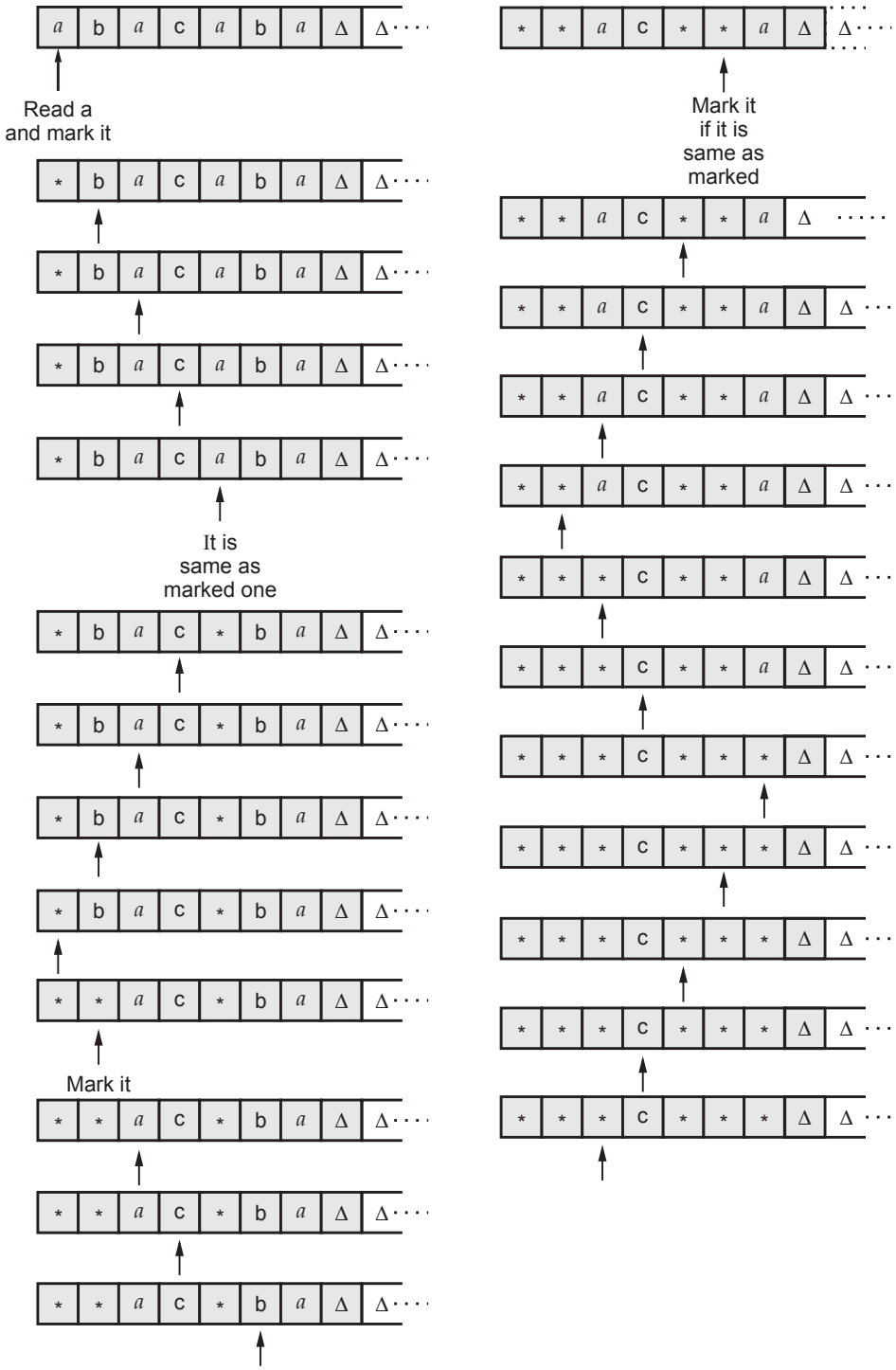


Fig. 5.4.4

Now machine goes to accept state. Thus in this TM we are scanning each symbol and trying to recognize the string.

5.4.4 Subroutine

In the high level languages use of subroutines built the modularity in the program development process. The same type of concept can be introduced in construction of TM. We can write the subroutines as a turing machine. Let us see how it works with the help of some example.

Example 5.4.3 *Construct a TM for the subroutine f (a, b) = a ∗ b where a and b are unary numbers.* **SPPU : May-19, End Sem, Marks 8**

Solution : The unary number are represented by 1's. That if a = 2 and b = 3 then for this subroutine we will perform a copy operation. That is marking first 1 of a and copying it after Δ at b number of times, similarly marking second 1 of a and copying it at the rightmost end (i.e. after previously copied 1's) for b number of times. This makes the multiplication function complete.

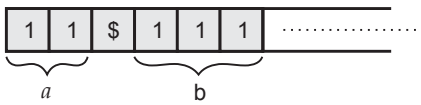


Fig. 5.4.5

				1 ↑	1	$	1	1	1	Δ	
				X	1 ↑	$	1	1	1	Δ	Moving to right upto $
				X	1	$	1 ↑	1	1	Δ	Copy this 1 after Δ by marking it as Y.
		X	1	$	Y	1	1	Δ	1 ↑	Δ	Now move to left upto Y
		X	1	$	Y ↑	1	1	Δ	1	Δ	
		X	1	$	Y	1 ↑	1	Δ	1	Δ	Mark this 1 as Y and copy this 1 at the rightmost end.
	X	1	$	Y	Y	1	Δ	1	1 ↑	Δ	Now move to left upto Y
	X	1	$	Y	Y ↑	1	Δ	1	1	Δ	
	X	1	$	Y	Y	1 ↑	Δ	1	1	Δ	Mark this 1 as Y and copy this 1 at the rightmost end.
X	1	$	Y	Y	Y	Δ	1	1	1	Δ	Now move to left upto Y
	X	1	$	Y	Y	Y ↑	Δ	1	1	1	

The head is pointing to Y next Y is Δ that means all the 1's are over we will convert these Y to 1's and repeat the same procedure for the 1 which is left to $.

		X	1	$	1	1	1	Δ	1	1	1	
		X	X ↑	$	1	1	1	Δ	1	1	1	Convert this 1 to X and move to right upto after the $ symbol.
		X	X	$	1 ↑	1	1	Δ	1	1	1	Mark it Y and copy this 1 to rightmost end
	X	X	$	Y	1	1	Δ	1	1	1	1 ↑	Move to left upto Y
	X	X	$	Y	1 ↑	1	Δ	1	1	1	1	Convert this 1 to Y and move to right, copy this 1 to rightmost end
X	X	$	Y	Y	1	Δ	1	1	1	1	1 ↑	Move to left upto Y

The above steps will be repeated and all the 1's between $ and Δ are copied to the rightmost end. This will be like this

X X $ Y Y Y Δ 1 1 1 1 1 1

Now we will move to left by converting all Y's and X's to 1's. Finally input tape will have

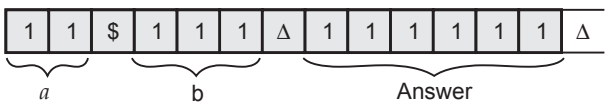


Fig. 5.4.6

We put these actions together and draw the transition graph and model out the turing machine.

The HALT state is a final state in the above TM.

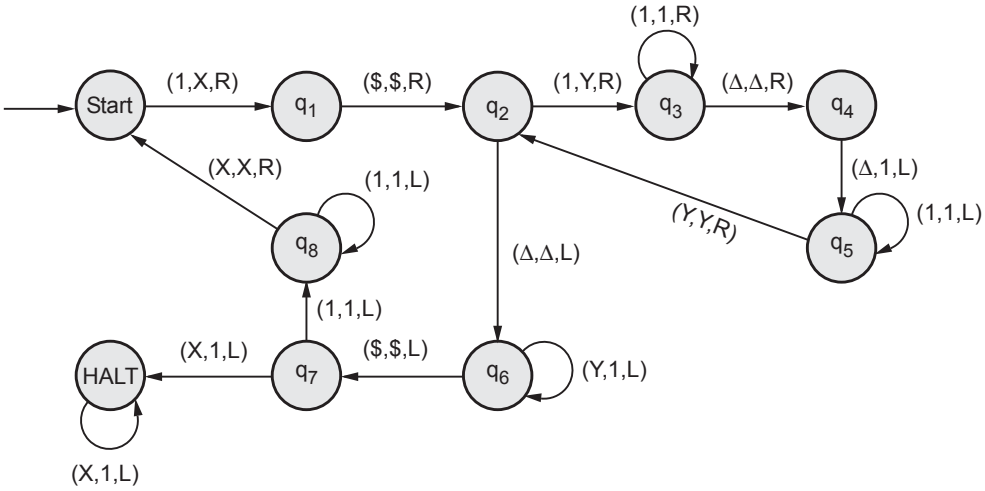


Fig. 5.4.7 Turing machine for multiplication subroutine

5.5 Language Acceptability by Turing Machines

SPPU : Dec.-06,07,09,11,13,14,17,18,19, May-06,11,13,14,16,18,19, Marks 10

The Turing machine accepts all the languages eventhough 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 5.5.1 *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.

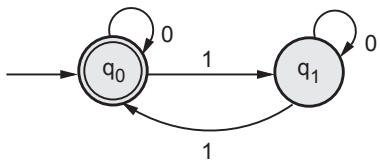


Fig. 5.5.1 FSM for even number of 1's

The same idea can be used to draw TM. We always assume that at the end of input Δ is placed.

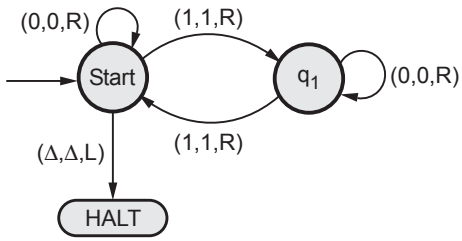


Fig. 5.5.2

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	1	1	0	1	0	1	Δ
head	↑						
state	q_1						
	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 5.5.2 *Design a turing machine which recognizes the input language having a substring as 101 and replaces every occurrence of 101 by 110.*

Solution : Replacement of any symbol by some another one means after reading of that specific symbol we should print the replacement symbol. In this case 101 has to be replaced by 110. The TM has to be constructed considering 101 as a substring and leaving 110 substring after complete scan of the input.

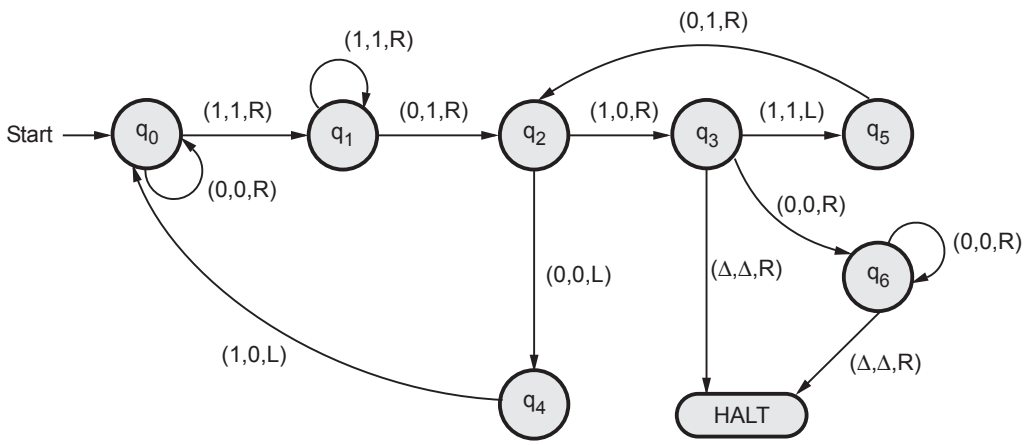


Fig. 5.5.3

Let us simulate it for the input.

1 0 0 1 0 1 0 Δ ↑	read 1, print 1, goto q_1
1 0 0 1 0 1 0 Δ ↑	goto R and next state is q_0
1 0 0 1 0 1 0 Δ ↑	goto R, next state is q_0
1 0 0 1 0 1 0 Δ ↑	goto R, next state q_1
1 0 0 1 1 1 0 Δ ↑	print 1, goto R and next state is q_2
1 0 0 1 1 1 0 Δ ↑	print 0, goto R and next state is q_3
1 0 0 1 1 0 0 Δ ↑	print 0, goto R and next state is q_6
1 0 0 1 1 0 0 Δ ↑	since we read current symbol as Δ we goto accept state i.e. HALT.

Example 5.5.3 *Construct a TM for the language of even number of 1's and even number of 0's over* $\Sigma = \{0, 1\}$. **SPPU : Dec.-17, End sem, Marks 8**

Solution : For this type of problem we have drawn FSM, as follows -

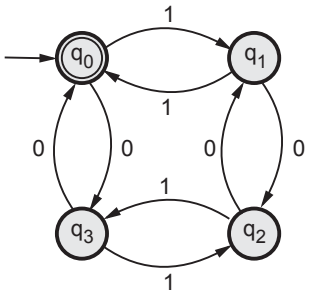


Fig. 5.5.4

And now it will be very much easy for us to convert this FSM to TM.

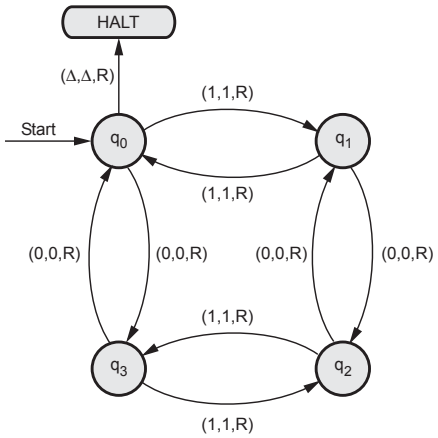


Fig. 5.5.5

Note that for any FSM we can construct turing machine. Thus turing machine more powerful than TM.

Example 5.5.4 *Construct TM for the language* $L = \{a^n b^n\}$ *where* $n \geq 1$.

SPPU : May-13, May-18, End Sem, Marks 8

Solution : 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 unequality 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 Δ ↑	Convert it A and move right in search of b.
A *a* b b Δ ↑	Skip it, move ahead
A *a* b b Δ ↑	Convert it to B and move left till A
A *a* B b Δ ↑	Move left
A *a* B b Δ ↑	Move right
A *a* B b Δ ↑	Convert *a* to A and move right in search of b
A A B b Δ ↑	Move right
A A B b Δ ↑	Convert b to B and move left
A A B B Δ ↑	Move left
A A B B Δ ↑	Now immediately before B is A that means all the *a*'s are marked by A. So we will move right to ensure that no b is present
A A B B Δ ↑	Move right
A A B B Δ ↑	Move right
A A B B Δ ↑	HALT

Thus equality between *a*'s and b's can be checked by moving tape head to and fro. Let us try to design a TM using above logic.

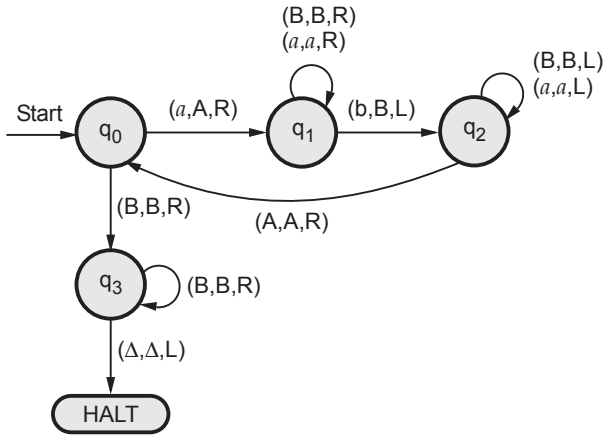


Fig. 5.5.6

Note that for the invalid inputs the machine does not go to HALT state.

For example

$\underset{\uparrow}{a}\ a\ b\ b\ b\ \Delta$	Convert to A, move right and next state = q_1
$A\ \underset{\uparrow}{a}\ b\ b\ b\ \Delta$	Move right, keeping a as it is, state = q_1
$A\ \underset{\uparrow}{a}\ b\ b\ b\ \Delta$	Convert to B, move to left
$A\ \underset{\uparrow}{a}\ B\ b\ b\ \Delta$	Move to left, state = q_2
$\underset{\uparrow}{A}\ a\ B\ b\ b\ \Delta$	Move to right and state = q_0
$A\ \underset{\uparrow}{a}\ B\ b\ b\ \Delta$	Convert to A, move to right now in state q_1
$A\ A\ \underset{\uparrow}{B}\ b\ b\ \Delta$	Move right keeping B as it is.
$A\ A\ B\ \underset{\uparrow}{b}\ b\ \Delta$	Convert b to B and move left state is q_2
$A\ A\ \underset{\uparrow}{B}\ B\ b\ \Delta$	Go on moving left
$\underset{\uparrow}{A}\ A\ B\ B\ b\ \Delta$	Now state q_0 move right
$A\ A\ \underset{\uparrow}{B}\ B\ b\ \Delta$	Move right skipping all B's
$A\ A\ B\ B\ \underset{\uparrow}{b}\ \Delta$	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 5.5.1 Transition table for example 5.5.4

Thus TM does not lead to HALT state for such a invalid input.

Example 5.5.5 *Construct a TM for* $L = \{a^n b^n c^n \mid n \geq 1\}$.

Solution : This problem cannot be solved even by PDA. We have solved this type of problem by two stack PDA.

The simulation for *a a* b b c c can be shown as below.

Fig. 5.5.7

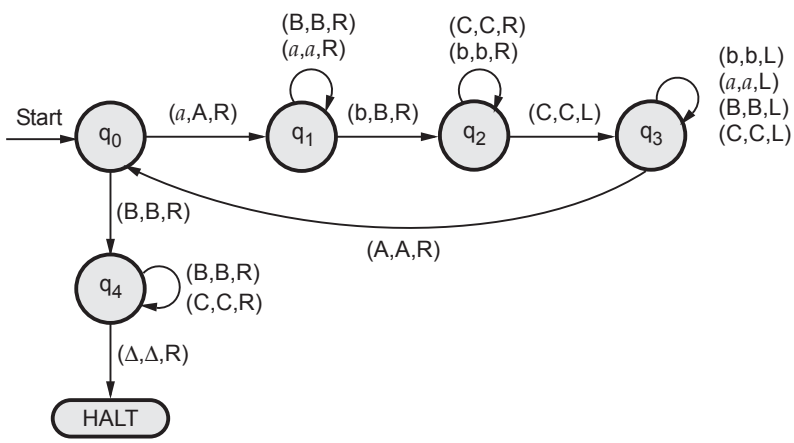


Fig. 5.5.8

a a b b c c Δ ↑	Convert *a* to A, move right the transition is from q_0 to q_1
A *a* b b c c Δ ↑	Move right upto b
A *a* b b c c Δ ↑	Convert b to B, move right, transition from q_1 to q_2
A *a* B b c c Δ ↑	Move right upto c
A *a* B b c c Δ ↑	Convert c to C, move right, transition from q_2 to q_3
A *a* B b C c Δ ↑	Move left upto C
A *a* B b C c Δ ↑	Move left till A, skipping *a*, b, B, C, see state q_3 then move one step right
A *a* B b C c Δ ↑	Convert *a* to A, move right
A *a* B b C c Δ ↑	Move right
A A B b C c Δ ↑	Convert b to B and move right, transition from q_1 to q_2
A A B B C c Δ ↑	Move right
A A B B C c Δ ↑	Convert c to C, move left till A, refer q_3 state
A A B B C C Δ ↑	Move right, keep B as it is refer transition from q_0 to q_4
A A B B C C Δ ↑	Go on moving right by ignoring B, C refer q_4 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 5.5.6 *Construct a TM machine for checking the palindrome of the string of even length.* **SPPU : May-11, Marks 8**

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,

a 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 Δ Δ ↑	If it is a, replaced it by Δ
* b a b b a b Δ ↑	Move to left, upto $*$
* b a b b a b Δ ↑	Move right, read it
* b a b b a b Δ ↑	Convert it to $*$ and move right
* * a b b a b Δ ↑	Move right upto Δ in search of b
* * a b b a b Δ ↑	Move left, if left symbol is b convert it to Δ
* * a b 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 Δ ↑	Move to left till *
* * * b b Δ ↑	Move right
* * * b b Δ ↑	Replace it by *
* * * * b Δ ↑	Move right upto Δ
* * * * b Δ ↑	Move left to see if it is b, if so then replace it by Δ
* * * * Δ ↑	Move left till *
* * * * Δ ↑	Move right and check whether it is Δ, if so
* * * * Δ ↑	Goto HALT state

Let us draw TM for it,

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.

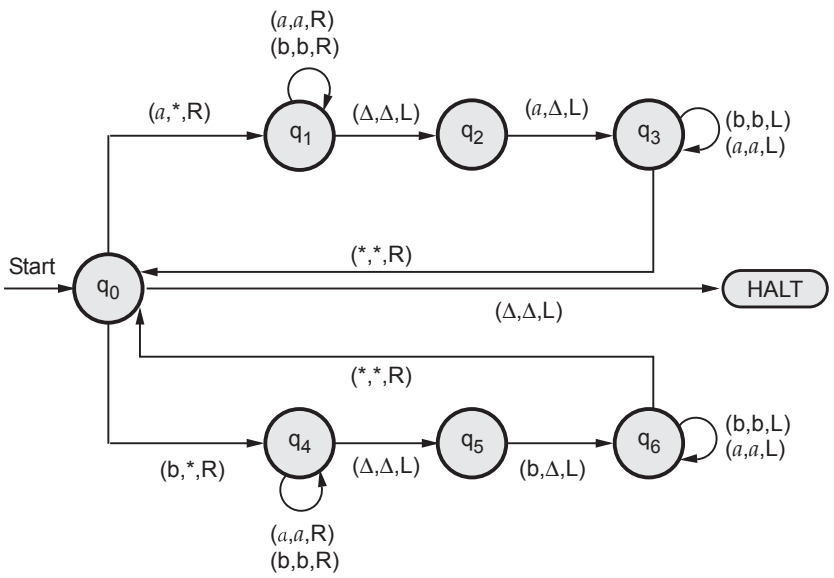


Fig. 5.5.9

Example 5.5.7 *Construct a TM for checking the palindrome of a string of odd palindrome for* $\Sigma = \{0, 1\}$.

SPPU : May-14, Marks 7, Dec.-19, End Sem, Marks 8

Solution : The TM will be very much similar to previous one but with a slight difference.

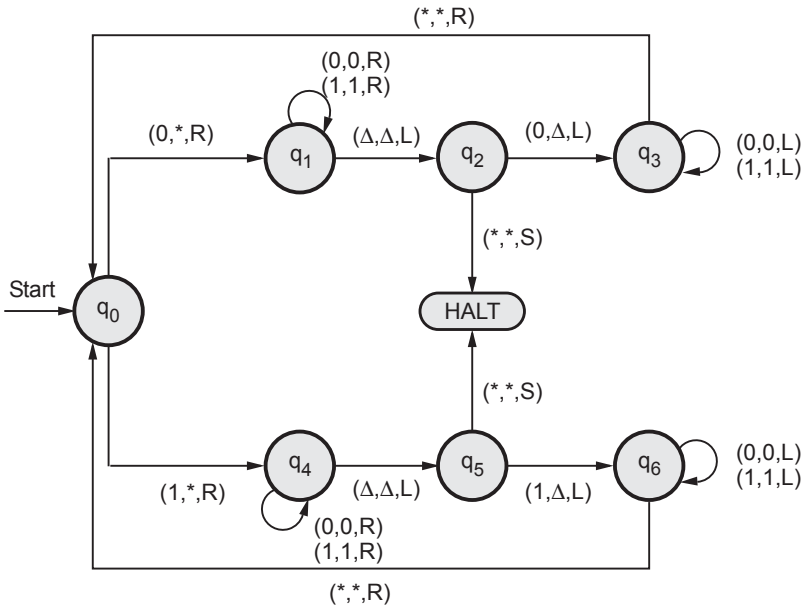


Fig. 5.5.10

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 5.5.8 *Construct TM for concatenation of the two strings of unary numbers. This TM is for a concatenate function.* **SPPU : Dec.-11, Marks 8**

Solution : The TM will be constructed for concatenate two strings of unary numbers. The strings are placed as follows

1	1	1	1	$	1	1	1	Δ

Fig. 5.5.11

The two strings are separated by $ we will replace that $ by next 1. That is we go on forwarding $ ahead.

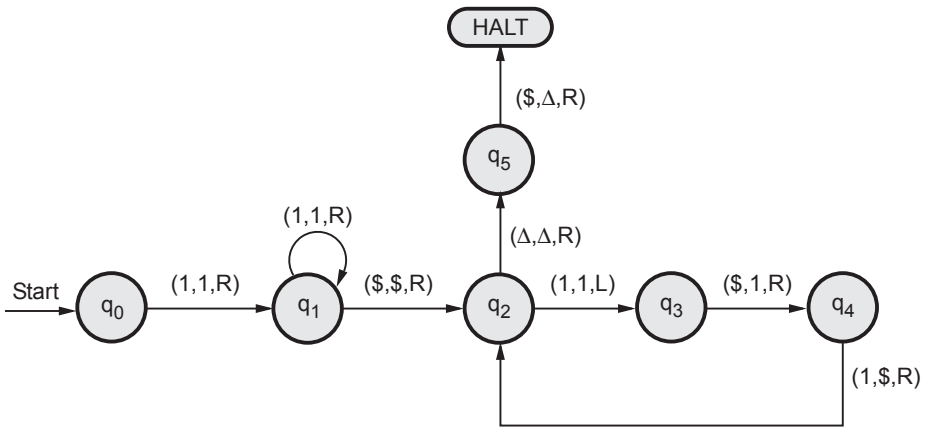


Fig. 5.5.12

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 5.5.9 *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,

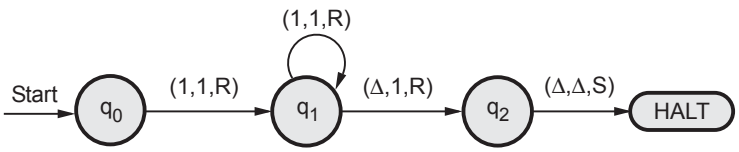


Fig. 5.5.13

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
q_0	$(q_1, 1, R)$	–
q_1	$(q_1, 1, R)$	$(q_2, 1, R)$
q_2	–	(HALT, Δ, S)
HALT	–	–

In the transition table the triple is of (next state, output, direction).

Example 5.5.10 *Construct TM for the addition function for the unary number system.*

SPPU : Dec.-17, 19, End Sem,Marks 6

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

i.e. 11 + 111

= 11111

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

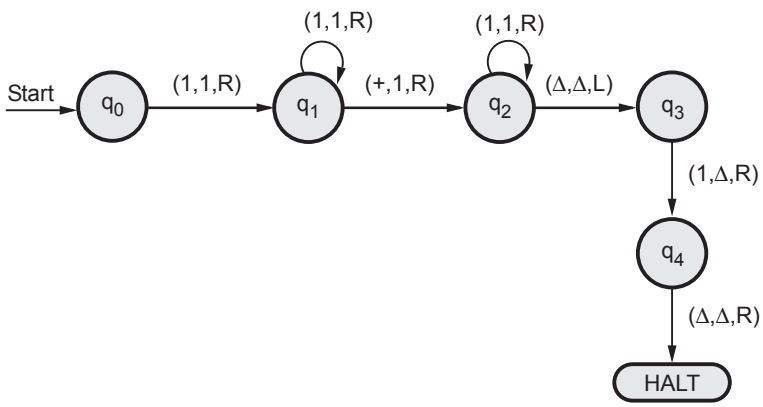


Fig. 5.5.14

Here we are implementing the function of f (a + b) = c. We assume a and b both are nonzero elements.

Example 5.5.11 *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,

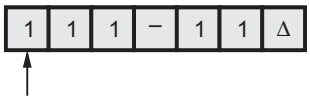


Fig. 5.5.15

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 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
1 1 * – * * Δ ↑	Move left till 1
1 1 * – * * Δ ↑	Convert 1 to ∗ and move right
1 * * – * * Δ ↑	Move right till Δ
1 * * – * * Δ ↑	HALT

Thus we get 1 on the input tape as answer for f (3 – 2).

Let us draw TM for it.

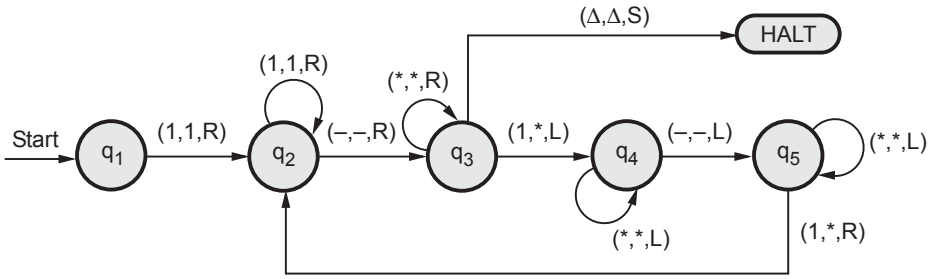


Fig. 5.5.16

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.

Example 5.5.12 *Construct turing machine that recognizes the language*

1) $L = \{0^n 1^m : n, m >= 0\}$

2) $L = \{x \in \{0,1\}* \mid x \text{ ends in } 00\}$ **SPPU : May-06, Marks 10**

Solution :

i) Let, $\{L = 0^n 1^m \mid n, m >= 0\}$

That means any number of 0's and 1's are allowed in a string but all the 0's must appear before all the 1's. Hence the TM can be,

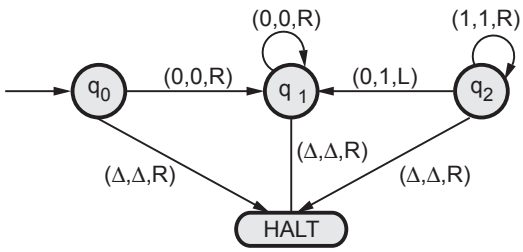


Fig. 5.5.17

ii) Let, $L = \{x \in \{0,1\}* \mid x \text{ end in } 00\}$ be the given language. We will first draw FA for this language.

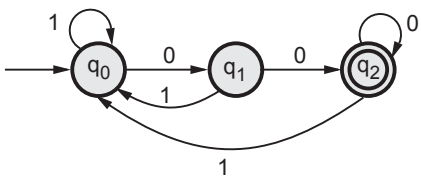


Fig. 5.5.18

The TM can then be as follows,

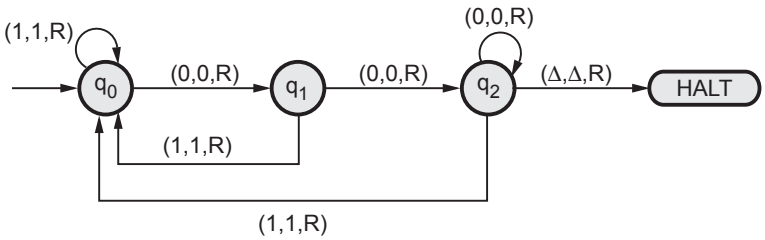


Fig. 5.5.19

Example 5.5.13 *Design a turing machine to replace string 110 by 101 in binary input string.*

SPPU : Dec.-06, Marks 8, Dec.-14, End Sem., Marks 9

Solution : The TM can then be,

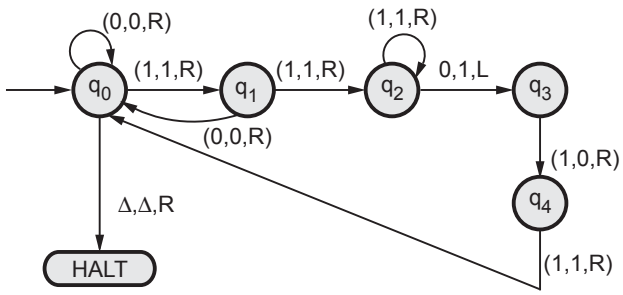


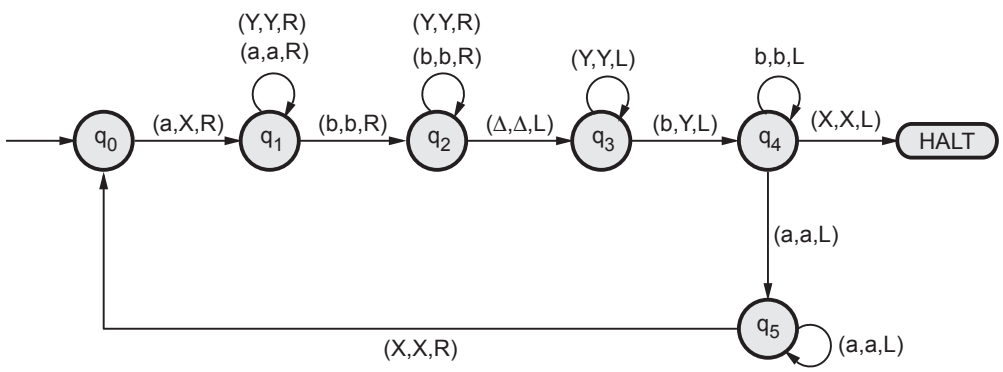
Fig. 5.5.20

Example 5.5.14 *Draw a transitions table for turing machine accepting each of the following languages.*

a) $\{a^i b^j / i < j\}$ *b) The language of balanced strings of parentheses.*

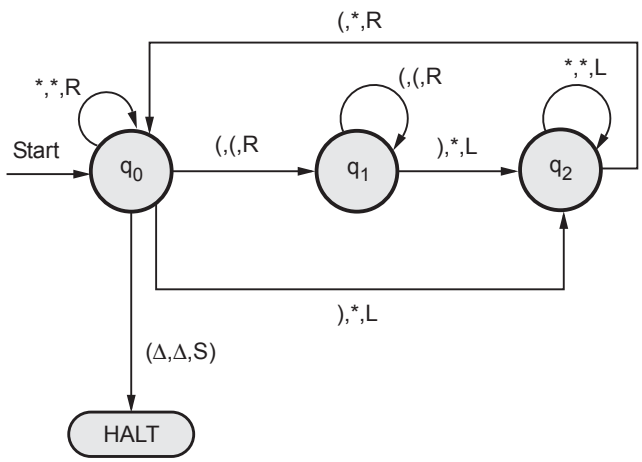
SPPU : Dec.-07, 09, May-09, Marks 8, Dec.-14, End Sem., Marks 9

Solution : a) The required TM can be drawn as follows -



The transition table can be

State \ Input	a	b	X	Y	Δ
q_0	(q_1, X, R)	–	–	–	–
q_1	(q_1, a, R)	(q_2, b, R)	–	–	–
q_2	–	(q_2, b, R)	–	(q_2, Y, R)	(q_3, Δ, L)
q_3	–	(q_4, Y, L)	–	(q_3, Y, L)	–
q_4	(q_5, a, L)	(q_4, b, L)	(HALT, X, L)	–	–
q_5	(q_5, a, L)	–	(q_0, X, R)	–	–



The transition table for this TM will be

State	(	)	*	Δ
q_0	$(q_1, (, R)$	$(q_2, *, L)$	$(q_0, *, R)$	(HALT, Δ, S)
q_1	$(q_1, (, R)$	$(q_2, *, L)$	–	–
q_2	$(q_2, *, R)$	–	$(q_2, *, L)$	–
HALT	–	–	–	–

Example 5.5.15 *Design a turing machine M to recognize the language* $\{1^n 2^n 3^n \mid n \geq 1\}$.

SPPU : Dec.-11, Marks 8; Dec.-13, Marks 6

Solution :

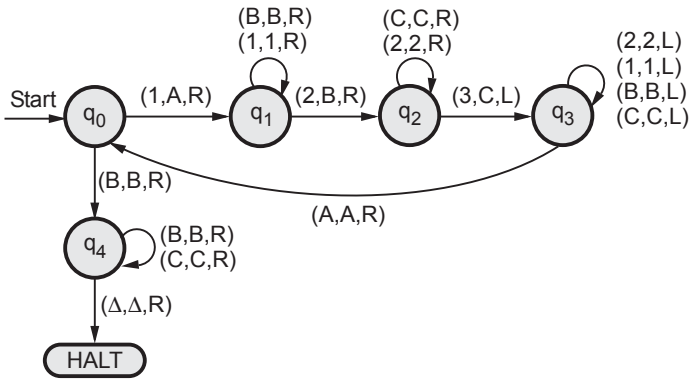


Fig. 5.5.21

Example 5.5.16 *Design a TM that replaces every occurrence of abb by baa.*

SPPU : May-16, Marks 6

Solution :

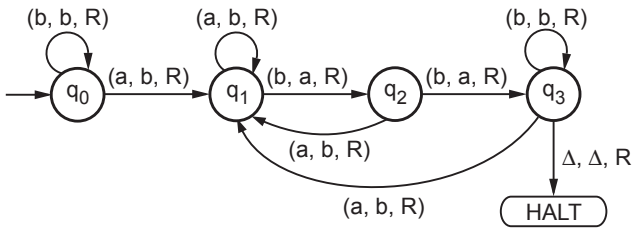


Fig. 5.5.22

Example 5.5.17 *Construct TM for 1's complement of binary number.*

SPPU : May-18, End Sem, Marks 6

Solution :

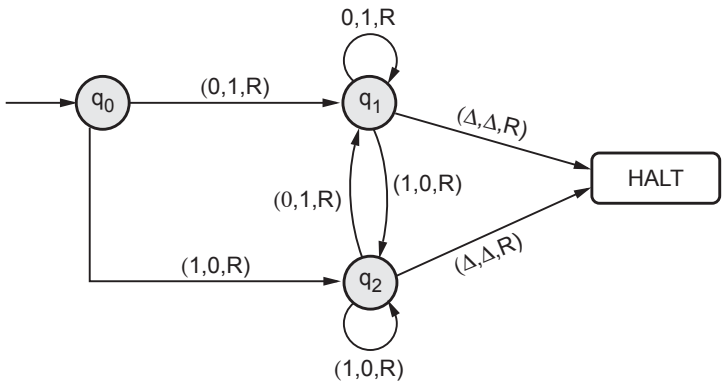


Fig. 5.5.23

Example 5.5.18 *Design a turing machine to accept the language L = {w/w ∈ (0 + 1)*} containing the substring 001.* **SPPU : May-18, End Sem, Marks 8**

Solution :

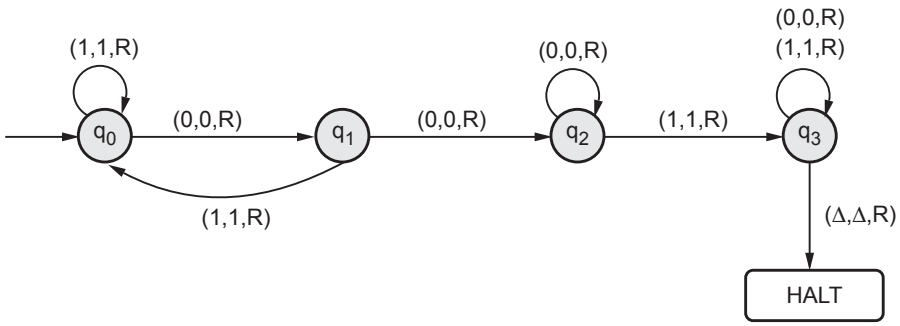
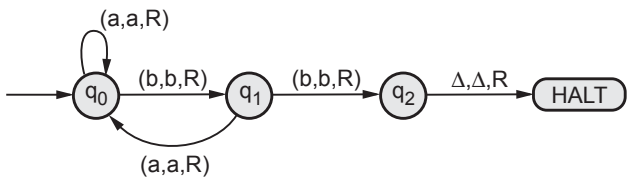


Fig. 5.5.24

Example 5.5.19 *Construct a turing machine for R = (a + b)*bb.*

SPPU : Dec.-18, End Sem, Marks 6

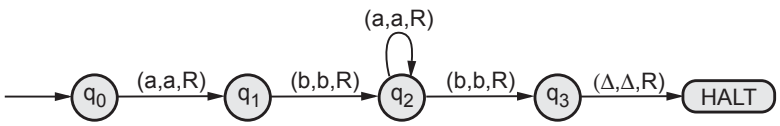
Solution :



Example 5.5.20 *Construct a turing machine for R = (aba*b).*

SPPU : Dec.-18, End Sem, Marks 6

Solution :



Example 5.5.21 *Design TM for the language L = {0^{2n}} over Σ = {0,1}.*

SPPU : May-19, End Sem, Marks 8

Solution : We will design finite automata for

L = {0^{2n}} over {0, 1}

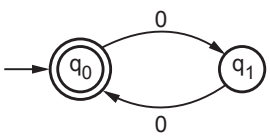


Fig. 5.5.25

From this FA we can design TM as follows -

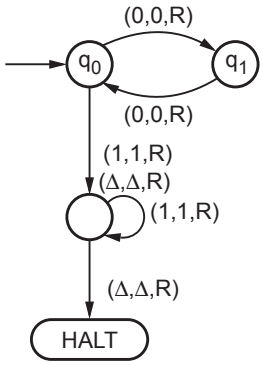


Fig. 5.5.26

Example 5.5.22 *Design TM to accept the set L of all strings formed with 0 and 1 and having substring '000'.* **SPPU : May-19, End Sem, Marks 8**

Solution : The FA can be designed as,

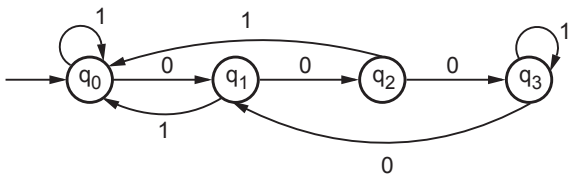


Fig. 5.5.27

From this FA we can design TM as,

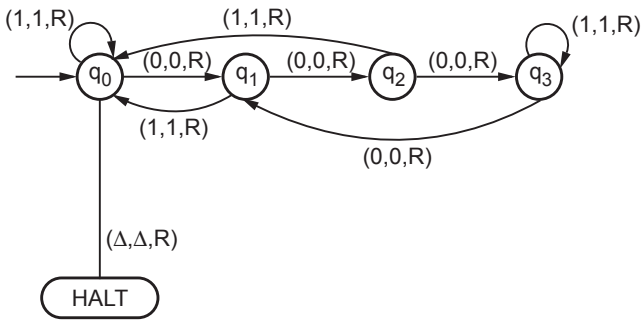


Fig. 5.5.28

Example 5.5.23 *Construct a Turing machine for the language* $L = \{ a^n bc^n \mid n >= 1\}$.

SPPU : Dec.-19, End Sem, Marks 6

Solution :

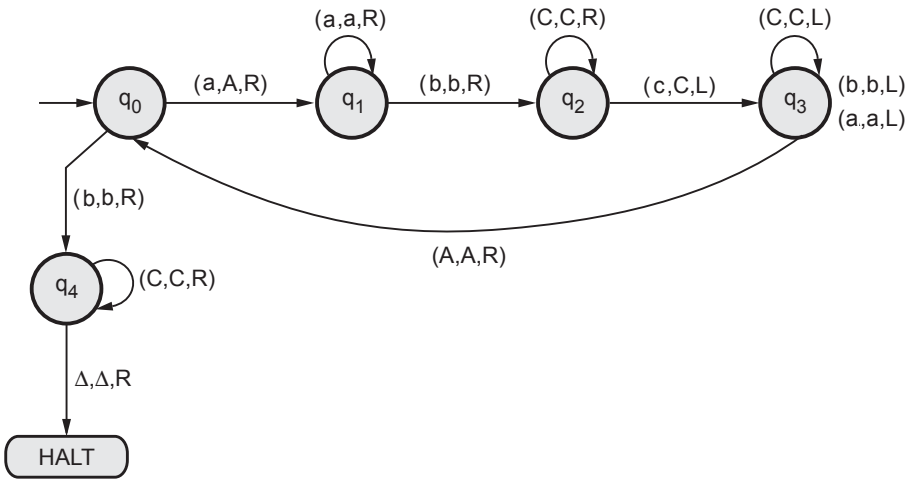


Fig. 5.5.29

5.6 Computing Function with Turing Machine

SPPU : Dec.-05,07,08,09,10,15,17,18,19, May-06,07,10,12,14,15,16, Marks 12

- The TM can perform string operations, arithmetic computations, recognizing languages either regular or nonregular languages.
- 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.

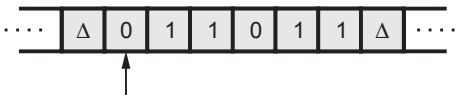


Fig. 5.6.1 Two way infinite tape

- The tape head as usual can move in forward and backward direction.

Theorem : L is recognized by a turing machine with a two way infinite tape if and only if it is recognized by a TM with one way infinite tape.

Proof : As theorem states, if any language is recognized by a TM with one way infinite tape then it should also be recognized by a TM with two way infinite tape.

Let M_1 be a TM with one way infinite tape and can be denoted by,

$$M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_1, B, F_1)$$

Similarly, M_2 be a TM with two way infinite tape and can be denoted by,

$$M_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2, S_1, B, F_2)$$

The input tapes are as shown by following figure.

In Fig. 5.6.2, the TM with one way infinite tape has a external symbol # placed in the very first cell from left side. This symbol is used as indicator for the left side termination.

If we want a language $L = \{q_0, q_1, q_2, q_3, q_4, q_5\}$ sequence then in the two way infinite tape the tape head is fixed at the rightmost symbol.

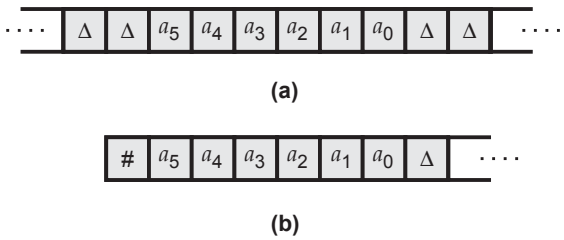


Fig. 5.6.2 (a) Two way infinite tape for M_2 TM (b) One way infinite tape for M_1 TM

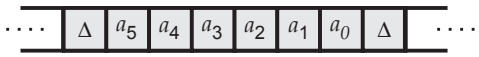


Fig. 5.6.3

The TM M_2 can be

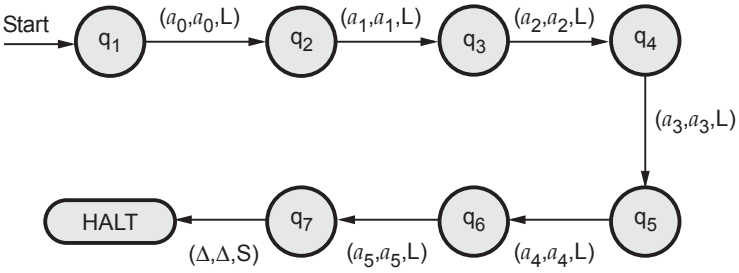


Fig. 5.6.4

$\delta(q_7, \Delta) \rightarrow$ (HALT, Δ, S) leads to HALT state.

Similarly, with one way infinite tape the machine M_1 will be

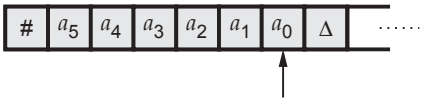


Fig. 5.6.5 (a)

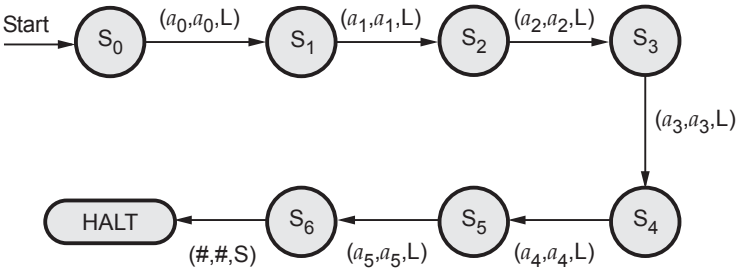


Fig. 5.6.5 (b)

Even we can make the TM with one way infinite tape as a multitrack tape to simulate it as a two way infinite tape. Let us now solve some interesting problems to see the working of two way infinite tape in the turing machine.

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* $\Sigma = \{a, b\}$. **SPPU : Dec.-08, Marks 6; May-12, Marks 8; May-14, Marks 7, Dec.-17,18, End Sem, Marks 8**

Solution : Here we will clearly take the advantage of infinite two ends of the input tape. The input can be placed as follows.

Fig. 5.6.6

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.

Δ b *a* *a* b Δ ↑	Convert b to B and move right in search of *a*
Δ B *a* *a* b Δ ↑	*a* has to be converted to A and move left
Δ B A *a* b Δ ↑	Skip B and move left
Δ B A *a* b Δ ↑	Keep Δ as it is and move right
Δ B A *a* b Δ ↑	Move right
Δ B A *a* b Δ ↑	Move right
Δ B A *a* b Δ ↑	Convert *a* to A and move right
Δ B A A b Δ ↑	Convert b to B move left
Δ B A A B Δ ↑	Move left

Δ B A A B Δ ↑	Move left
Δ B A A B Δ ↑	Move left
Δ B A A B Δ ↑	If Δ comes move to right by ignoring all A's and B's
Δ B A A B Δ ↑	Move right
Δ B A A B Δ ↑	Move right
Δ B A A B Δ ↑	Move right
Δ B A A B Δ ↑	Move right
Δ B A A B Δ ↑	Δ is reached, goto HALT state

The TM can be

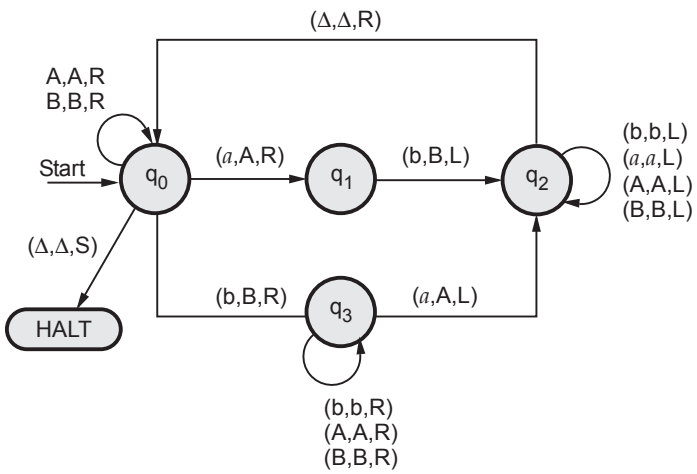


Fig. 5.6.7

Similarly if the string is

Δ *a* b *a* b Δ ↑	Convert *a* to A move right
Δ A b *a* b Δ ↑	Convert b to B move left

Δ A B *a* b Δ ↑	Skip A and move left
Δ A B *a* b Δ ↑	Move right, ignore all A's and B's
Δ A B *a* b Δ ↑	Convert *a* to A, move right
Δ A B A b Δ ↑	Convert b to B, move left
Δ A B A B Δ ↑	Skip all A's and B's and move left
Δ A B A B Δ ↑	Move right by skipping all A's and B's
Δ A B A B Δ ↑	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.*

SPPU : Dec.-05, Marks 10; Dec.-10, Marks 8, Dec.-19, End Sem, Marks 6

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

	0 1 1 0	
	1 0 0 1	1's complement
+	1	Adding 1
=	1 0 1 0	Two's complement of 0110

By our idea also we will get the same result.

Δ 0 1 1 0 Δ ↑	Move to rightmost end upto Δ
Δ 0 1 1 0 Δ ↑	Move left
Δ 0 1 1 0 Δ ↑	It is 0, so keep it as it is and move left
Δ 0 1 1 0 Δ ↑	First 1 from LSB has encountered so keep it as it is and move left

Δ 0 1 1 0 Δ ↑	Complement it to 0 and move left
Δ 0 0 1 0 Δ ↑	Convert it to 1 and move left
Δ 1 0 1 0 Δ ↑	Since Δ is reached, stop !

The TM could be

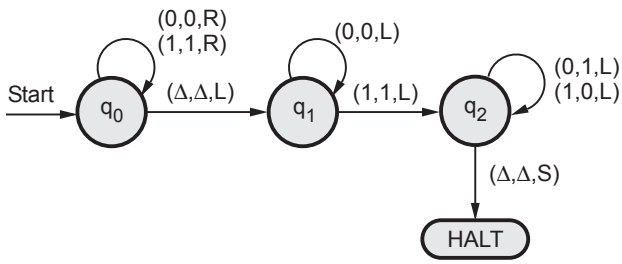


Fig. 5.6.8

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

Δ ( ) ( ) Δ ↑	Move right in search of ')', skipping '('
Δ ( ) ( ) Δ ↑	Mark it as ∗ and move left, in search of '('
Δ (∗ ( ) Δ ↑	Now '(' has encountered mark it as ∗ Move left in search of '(', mark it *
Δ ∗ ∗ ( ) Δ ↑	Move right, skip ∗
Δ ∗ ∗ ( ) Δ ↑	Move right
Δ ∗ ∗ ( ) Δ ↑	Convert to ∗ and move left

$\Delta * * (\ * \Delta$ Convert (to $*$

$\uparrow$

$\Delta * * * * \Delta$ Move right till Δ

$\uparrow$

$\Delta * * * * \Delta$ HALT

$\uparrow$

The TM will look like this -

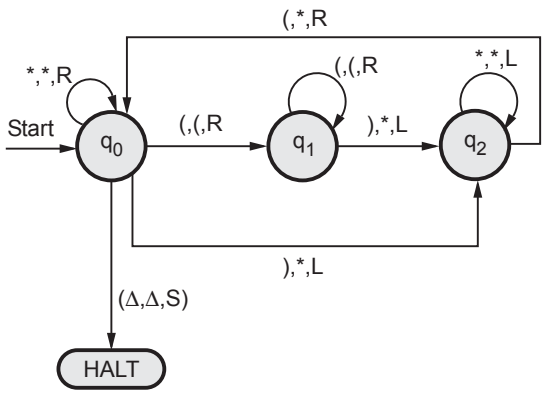


Fig. 5.6.9

The transition table for this TM will be

State	(	)	*	Δ
q_0	$(q_1, (, R)$	$(q_2, *, L)$	$(q_0, *, R)$	(HALT, Δ, 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.*

SPPU : Dec.-09, May-10, Marks 10

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

... Δ 1 0 1 0 0 1 Δ Δ Δ ... We will move upto the end of the input string

$\uparrow$

Δ 1 0 1 0 0 1 Δ Δ ... ↑	Move left
Δ 1 0 1 0 0 1 Δ Δ ... ↑	Mark it ∗ and copy it after Δ
Δ 1 0 1 0 0 ∗ Δ 1 ↑	Move left upto ∗
Δ 1 0 1 0 0 ∗ Δ 1 Δ ↑	Move left, convert 0 to * and place it, at the end
Δ 1 0 1 0 ∗ ∗ Δ 1 0 Δ ↑	Move left upto ∗
Δ 1 0 1 0 ∗ ∗ Δ 1 0 Δ ↑	Move left, convert 0 to ∗ and place it at the end
Δ 1 0 1 ∗ ∗ ∗ Δ 1 0 0 Δ ↑	Move left upto ∗
Δ 1 0 1 ∗ ∗ ∗ Δ 1 0 0 Δ ↑	Move left, convert 1 to ∗ and place it at the end
Δ 1 0 ∗ ∗ ∗ ∗ Δ 1 0 0 1 Δ ↑	Move left upto ∗
Δ 1 0 ∗ ∗ ∗ ∗ Δ 1 0 0 1 Δ ↑	Move left, convert 0 to * and place it at the end
Δ 1 ∗ ∗ ∗ ∗ ∗ Δ 1 0 0 1 0 Δ ↑	Move left skipping *
Δ 1 ∗ ∗ ∗ ∗ ∗ Δ 1 0 0 1 0 Δ ↑	Copy 1 at the end of tape by converting 1 to *
Δ ∗ ∗ ∗ ∗ ∗ ∗ Δ 1 0 0 1 0 1 Δ ↑	Move left skipping all ∗
Δ ∗ ∗ ∗ ∗ ∗ ∗ Δ 1 0 0 1 0 1 Δ ↑	Left to ∗ is a Δ character so we goto HALT state

The TM will be,

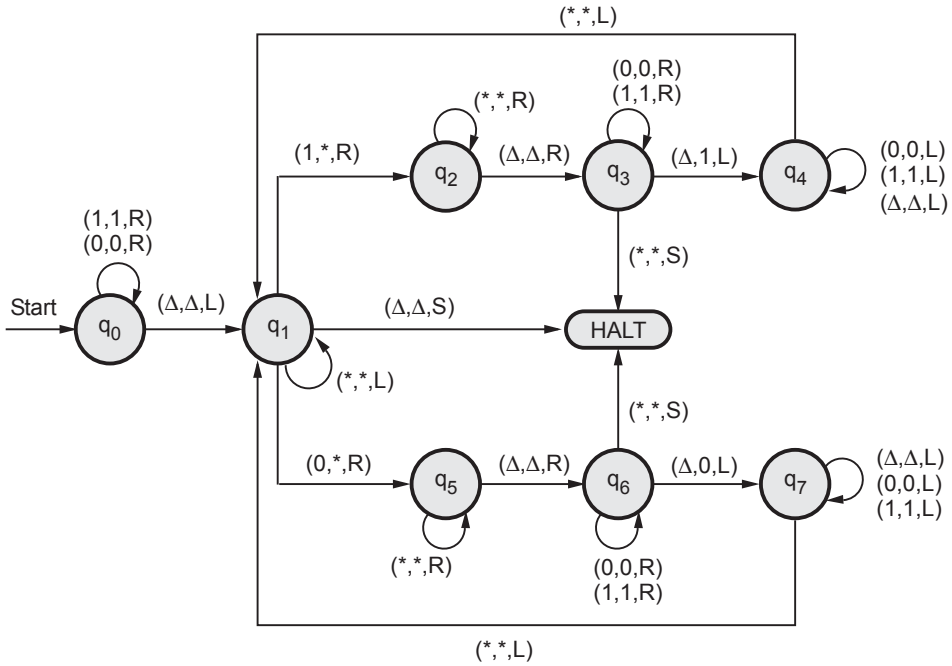


Fig. 5.6.10

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, Δ, 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 *Construct TM for copying the input unary string on the tape.*

SPPU : May-14, Marks 8

Solution : The turing machine is able to perform copy operation also. In this TM will read each symbol mark it as A (to keep track of visited symbols) and copy it at the right end after Δ.

Finally after copying the complete string we will move left and reconvert 'A' to 1 again.

The TM will be as follows.

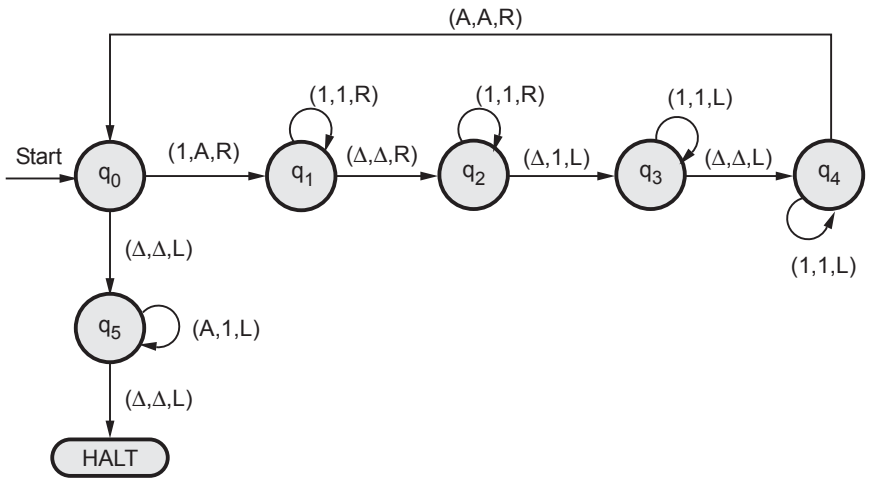


Fig. 5.6.11

Let us simulate it.

.... Δ 1 1 1 1 Δ ... ↑	Mark 1 as A and move right refer start state q_0
Δ A 1 1 1 Δ ↑	Keep 1 as it is and move right, see q_1 state
Δ A 1 1 1 Δ ↑	Move right
Δ A 1 1 1 Δ ↑	Move right
Δ A 1 1 1 Δ Δ ↑	Move right simply transition from q_1 to q_2
Δ A 1 1 1 Δ 1 ↑	Converted Δ to 1 and move left
Δ A 1 1 1 Δ 1 ↑	Move left

```
Δ A 1 1 1 Δ 1          Move left by keeping 1's as it is till 'A'
        ↑

Δ A 1 1 1 Δ 1          Move right
  ↑

Δ A 1 1 1 Δ 1          Convert 1 to A and move right upto Δ
    ↑

Δ A A 1 1 Δ 1 Δ        Move right even after 1
          ↑

Δ A A 1 1 Δ 1 Δ        Convert Δ to 1 and move left upto A
              ↑

Δ A A 1 1 Δ 1 1        Move right convert 1 to A
    ↑

Δ A A A 1 Δ 1 1 Δ      Convert Δ to 1
                ↑

Δ A A A 1 Δ 1 1 1      Move left upto A
                ↑

Δ A A A 1 Δ 1 1 1      Move right convert 1 to A and copy it at the end
        ↑

Δ A A A A Δ 1 1 1 1    Move left upto Δ
                ↑

Δ A A A A Δ 1 1 1 1    Move left and convert A to 1 upto Δ
          ↑

Δ A A A 1 Δ 1 1 1 1
        ↑

Δ 1 1 1 1 Δ 1 1 1 1    Here TM HALT
↑
```

Example 5.6.6 *Construct TM for performing logarithmic operation of given binary number i.e. compute log (n) function.*

Solution : For obtaining logarithmic value of the binary number we will enforce *a* condition as, that binary number has to be in the power of 2 form.

That means 1, 10, 100, 1000, 10000, ... are valid inputs for our TM.

$$\log(1) = 0$$

$$\log(10) = \log(2) = 1$$

$$\log(100) = \log(4) = 2$$

Observe the above examples carefully and you will find that the number of zeros following 1 is the answer for logarithmic value.

For example

log (100) = Number of zeros after 1 and 2 are even answer

= 2

The TM will be as follows -

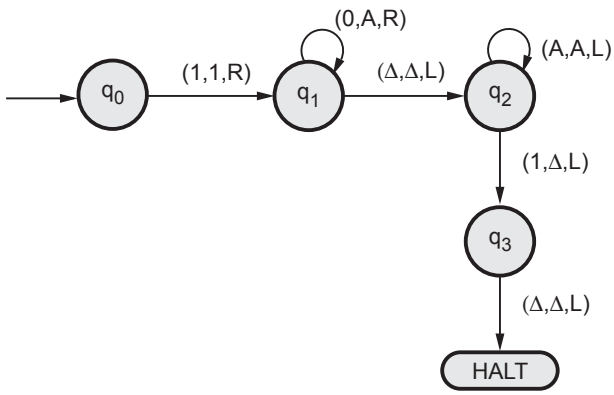


Fig. 5.6.12

Let us take some input as log (8) = 3.

Δ 1 0 0 0 Δ ↑	Move right
Δ 1 0 0 0 Δ ↑	Convert 0 to A and move right
Δ 1 A 0 0 Δ ↑	Convert 0 to A and move right
Δ 1 A A 0 Δ ↑	Convert 0 to A and move right
Δ 1 A A A Δ ↑	Now move left
Δ 1 A A A Δ ↑	Move left, ignoring A's upto 1
Δ 1 A A A Δ ↑	Convert 1 to Δ
Δ Δ A A A Δ ↑	Move left
Δ Δ A A A Δ ↑	The answer is in the unary form and that is number of A's = 3

Example 5.6.7 *Design TM to recognize an arbitrary string divisible by 4 from Σ = {0, 1, 2}.*

SPPU : May-15, End Sem, Marks 10

Solution : As input string is Σ = {0, 1, 2}, any arbitrary string of input should be a ternary string and we have check whether such ternary input is divisible by 4 or not. For divisibility by 4 there are four case -

Remainder 0, Remainder 1, Remainder 2, Remainder 3

Now we will consider separate states for each of these cases -

q_0 - Remainder 0 state

q_1 - Remainder 1 state

q_2 - Remainder 2 state

q_3 - Remainder 3 state

HALT : is a final. When you get a valid string (string which is divisible by 4). The TM is using a triplet of (input, output, direction) during the transition from one state to another.

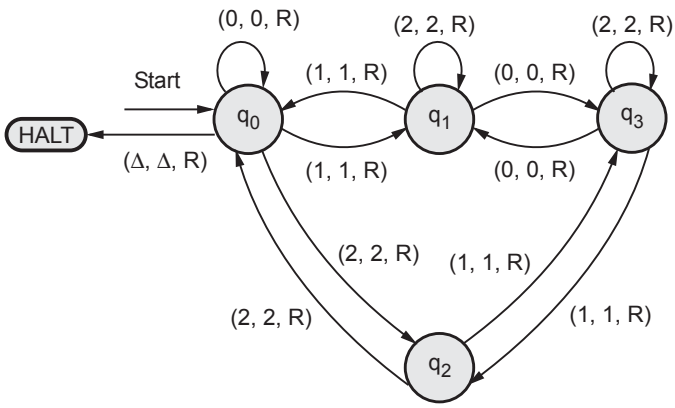


Fig. 5.6.13

1	2	1	Δ	……

Input tape with sample input

For the sample input string 121Δ we will reach to halt state finally as $(121)_3 = (16)_{10}$ which is divisible by 4.

Example 5.6.8 *Construct TM for the function f(x) = x + 3.*

Solution : We will represent number x using unary number. Hence the input set Σ = {1}. If we assume x = 4 then sample input tape can be represented as

1	1	1	1	Δ	Δ	Δ	……

Input tape with x = 4

Addition by constant 3 means we will simply convert last three Δ's to 1. Hence the required TM will be -

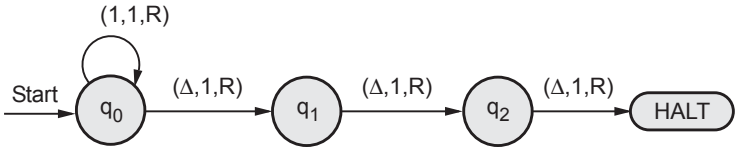


Fig. 5.6.14

The transition table can be as shown below.

State \ Input	1	Δ
q_0	(1, q_0, R)	(1, q_1, R)
q_1	–	(1, q_2, R)
q_2	–	(1, HALT, R)
HALT	–	–

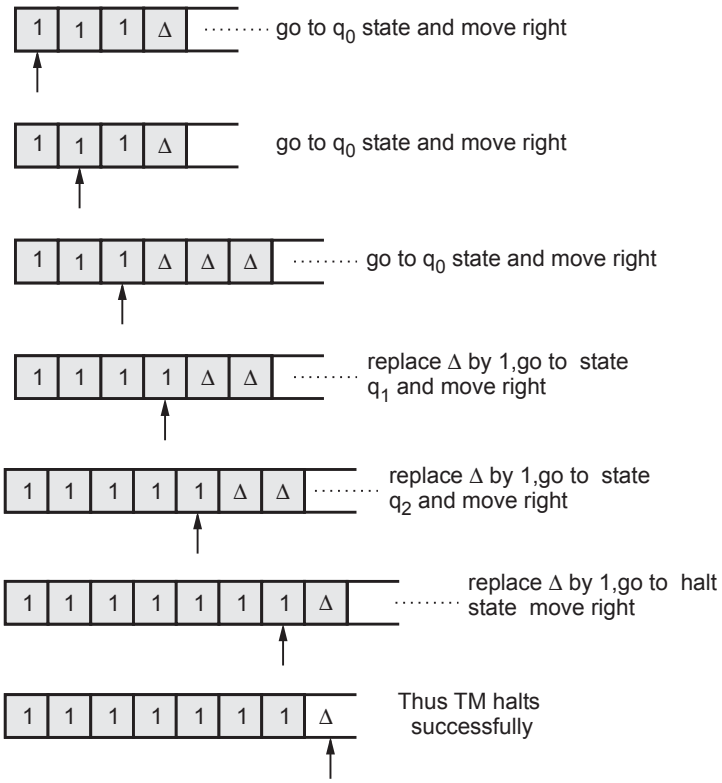


Fig. 5.6.14 (a)

Example 5.6.9 *Let T be the turing machine defined by the 5-tuples*

$(S_0, 0, 0, S_1, L)$ $(S_0, 1, 0, S_0, L)$

$(S_0, B, B, Halt, L)$ $(S_1, 0, 1, S_0, L)$

$(S_1, 1, 1, S_0, R)$

for each of the following initial tapes, determine the final tape when T halts, assuming that T begins in initial position.

a. 110B b. 0011B c. 0101B **SPPU : May-06, 07, Dec.-07, Marks 12**

Solution : The turing machine can be drawn from given δ function, which is as follows -

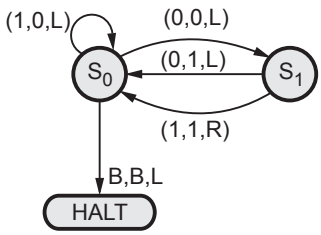


Fig. 5.6.15

a)			1	1	0	B	In S_0, Convert 1 to 0 and move left.
			↑				
		↑	0	1	0	B	If we assume B (i.e. blank symbol on the left side i.e. if there is a two way tape) then HALT. Leaving "010" on tape.
b)		0	0	1	1	B	Convert 0 to 0 and move left
		↑					
	↑	0	0	1	1	B	In state S_1 there is no transition for B. Hence just leave " 0011" on the tape without entering the HALT state.
c)		0	1	0	1	B	Convert 0 to 0, move left, go to state S_1.
		↑					

No transition is possible. Hence leave "0101" on the tape.

Example 5.6.10 *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.

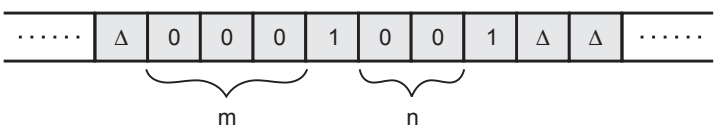


Fig. 5.6.16

The TM can be

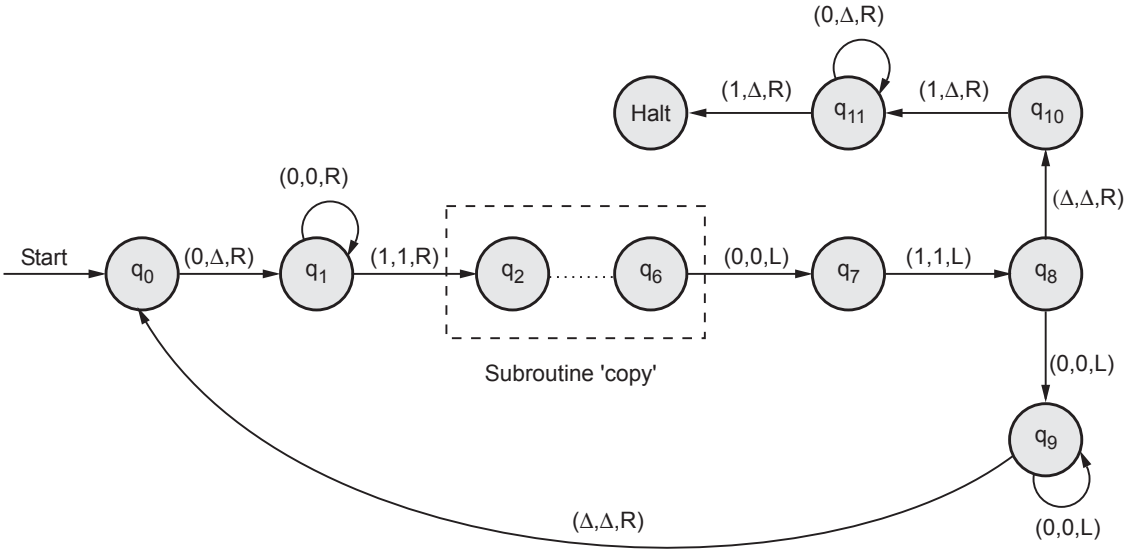


Fig. 5.6.17

The subroutine copy can be as shown below.

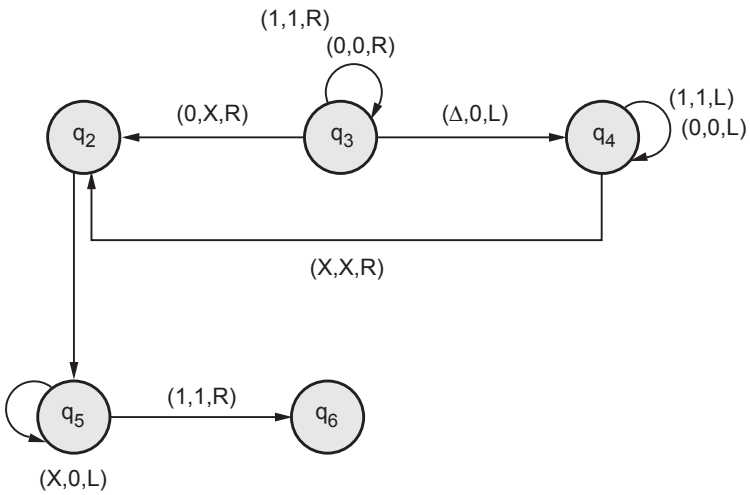


Fig. 5.6.18

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.

Δ 0 0 0 1 0 0 1 Δ … Convert to Δ and move right.

↑

Δ Δ 0 0 1 0 0 1 Δ Δ Δ Δ Δ Δ … Move right.

↑

```
Δ Δ 0 0 1 0 0 1 Δ Δ Δ...        Move right.
    ↑
Δ Δ 0 0 1 0 0 1 Δ Δ Δ ...       Move right and call 'copy'.
      ↑
Δ Δ 0 0 1 0 0 1 Δ Δ Δ ...       Convert 0 to X and move right.
        ↑
Δ Δ 0 0 1 X 0 1 Δ Δ Δ ...       Move right.
          ↑
Δ Δ 0 0 1 X 0 1 Δ Δ  Δ ...      Move right.
            ↑
Δ Δ 0 0 1 X 0 1 Δ ...           Convert to 0 and move left.
              ↑
Δ Δ 0 0 1 X 0 1 0 Δ Δ ...       Move left.
          ↑
Δ Δ 0 0 1 X 0 1 0 Δ Δ           Move left.
      ↑
Δ Δ 0 0 1 X 0 1 0 Δ Δ           Move right.
        ↑
Δ Δ 0 0 1 X X 1 0 Δ Δ           Convert 0 to X, move right.
        ↑
Δ Δ 0 0 1 X X 1 0 Δ Δ           Move right, skip all 0's and 1's.
              ↑
Δ Δ 0 0 1 X X 1 0 Δ             Convert Δ to 0 and move left.
                ↑
Δ Δ 0 0 1 X X 1 0 Δ             Skip all 1's and 0's and reach at
                ↑               X by moving left.
Δ Δ 0 0 1 X X 1 0 0 Δ           Move right.
              ↑
Δ Δ 0 0 1 X X 1 0 0 Δ           Move left and convert X to 0's.
                ↑
```

Δ Δ 0 0 1 0 0 1 0 0 Δ Move right.
↑

Δ Δ 0 0 1 0 0 1 0 0 Δ Move left, skip all 0's and 1's
↑ and reach at Δ by moving left.

Δ Δ 0 0 1 0 0 1 0 0 Δ Move right convert 0 to Δ .
↑

Δ Δ Δ 0 1 0 0 1 0 0 Δ
↑

Now again perform copy operation as illustrated above. The tape will then consist of

Δ Δ Δ 0 1 0 0 1 0 0 0 0 Δ Move right convert 0 to Δ and
↑ repeat the above illustrated steps.

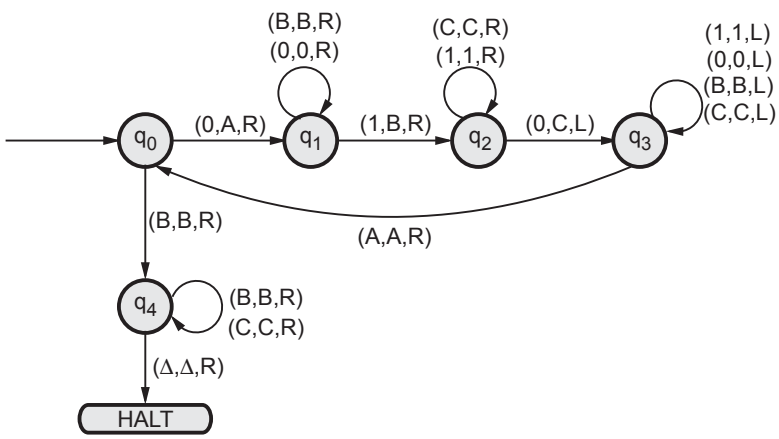
Δ Δ Δ Δ 1 0 0 1 0 0 0 0 0 0
↑

As we have read out the value of m from m × n we will convert remaining symbols to Δ leaving only multiplication result on the tape.

Thus finally Δ Δ Δ Δ Δ Δ Δ 0 0 0 0 0 0 Δ HALT

Example 5.6.11 *Design turing machine that accepts a language* $L = \{0^n 1^n 0^n \mid n >= 1\}$. **SPPU : May-15, End Sem, Dec.-18, End Sem, Marks 8**

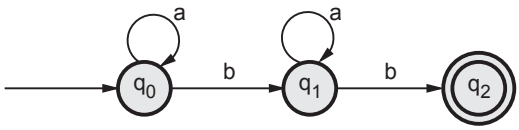
Solution :



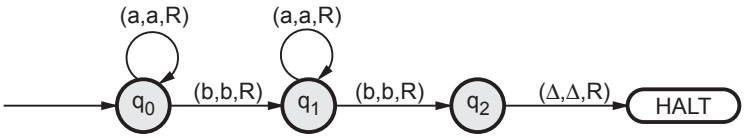
Example 5.6.12 *Construct a TM that accepts a language L, a^*ba^*b.*

SPPU : May-15, End Sem, Marks 6

Solution : We will construct FA for given L i.e. a*ba*b



The TM will be,

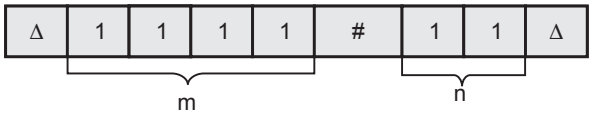


Example 5.6.13 *Design turing machines for each of the following problems : i) Given two unary numbers, m and n, display,*

'G', if m > n, 'E', if m = n, 'L', if m < n

SPPU : Dec.-15, End Sem, Marks 10

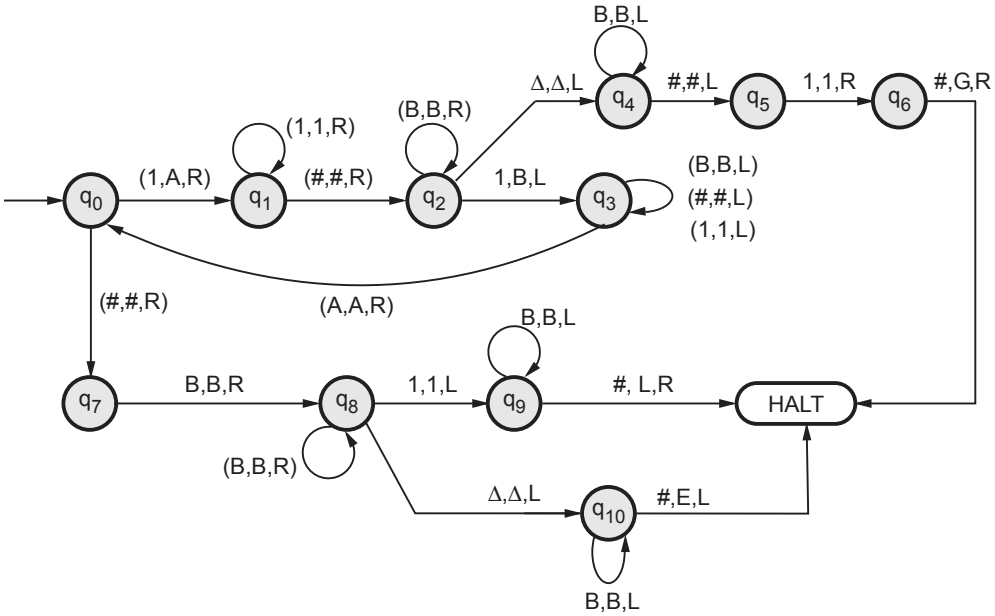
Solution : i) The two unary numbers are placed on the tape and are separated by #. For example



The logic can be as follows :

m > n case	m < n case	m = n case
Δ 1111#11 Δ	Δ 11 # 1111 Δ	Δ11 # 11 Δ
Δ A111#B1 Δ	Δ A1# B111 Δ	ΔA1 # B1 Δ
Δ AA11#BB Δ	Δ AA # BB11 Δ	ΔAA # BB Δ
Δ AA11GBB Δ	Δ AA L BB11 Δ	Δ AA E BB Δ

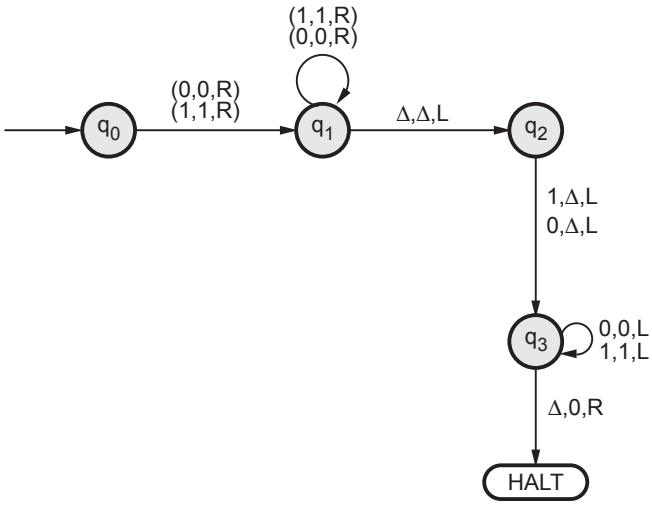
The TM will be -



Example 5.6.14 *Design a turing machine to perform right shift operation on a binary number.*

SPPU : Dec.-15, End Sem, Marks 6

Solution : In right shift operation the LSB bit is discarded and at MSB the padding 0 is attached. The right shift operation also means division by 2. The TM will be,



Example 5.6.15 *Construct a two tape TM to convert an input W into WW^R*

SPPU : May-16, End Sem, Marks 6

Solution :

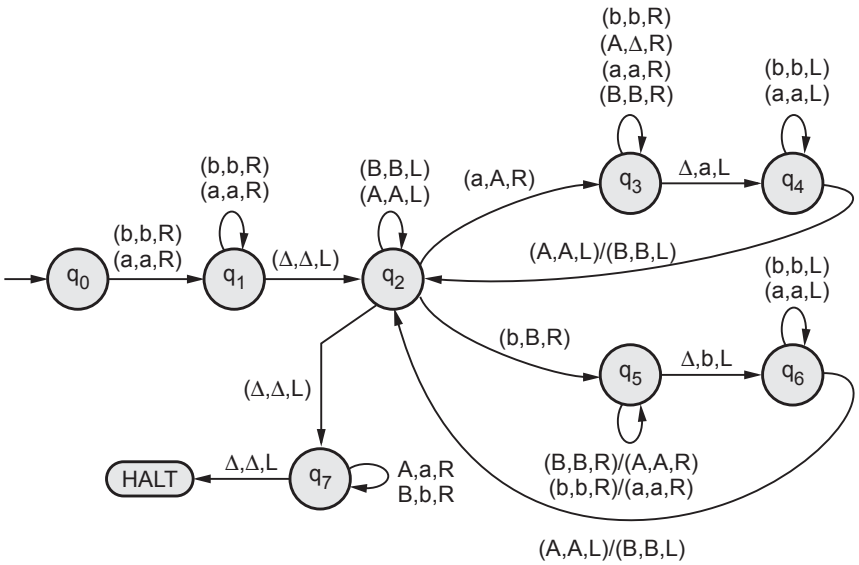


Fig. 5.6.19

Simulation of string Δa b a b Δ

Δab ab Δ … ↑	Read a, move right towards Δ.
Δab ab Δ ↑	Read Δ, keep Δ as it is and move left.
Δab ab Δ ↑	Read b, convert it to B, move right.
Δab aBΔ ↑	Convert Δ to be and move left.
Δab aB b ↑	Read B as it is and move left.
Δab aB bΔ ↑	Read a, convert it to A move right.
Δab aB b Δ ↑	Move towards Δ.
Δab A B b Δ ↑	Convert Δ to a and move left.

Δab A B b aΔ ↑	Keep moving to left by skipping a's or b's.
Δab A B b aΔ ↑	Read B, move left. Skip all capital letters and move towards left.
Δab A B b aΔ ↑	Read b, convert it to B and move right.
ΔaB A B b aΔ ↑	Go on moving right towards Δ.
ΔaB A B b aΔ ↑	Convert Δ to b and move left.
ΔaB A B b ab Δ ↑	Keep moving left towards a.
ΔaB A B bab Δ ↑	Convert a to A. Keep moving right towards Δ.
Δ A B A B b ab Δ ↑	Convert Δ to a
Δ A B A B b ab aΔ ↑	Keep moving left towards A.
Δ A B A B b ab aΔ ↑	Move left.
Δ A B A B b ab aΔ ↑	Move right convert all capitals to small letters.
Δab ab b ab aΔ ↑	HALT.

5.7 Variants of Turing Machines

SPPU : Dec.-17, Marks 4

There are two types of Turing machines - Deterministic turing machine and non deterministic turing machine.

5.7.1 Deterministic Turing Machine (DTM)

The deterministic Turing machine is a kind of turing machine in which the set of rules denote the single specific action on reading particular input in current specific state. It is just similar to deterministic finite automata.

For example - Refer example 5.5.12.

5.7.2 Non Deterministic Turing Machine (NTM)

The non deterministic turing machine is a kind of turing machine in which the set of rules denote more than one specific action on reading particular input in current specific state. This kind of TM just similar to NFA.

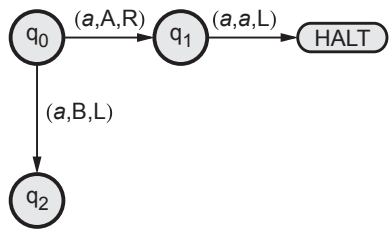


Fig. 5.7.1 NTM

For example -

Note that any language accepted by non deterministic turing machine can also be accepted by deterministic turing machine.

The power of non deterministic turing machine is just similar to the power of deterministic turing machine.

5.7.3 Universal Turing Machine

- 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.
- 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.

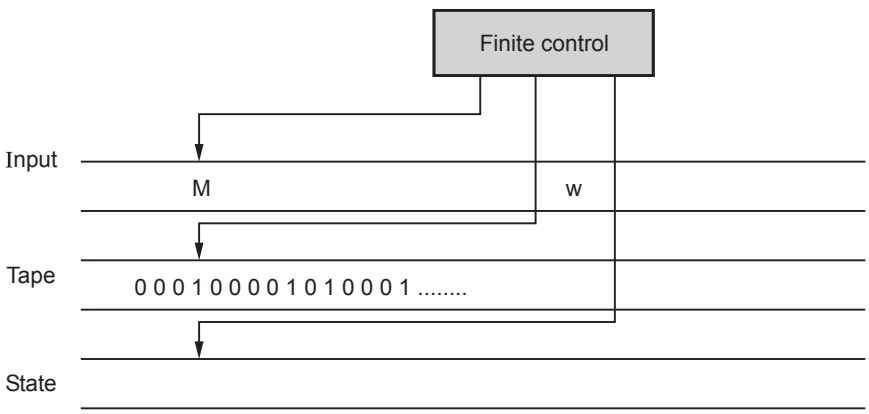


Fig. 5.7.2

5.7.4 Multi-tape 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. 5.7.3.

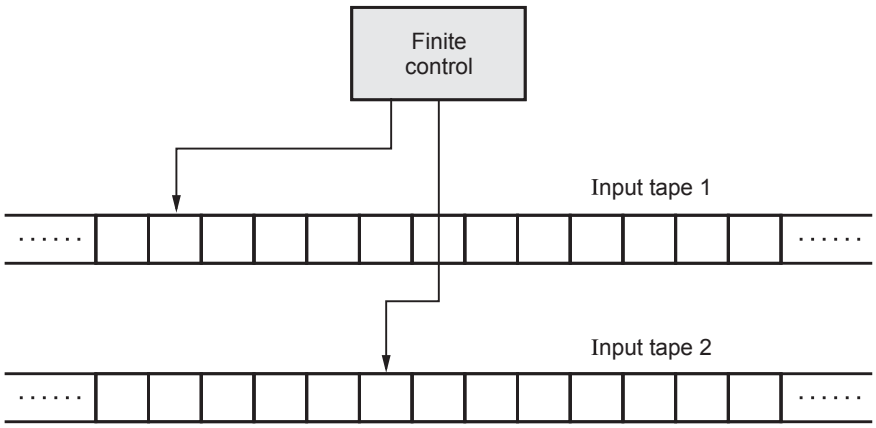


Fig. 5.7.3 A Multitape turing machine

Review Question

1. *Write short note on - i) Universal turing machine ii) Multitape turning machine.*

SPPU : Dec.-17, End Sem, Marks 4

5.8 Halting Problem of TM

SPPU : May-16,19, Dec.-17,19, Marks 4

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."

The halting problem is unsolvable.

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. 5.8.1.

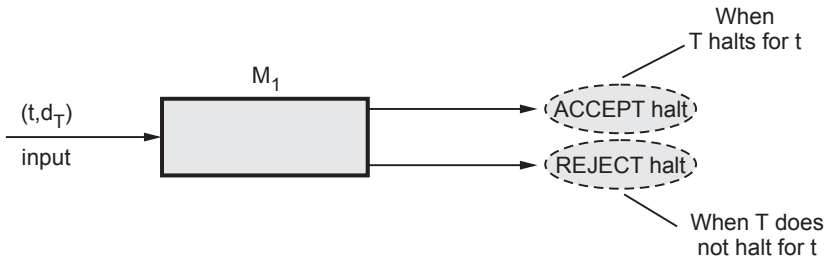


Fig. 5.8.1

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.

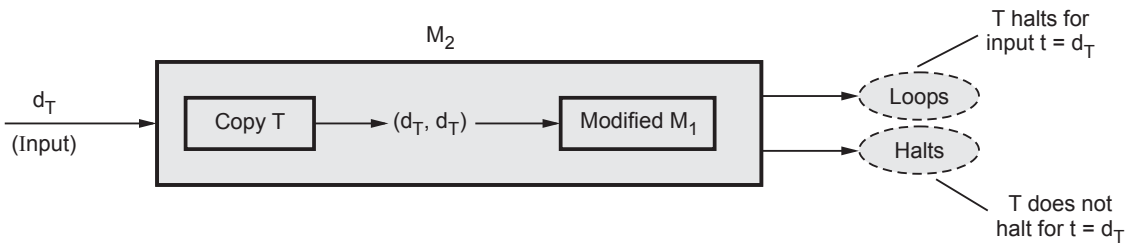


Fig. 5.8.2

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.

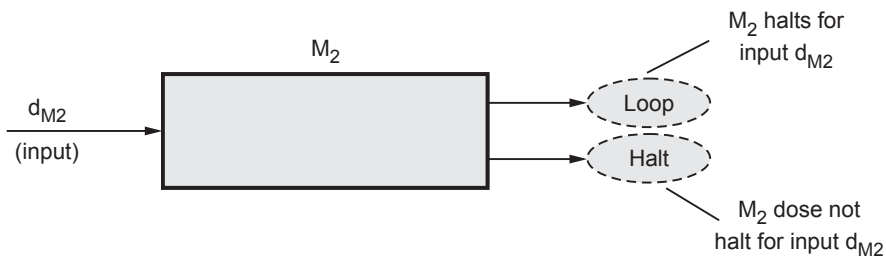


Fig. 5.8.3

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.

Limitations of Turing Machine

Turing machine is a model which can solve any problem which has some algorithm. But not all the problems be solved by TM. There are some problems which cannot be solved by TM. For example : Halting problem cannot be solved by TM. The Halting problem can be stated as follows - "It is not possible to determine whether a TM will ever halt on particular input". Such a problem is called **unsolvable problem**. There some more problems that are not solvable by TM -

i) Deciding whether two context free grammars are equivalent.

ii) Whether a given program will loop forever on particular input.

iii) If two CFGS are given G_1 and G_2 then $L(G_1) \cap L(G_2) = \phi$ is undecidable.

iv) Whether the language accepted by TM is empty it is not possible for a TM to answer "yes" or "no" for above problems. Hence these problems are unsolvable by TM.

Example 5.8.1 *Differentiate between FA and TM.* **SPPU : May-19, End Sem, Marks 2**

Solution :

Sr. No.	FA	TM
1.	It accepts regular languages.	It accepts recursive and recursively enumerable languages.
2.	It can not be programmed.	It can be programmed.
3.	It accepts small class of language.	It accepts large class of languages.

Review Questions

1. *Write a short note on : Halting problem.* **SPPU : May-16, End Sem, Marks 3, Dec.-19, End Sem, Marks 4**
2. *Write short note on - Limitations of turing machine.* **SPPU : Dec.-17, End Sem, Marks 2**
3. *Write short notes on : i) Universal turing machine ii) Multi-tape turing mahine iii) Limitation of turing machine.* **SPPU : Dec.-17, End Sem, Marks**

5.9 Halting Vs Looping

Halting problem is denoted by following question - "Given a program or algorithm will ever halt or not.

Halting means that the program on certain input will either **accept it and halt** or **reject it and halt.** It would never go into an infinite loop. Basically halting means terminating. So can we have an algorithm that will tell that the given program will halt or not.

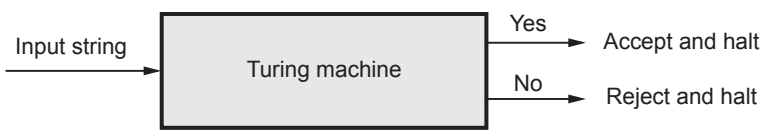


Fig. 5.9.11 Halting

Looping means the program on certain input will either accept it and **enter it in infinite loop** or **reject it and halt.**

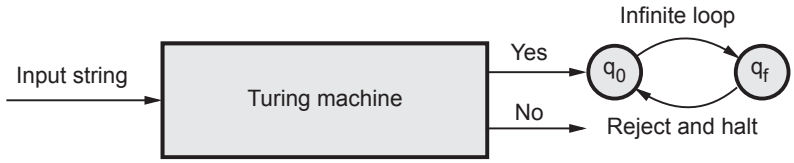


Fig. 5.9.2 Looping

The halting and looping both are undecidable problems.

5.10 A Turing - unrecognizable Language

- Turing machine can accept Type 0 Grammar.
- The type 0 grammar is said to be unrestricted grammar. Examples of unrestricted languages are almost all natural languages.
- The class of languages accepted by Type 0 grammar are called **recursively enumerable languages.**
- The productions can be in the form of $\alpha \rightarrow \beta$ where α is a string of terminals and non-terminals with at least one non-terminal and α cannot be null. β is a string of terminals and non-terminals.

There are two type of languages - 1) Recursive language and 2) Recursively Enumerable (RE) language.

Recursive Languages :

A language is said to be recursive if there exists a Turing machine that **accepts** every string of the language and every string is **rejected** if it is not belonging to that language.

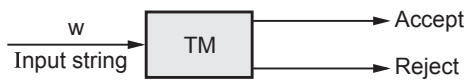


Fig. 5.10.1 TM Accepting Recursive Language

Recursively Enumerable Languages :

A language is said to be recursive if there exists a Turing machine that **accepts** every string belonging to that language. And if the string does not belong to that language then it can cause a Turing machine to enter in an **infinite loop**.

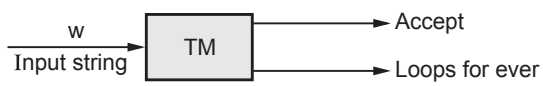


Fig. 5.10.2 TM Accepting RE language

A language is recognizable iff there is a turing machine which will halt and accept only the strings in that language and for strings not in the language, the TM either rejects, or does not halt at all.

Let, the language L is denoted by,

$$A_{TM} = \{<M, w> \mid M \text{ is a TM and M accepts } w\}$$

is undecidable. That is, there does not exist a turing machine that :

1. Take as input (some encoding of) M and w,
2. Decide whether M accepts w.

Here L is turing recognizable if there is a turing machine that accepts exactly the words in L, but can either reject or loop indefinitely on an input that is not in L.

To prove that, We will show that $\overline{A_{TM}}$ the complement of ATM , is not turing-recognizable. The proof is by contradiction. Suppose A_{TM} is Turing recognizable, Suppose was also Turing recognizable.

Then A_{TM} would be decidable, because we could just make a machine that runs the recognizers for A_{TM} and $\overline{A_{TM}}$ in parallel, and accept if A_{TM} 's recognizer accepts, and rejects if $\overline{A_{TM}}$'s accepts. However, we know that A_{TM} (the famous halting problem) is undecidable, so we have a contradiction. Thus, $\overline{A_{TM}}$ is turing - Unrecognizable.

5.11 Reducibility

- Reducibility is problem solving technique in mathematics.
- Let A and B are two problems and A is reduced to B. If we solve B, we solve A as well. If we can't solve A, we can't solve B.
- For example - If we solve the Eulerian cycle problem, we solve the Eulerian path problem.
- To decide if a problem is solvable or not we use mapping reducibility technique.
- If there is a mapping reduction from language A to language B, we say that language A is mapping reducible to language B.
- Notation used for mapping reducibility is $A \leq M^B$ iff language A is mapping reducible to language B.
- If A is reduced to B and B is decidable, then A is decidable.
- If A is undecidable and reducible to B then B is also undecidable.

5.12 Recursion Theorem

Recursive function is class of functions those are turing computable.

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) = successor of x (roughly , "x+1")

The identity function : id(x) = x

The zero functions returns zero regardless of its argument.

The successor functions 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 functions 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,

$$id(x, y) = x$$

$$id(x, y) = y$$

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 -

$$1! = 1$$

$$n! = n*(n - 1)$$

A strict definition of the factorial function, f(n) then, consists of these two equations :

$$f(n) = 1 \qquad \text{when } n = 1 \qquad ...(5.12.1)$$

$$f(n) = n*(f(n-1)) \qquad \text{when } n > 1 \qquad ...(5.12.2)$$

Consider n = 5 then using equation (5.12.2) we will get,

$$f(5) = 5*f(4)$$
$$f(4) = 4*f(3)$$
$$f(3) = 3*f(2)$$
$$f(2) = 2*f(1)$$

By putting the value of equation (5.12.1) for calculating f(5.12.2) we will get,

$$f(2) = 2*1 = 2$$

Then $f(3) = 3*f(2)$
$= 3*2$
$= 6$

Then $f(4) = 4*f(3)$
$= 4*6$
$= 24$

Then $f(5) = 5*f(4)$
$= 5*24$
$= 120$

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.

5.12.1 Class of Recursive Functions

The classification of recursive functions is as shown in Fig. 5.12.1.
(See Fig. 5.12.1 on next page).

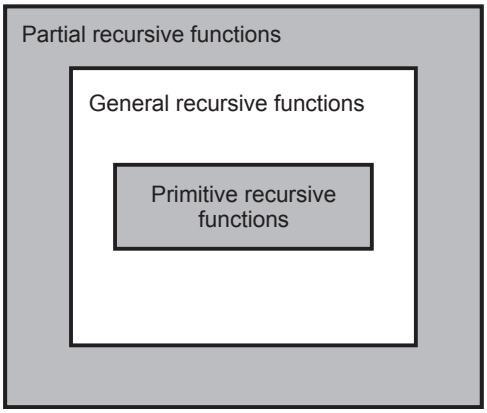


Fig. 5.12.1 Classification of recursive functions

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.

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 5.12.1 *Compute A(1, 1) A(1, 2), A(2, 1), A(2, 2) using Ackermann's function.*

Solution : Let,

$$A(0, y) = y + 1 \quad \text{... (1)}$$

$$A(x+1, 0) = A(x, 1) \quad \text{... (2)}$$

$$A(x+1, y+1) = A(x, A(x+1, y)) \quad \text{... (3)}$$

By the Ackermann's function,

To compute A(1, 1) put x = 0, y = 0, then

$$\begin{aligned} A(1, 1) &= A(0 + 1, 0 + 1) && \text{...Using equation (3)} \\ &= A(0, A(0 + 1, 0)) \\ &= A(0, A(0, 1)) && \text{...Using equation (2)} \\ &= A (A (0, 1)) && \text{... Using equation (1)} \\ &= A(0, 2) && \text{...Using equation (1)} \end{aligned}$$

Hence A(1, 1) = 3

To compute A(1, 2) put x = 0 and y = 1

$$\begin{aligned} A(1, 2) &= A(0 + 1, 1 + 1) && \text{...Using equation (3)} \\ &= A(0, A(1, 1)) \\ &= A(0, 3) && \text{...Using equation (1)} \\ &= 4 \end{aligned}$$

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.

$$A(1, 5) = A(0 + 1, 4 + 1)$$

$$= A(0, A(1, 4))$$
$$= A(0, (A(0 + 1, 3 + 1)))$$
$$= A(0,(A(0,A(1, 3))))$$
$$= A(0,(A(0,A(0 + 1,(2 + 1))))$$
$$= A (0, (A (0,A(0,A(1, 2))))$$
$$= A(0,(A(0,A(0,(4))))$$
$$= A(0,(A(0, 5)))$$
$$= A(0, 6)$$

$$A(1, 5) = 7$$

Hence $A(2, 2) = A(1, 5)$

$$A(2, 2) = 7$$

5.13 The Model of Linear Bounded Automata

- The Linear Bounded Automata (LBA) is a model which was originally developed as a model for actual computers rather than model for computational process.
- A linear bounded automaton is a restricted form of a non deterministic Turing machine.

Concept : A linear bounded automaton is a multitrack Turing machine which has only one tape and this tape is exactly of same length as that of input.

- The Linear Bounded Automaton (LBA) accepts the string in the similar manner as that of Turing machine does.
- For LBA halting means accepting. In LBA computation is restricted to an area bounded by length of the input. This is very much similar to programming environment where size of variable is bounded by its data type.

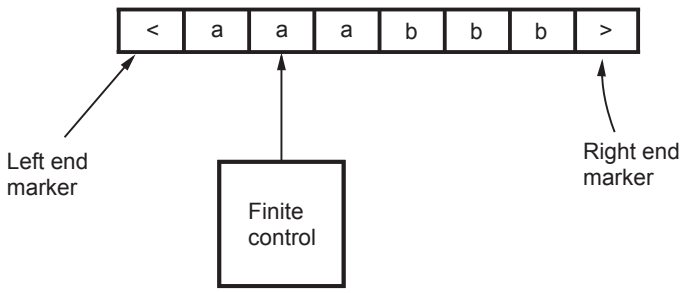


Fig. 5.13.1 Linear bound automaton

- The LBA is powerful than Non deterministic Push Down Automata(NPDA) but less powerful than Turing machine.
- The input is placed on the input tape with beginning and end markers.

In the given Fig. 5.13.1 the input is bounded by < and >.

A linear bounded automata can be **formally defined as** :

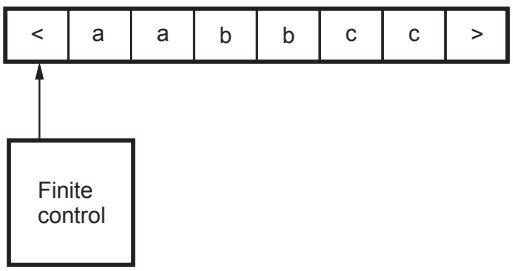
It is 7 - tuple non deterministic Turing machine with

$$M = \left(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}\right) \text{ having}$$

1. Two extra symbols of left end marker and right end marker which are not elements of Γ.
2. The input lies between these end markers.
3. The TM cannot replace < or > with anything else nor move the tape head left of < or right of >.

Example 5.13.1 *Construct a language* $L = \left\{a^n\ b^n\ c^n \mid n \geq 1\right\}$ *using LBA.*

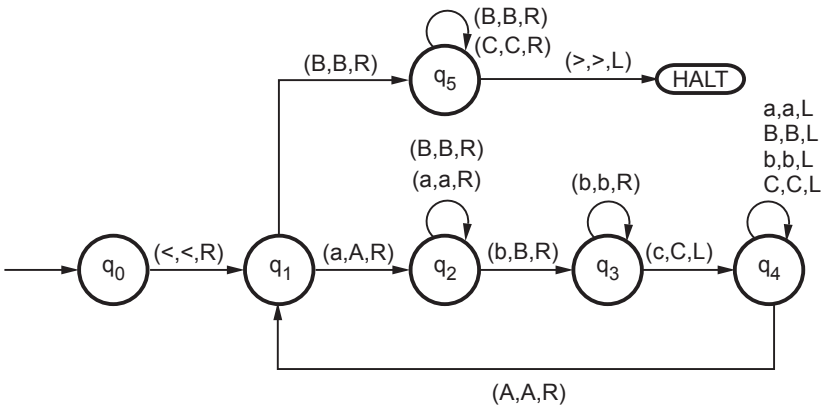
Solution :



Step 1 : The input is placed on the input tape which is enclosed within left end marker and right end marker.

Step 2 : We will apply the simple logic as : when we read 'a' convert it to A then move right by skipping all a's. On encountering first 'b' we will convert it to B. Then move right by skipping all b's. On receiving first c convert it to C. Move in left direction unless you get A. Repeat the above procedure and convert equal number of a's, b's, and c's to corresponding A's, B's and C's.- Finally move completely to the rightmost symbol if it is '>' a right end marker, then HALT. The machine will be -

Step 3 :



Simulation : Consider input **aabbcc**

< aabbcc >	Move right.
↑	
< aabbcc >	Convert to A, move right.
↑	
< Aabbcc >	Move right.
↑	
< Aabbcc >	Convert to B, move right.
↑	
< AaBbcc >	Move right.
↑	
< AaBbcc >	Convert to C, move left.
↑	
< AaBbCc >	Move left.
↑	
< AaBbCc >	Move left.
↑	
< AaBbCc >	Move left.
↑	

```
< AaBbCc >          Move right.
  ↑
< AaBbCc >          Convert to A, Move right.
    ↑
< AABbCc >          Move right.
      ↑
< AABbCc >          Convert to B, move right.
       ↑
< AABBCc >          Move right.
         ↑
< AABBCc >          Convert to C, move left.
          ↑
< AABBCC >         Move left continuously by skipping B's.
         ↑
< AABBCC >         Move right
    ↑
< AABBCC >         If we get B, we will move right to check whether
      ↑            all b's and c's are converted to B and C.
< AABBCC >         If we get right end marker '>' then we HALT by
            ↑      accepting the input aabbcc.
```

Thus in LBA the length of tape exactly equal to the input string and tape head can not move left of '<' and right of '>'.

Context Sensitive Languages (C.S. Languages)

The context sensitive languages are the languages which are accepted by linear bounded automata. These type of language 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 $A \rightarrow \varepsilon$ 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,
$L = \left\{ a^n \; b^n \; c^n | n \geq 1 \right\}$. 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 $S \rightarrow SABC$
SABC	rule $S \rightarrow aBC$
aBCABC	rule $CA \rightarrow AC$
aBACBC	rule $CB \rightarrow BC$
aBABCC	rule $BA \rightarrow AB$
aABBCC	rule $aA \rightarrow aa$
aaBBCC	rule $aB \rightarrow ab$
aabBCC	rule $bB \rightarrow bb$
aabbCC	rule $bC \rightarrow bc$
aabbcC	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.

Every contentext sensitive language can be represented by LBA.

5.14 Fill in the Blanks

Q.1 The language accepted by Turing machine is called ___ .

Q.2 Recursive is synonym to ___ .

Q.3 The complement of recursive language is ___ .

Q.4 NP stands for ____ .

Q.5 P stands for _____ .

Q.6 Searching the element is done in ____ time.

Q.7 Kruskal's algorithm is done in ___ time.

Q.8 ___ and___ is said to be NP complete.

Q.9 PCP stands for ____.

5.15 Multiple Choice Questions

Q.1 Which of the following problems is undecidable ?

a Membership problem for CFGs
b Ambiguity problem for CFGs
c Finiteness problem for FSAs
d Equivalence problem for FSAs

Q.2 A language L is said to be _____ if there is a turing machine M such that L(M) = L and M halts at every point.

a turing acceptable
b decidable
c undecidable
d none of the mentioned

Q.3 DFA = {M is a DFA that recognizes input w}. This language is ____ .

a decidable
b undecidable
c non finite
d none of these

Q.4 A problem is called _____ if its has an efficient algorithm for itself.

a tractable
b intractable
c computational
d none of the mentioned

Q.5 A PCP problem is solvable if ____ .

a |?|=1
b |?|=2
c |?|=4
d |?|=8

Q.6 Traveling salesman problem belong to _____.

a P
b NP
c linear
d none of these

Q.7 If PCP is decidable then MPCP is _____ .

a decidable
b undecidable
c can't say
d none

Answer Keys for Fill in the Blanks :

Q.1	recursive enumerable and recursive	**Q.2**	decidability
Q.3	recursive	**Q.4**	Nondeterministic Polynomial time
Q.5	Polynomial time	**Q.6**	polynomial time
Q.7	nonpolynomial time	**Q.8**	NP, P
Q.9	Post Correspondence Problem		

Answer Keys for Multiple Choice Questions :

Q.1	b	**Q.2**	b	**Q.3**	a	**Q.4**	a
Q.5	a	**Q.6**	b	**Q.7**	a		

❑❑❑

UNIT - VI

6 Computability and Complexity Theory

Syllabus

Computability Theory : *Decidable Problems and Un-decidable Problems, Church-Turing Thesis.*
Reducibility : *Undecidable Problems that is recursively enumerable, A Simple Un-decidable problem.*
Complexity Classes : *Time and Space Measures, The Class P, Examples of problems in P, The Class NP, Examples of problems in NP, P Problem Versus NP Problem, NP-completeness and NP-hard Problems.*

Contents

Part I : Computability Theory

6.1 Decidable Problems and Un-decidable Problems
. . . . . **May-06,10,15,16,**
. . . . . **Dec.-07,11,12,13** · · · · · · · Marks 12

6.2 Church-Turing Thesis

Part II : Reducibility

6.3 Undecidable Problems that is Recursively Enumerable
. . . . . **Dec.-05,07,08,16,17,**
. . . . . **May-08,09,10,18,19,** · · · · · · Marks 8

6.4 A Simple Un-decidable Problem. **May-07,12,14,18,**
. . . . . **Dec.-10,11,12,19,** · · · · · · · · Marks 8

Part III : Complexity Classes

6.5 Time and Space Measures
6.6 The Class P and NP . . . . . **Dec.-15,18,19,** · · · · · · · · · Marks 4
6.7 Examples of Problems in P . . . . . **Dec.-17,18,** · · · · · · · · · · · · Marks 8
6.8 Examples of Problems in NP . . . . . **May-16,18, Dec.-17,19,** · · · Marks 8
6.9 NP-completeness and NP-hard Problems . . . . **Dec.-16,** · · · · · · · · · · · · · · Marks 8
6.10 Polynomial Time Reduction . . . . . **May-08,16, Dec.-18,19,** · · · · Marks 8
6.11 Satisfiability Problems . . . . . **Dec.-16,18,** · · · · · · · · · · · · Marks 8
6.12 Restricted Satisfiability Problem

Part I : Computability Theory

6.1 Decidable Problems and Un-decidable Problems

SPPU : May-06,10,15,16, Dec.-07,11,12,13, Marks 12

- If a language is recursive then it is called decidable language and if the language is not recursive then such a language is called undecidable language.
- The class of decidable problems are also called as solvable problems and the class of undecidable problems is called unsolvable problems.

6.1.1 Decidability Problems Concerning Regular Languages

SPPU : May-15, 16

The decidability problems concerning regular languages can be understood with the help of following theorems.

Theorem 1 : Prove that following decision problems are recursive
(i) Two DFA's are equivalent or Not (ii) NFA Accepts a word or not

SPPU : May-15 End Sem, Marks 10, May-16, End Sem, Marks 5

Proof : (i)

- To prove that the given decision problem is recursive. We require a TM T(M) which simulates the DFA.
- Let, M_1 and M_2 are two DFAs. Consider the Turing machines $T(M_1)$ and $T(M_2)$ Obtain set of strings accepted by $T(M_1)$ and rejected by $T(M_2)$. That is

$$S_1 = T(M_1) \cap \overline{T(M_2)}.$$

- Similarly obtain the set of strings which $T(M_2)$ accepts and $T(M_1)$ rejects. That is

$$S_2 = T(M_2) \cap \overline{T(M_1)}.$$

- If these two sets are empty then it shows that $L(M_1)$ and $L(M_2)$ are equivalent and can be simulated by TM for the set of languages as.

$$L(M) = L(M_1) \cap \overline{L(M_2)} \cup \overline{L(M_1)} \cap L(M_2). \text{ The } L(M) = \phi.$$

- This shows that two DFA's are equivalent or not is a recursive problem.

(ii) Let, M be the NFA and input word w which decides if M accepts w. Convert this NFA to DFA. Then run the algorithm for DFA on Turing Machine T(M). The TM T(M) accepts w if and only if M reaches a final state on w. This shows that NFA accepts a word or not is recursive.

Theorem 2 : Prove that following decision problems is recursive : DFA Accepts a word or not.

SPPU : May-15, End Sem, Marks 5

Proof :

Let, M be the DFA and input word w which decides if M accepts w. Simulate an algorithm for DFA to run on Turing machine T(M). The TM T(M) accepts w if and only if M reaches a final state for w otherwise rejects. This show that DFA accepts a word or not is recursive.

6.1.2 Decidability Problems Concerning Context Free Languages

SPPU : May-06,10,15,16, Dec.-07,11,12,13

The undecidable problems about CFG are -

1) Whether or not two different context free languages define the same language ?

2) Whether given CFL is ambiguous or not ?

3) Whether complement of given CFL is context free language ?

4) Whether the intersection of two context free languages is context free ?

There is no algorithm to answer these questions. Hence these problems are known as **undecidable** problems.

Theorem 3 : Show that the following problem is undecidable. "Given two CFGs G_1 and G_2, is $L(G_1) \cap L(G_2) = \phi$".

Proof :

Let $L(G_1)$ and $L(G_2)$ be two CFLs. The CFL is not closed under intersection. That means $L(G_1) \cap L(G_2)$ does not necessarily generates CFL. This also implies that there is no algorithm which can tell whether $L(G_1) \cap L(G_2) = \phi$. Hence it is said that $(L(G_1) \cap L(G_2) = \phi$ is undecidable.

Theorem 4 : Prove that CFG G generates the string w or not.

SPPU : May-15, End Sem, Marks 5, May-16, End Sem, Marks 5

Proof :

Convert CFG G to G″ in chomsky normal form. Now the string $w \in L(G'')$ iff w can be derived in $2|w|-1$ steps where none of the intermediate string is of length more than $|w|$. If these derived step will derive w, the T(M) accepting L(G″) will accept otherwise rejects. This shows CFG G generates string w or not is recursive.

Theorem 5 : Let G be a CFG and r be a regular expression. Show that the problem 1) L(G) = L(r) 2) L(r) ∈ L(G) are undecidable.

SPPU : May-06, 10, Dec.-07, Marks 12, Dec.-11, Marks 8

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 6 : It is undecidable whether a CFG is ambiguous.

SPPU : Dec.-13, Marks 6

Proof :

Let $A = (x_1, x_2 \ldots x_k)$ and $B = (y_1, y_2, y_3 \ldots y_k)$ are two strings.

The G_1 and G_2 are two context free grammars. The production rules for these grammars are as follows -

$S_1 \rightarrow a_i S_1 x_i$ for $i = 1, 2, 3, \ldots k$

$S_1 \rightarrow a_i x_i$

$S_2 \rightarrow a_i S_2 y_i$ for $i = 1, 2, 3, \ldots k$

$S_2 \rightarrow a_i y_i$

The grammar G_1 generates the sentence of the form

$a_{in} \; a_{in-1} \ldots a_{i1} \, x_{i1} \, x_{i2} \ldots x_{in}$

The grammar G_2 generates the sentence of the form

$a_{in} \; a_{in-1} \ldots a_{i1} y_{i1} y_{i2} \ldots y_{in}$

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.

Review Question

1. *Explain with example decidable and non decidable language.*

SPPU : May-16, End Sem, Marks 6

6.2 Church-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).

Part II : Reducibility

6.3 Undecidable Problems that is Recursively Enumerable

SPPU : Dec.-05,07,08,16,17, May-08,09,10,18,19, Marks 8

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. 6.3.1 shows the relationship between these languages.

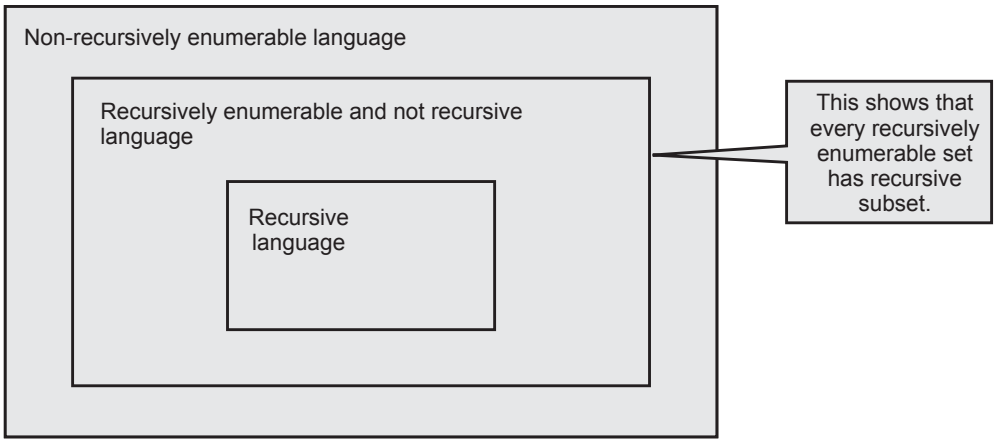


Fig. 6.3.1 Relationship between languages

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'.

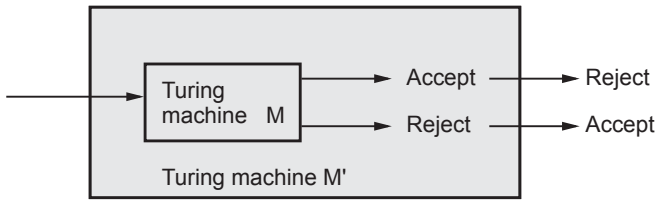


Fig. 6.3.2 Construction of M ' accepting L'

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'.

Theorem 2 : If a language L and its complement L' both are RE then L is a recursive language.

SPPU : Dec.-05, Marks 6

Proof : Consider a turing machine M made up of two turing machines M1 and M2. The machine M2 is complement of machine M1. We can also denote that L(M) = L(M1)

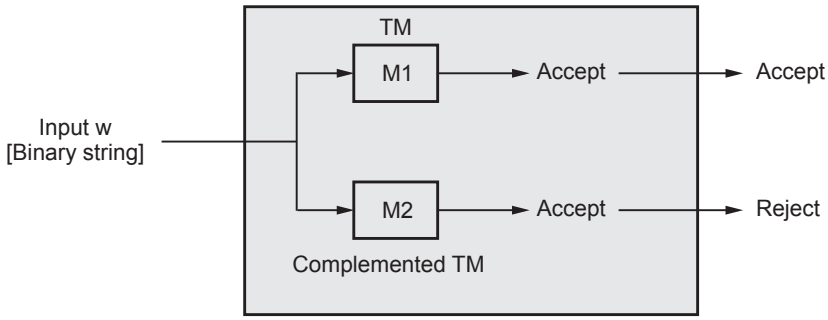


Fig. 6.3.3

and L(M2). Both M1 and M2 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 M1 and machine M2. The states of M consists of all the states of machine M1 and all the states of machine M2. The machine M made up of M1 and M2 is as shown in Fig. 6.3.3.

If the input w of language L is given to M then M1 will eventually accept and therefore M will accept L and halt. If w is not in L, then it is in L'. So M2 will accept and therefore M will half 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 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.

SPPU : May-08, 10, Marks 8, Dec.-16, End Sem, Marks 6

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_1 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. 6.3.4.

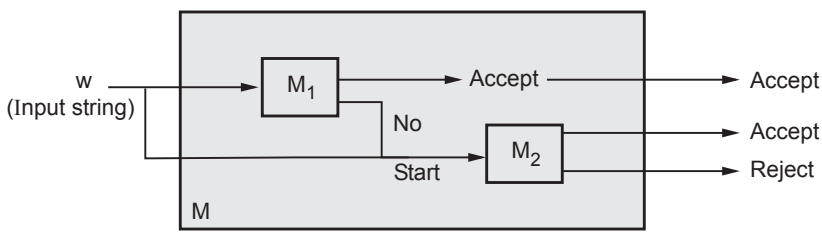
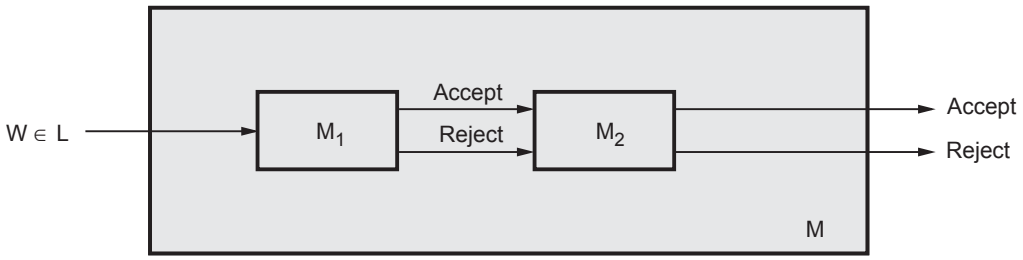


Fig. 6.3.4

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 $L = L_1 \cap L_2$.

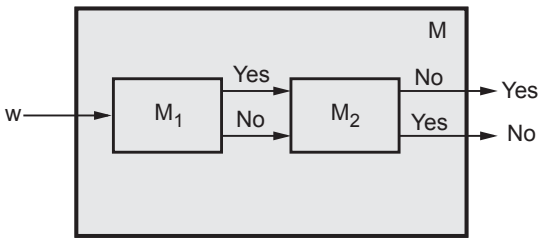


M_1 is a TM which accepts L_1 and M_2 is a TM which accepts L_2. The TM M is simulated which halts on accepting $w \in L$ such that $L = L_1 \cap L_2$. Hence intersection of two recursive languages is also recursive.

Theorem 4 : If L1 and L2 are two recursive languages and if L is defined as : L = {w| w is in L1 and not in L2 and not in L1}. Prove or disprove that L is recursive. **SPPU : Dec.-08, Marks 8**

Proof : Let L1 is a recursive language which is accepted by M1 and L2 is a recursive language which is accepted by M2. Languages L1 and L2 are such that if $w \in L1$ then $w \notin L2$. Similarly if $w \notin L1$ then $w \in L2$.

Find $L = L1 \cup L2$. 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 in uncountable. **SPPU : Dec.-07, May-09, 10, Marks 6**

Proof : If there is any language L which is recursively enumerable then there exists a TM which semidecides it (either accepts and halt 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.

Theorem 6 : Prove that i) AREX = $\{< R, W > | R$ is a regular expression that generates string w$\}$ is a decidable language.
ii) ECFG = $\{< G > | G$ is a CFG and L(G) = ϕ $\}$ is a decidable language.

SPPU : May-19, End Sem, Marks 8

Proof :

i)

- Let M be the DFA and input word w $\in$ R which is accepted by M.
- Simulate an algorithm for DFA to run on Turing Machine TM
- The TM accepts w if and only if M reaches a final state for w otherwise rejects.
- This shows that the regular expression generates string that denotes decidable language.

ii)

- We construct TM M to decide a problem.
- Scan input string <G>, determining whether the input constitutes a valid CFG. If not, then reject.
- Mark every terminal in every rule.
- For each rule A $\rightarrow$ w, if every symbol of w is marked, mark every occurence of A in rules.
- Continue until no new variables are marked.
- If S (Start symbol) is not marked then accept <G> otherwise reject.
- This shows that if G is a CFG then then L(G) = ϕ is decidable language.

Review Questions

1. *Define and explain recursive and recursively enumerable languages.*

   SPPU : Dec.-17, End Sem, Marks 8

2. *Write short note - Undecidability.* **SPPU : May-18, End Sem, Marks 4**

6.4 A Simple Un-decidable Problem

SPPU : May-07,12,14,18, Dec.-10,11,12,19, Marks 8

6.4.1 Post 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 $A = w_1, w_2, w_3, \ldots, w_n$ and $B = x_1, x_2, x_3 \ldots x_n$ then there exists a non empty set of integers $i_1, i_2, i_3, \ldots i_n$ such that, $w_1, w_2, w_3, \ldots w_n = x_1, x_2, x_3, \ldots x_n$"

To solve the post correspondence problem we try all the combinations of $i_1, i_2, i_3 \ldots, i_n$ to find the $w_i = x_i$ then we say that PCP has a solution.

Example 6.4.1 *Consider the correspondence system as given below -*

A = (1; 0; 010; 11) and B = (10; 10; 01; 1). The input set is

Σ = {0, 1}. Find the solution.

Solution : A solution is 1; 2; 1; 3; 3; 4. That means $w_1w_2w_1w_3w_3w_4 = x_1x_2x_1x_3x_3x_4$.

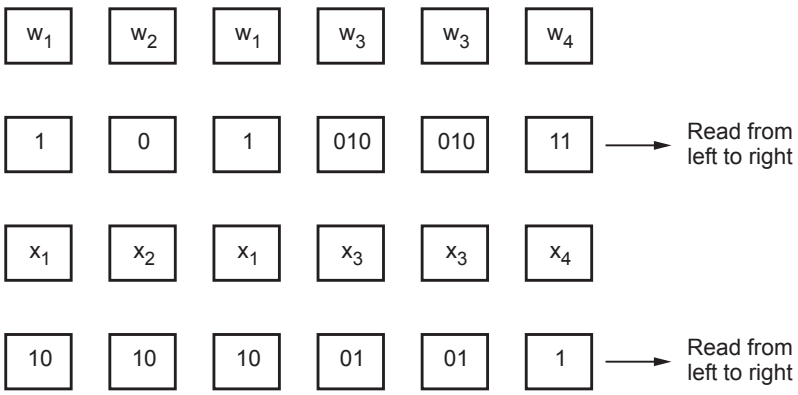


Fig. 6.4.1

The constructed string from both lists is 10101001011.

Example 6.4.2 *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 $w_1w_5w_2w_3w_4w_4w_3w_4 = x_1x_5x_2x_3x_4x_4w_3w_4$. This solution gives a unique string bababababaaaabaa = bababababaaaabaa. Hence solution is 15234434.

Example 6.4.3 *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

$$\begin{aligned} A1A3A1A1A1A3A2A2 &= 100 + 1 + 100 + 100 + 1 + 0 + 0 \\ &= 1001100100100 \\ B1B3B1B1B3B2B2 &= 1 + 00 + 1 + 1 + 00 + 100 + 100 \\ &= 1001100100100 \end{aligned}$$

Example 6.4.4 *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 bababab 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 6.4.5 *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

2	1	1	3		2	1	1	3
b*a*b^3	b	b	b*a*	=	b*a*	b^3	b^3	*a*

which forms b*a*b^{3}b^3*a*. Thus PCP has a solution.

Review Questions

1. *What is post correspondence problem ? Explain with example.* **SPPU : Dec.-10, Marks 8**
2. *Write short note on - Post correspondence problem.*

SPPU : May-07, 12, 14, Dec.-12, Marks 4, Dec.-11, Marks 8, May-18, End Sem, Marks 4, Dec.-19, End Sem, Marks 4

Part III : Complexity Classes

6.5 Time and Space Measures

- The efficiency of an algorithm can be decided by measuring the performance of an algorithm. We can measure the performance of an algorithm by computing two factors.

  1. Amount of **time** required by an algorithm to execute.

2. Amount of **storage** required by an algorithm.

Hence we define two terms - Time Complexity and Space Complexity

- **Time complexity** of an algorithm means the amount of time taken by an algorithm to run. By computing time complexity we come to know whether the algorithm is slow or fast.
- **Space complexity** of an algorithm means the amount of space (memory) taken by an algorithm. By computing space complexity we can analyze whether an algorithm requires more or less space.
- To choose the best algorithm, we need to check efficiency of each algorithm. The efficiency can be measured by computing time complexity of each algorithm. Asymptotic notation is a shorthand way to represent the time complexity.
- Using asymptotic notations we can give time complexity as "fastest possible", "slowest possible" or "average time".
- Various notations such as Ω, Θ and O used are called **asymptotic notions**.

Big Oh Notation

The **Big oh** notation is denoted by 'O'. It is a method of representing the **upper bound of** algorithm's running time. Using big oh notation we can give longest amount of time taken by the algorithm to complete.

Definition

Let F(n) and g(n) be two non-negative functions.

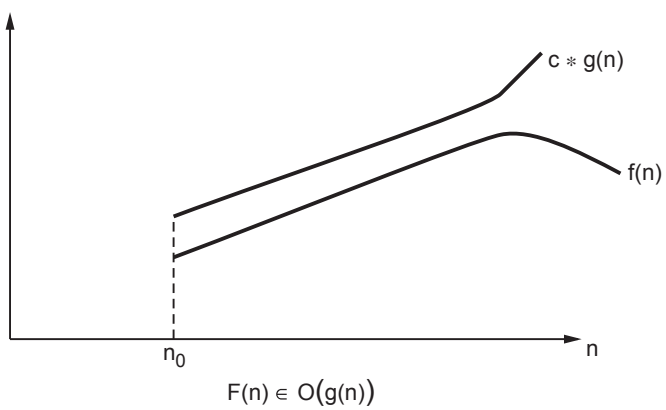


Fig. 6.5.1

Let n_0 and constant c are two integers such that n_0 denotes some value of input and $n > n_0$. Similarly c is some constant such that c > 0. We can write

$$f(n) \leq c * g(n)$$

then f(n) is big oh of g(n). It is also denoted as $f(n) \in O(g(n))$. In other words f(n) is less than g(n) if g(n) is multiple of some constant c.

Example 6.5.1 *Consider function $f(n) = 2n + 2$ and $g(n) = n^2$. Then we have to find some constant c, so that $f(n) \leq c * g(n)$. As $F(n) = 2n + 2$ and $g(n) = n^2$ then we find c for $n = 1$ then*

Solution :

$$f(n) = 2n + 2 = 2(1) + 2$$

$$f(n) = 4$$

and $$g(n) = n^2 = (1)^2$$

$$g(n) = 1$$

i.e. $$f(n) > g(n)$$

If n = 2 then,

$$f(n) = 2(2) + 2 = 6$$

$$g(n) = (2)^2$$

$$g(n) = 4$$

i.e. $$f(n) > g(n)$$

If n = 3 then,

$$f(n) = 2(3) + 2 = 8$$

$$g(n) = (3)^2$$

$$g(n) = 9$$

i.e. $f(n) < g(n)$ is true.

Hence we can conclude that for n > 2, we obtain

$$f(n) < g(n)$$

Thus always upper bound of existing time is obtained by big oh notation.

6.5.1 Order of Growth

Measuring the performance of an algorithm in relation with the input size n is called order of growth. For example, the order of growth for varying input size of n is as given below.

n	log n	n log n	n^2	2^n
1	0	0	1	2
2	1	2	4	4
4	2	8	16	16
8	3	24	64	256
16	4	64	256	65,536
32	5	160	1024	4,294,967,296

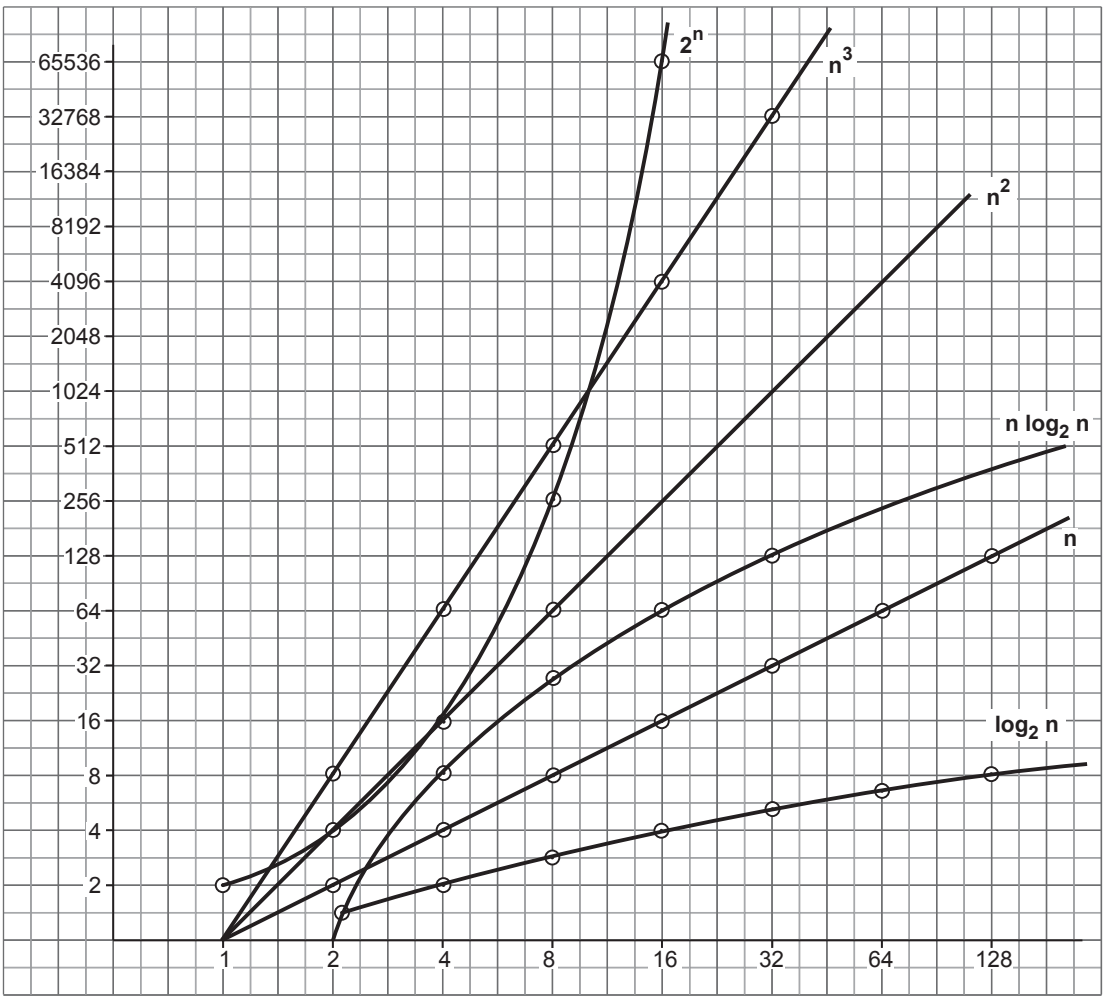


Fig. 6.5.2 Rate of growth of common computing time function

From the above drawn graph it is clear that the logarithmic function is the slowest growing function. And the exponential function 2^n is fastest and grows rapidly with varying input size n. The exponential function gives huge values even for small input n. For instance : For the value of n = 16 we get 2^{16} = 65536.

6.6 The Class P and NP

SPPU : Dec.-15,18,19, Marks 4

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(logn), sorting of elements O(logn).

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 polnomial.
  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.

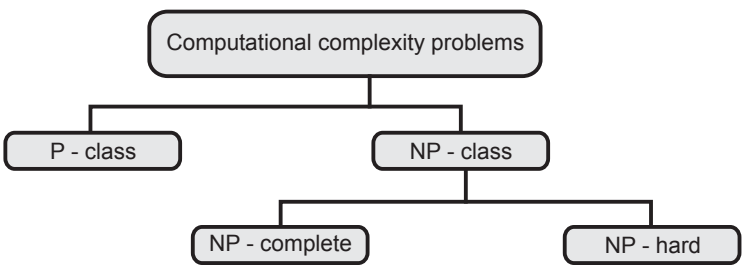


Fig. 6.6.1

- 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.

6.6.1 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.

Review Questions

1. *Explain with example - computational complexity.* **SPPU : Dec.-15, End Sem, Marks 2**
2. *Differentiate between P class and NP class problems.* **SPPU : Dec.-15, End Sem, Dec.-18,19, End Sem, Marks 4**

6.7 Examples of Problems in P

SPPU : Dec.-17,18, Marks 8

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 : Find the minimum spanning tree for the following figure using Kruskal's algorithm.

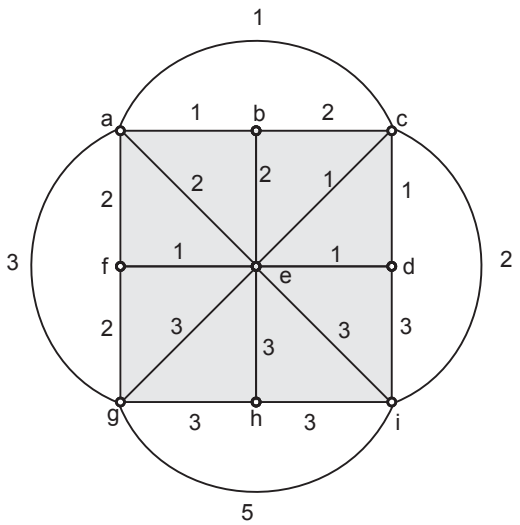


Fig. 6.7.1

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.

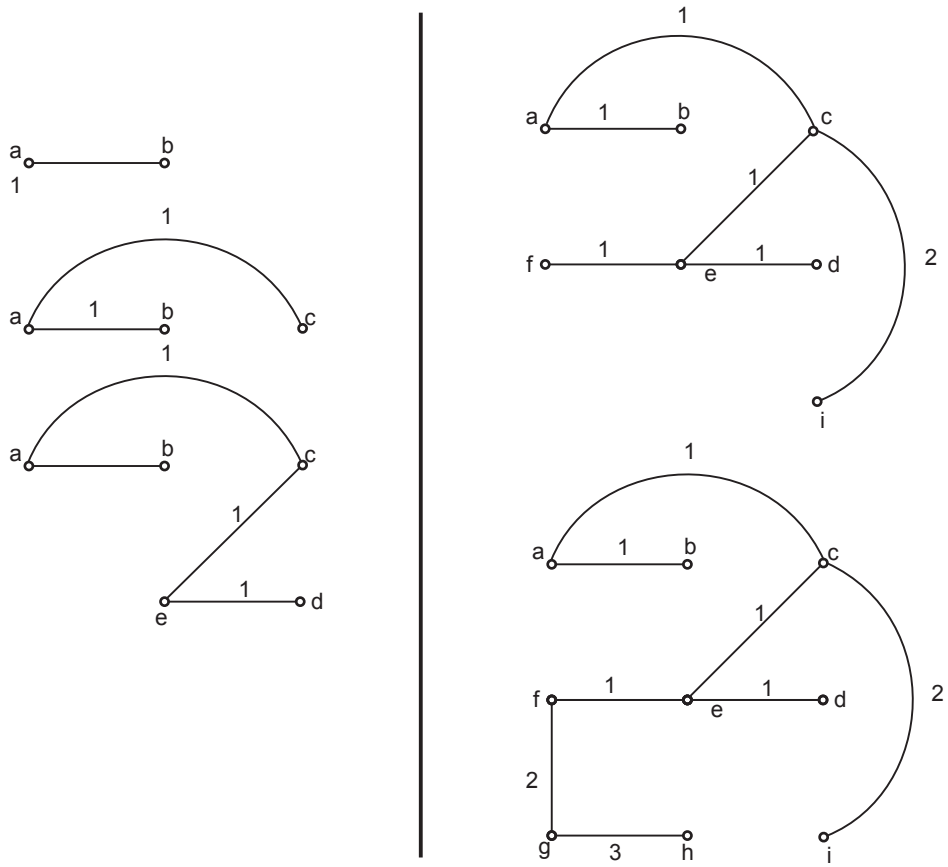


Fig. 6.7.2

Review Question

1. *What is Kruskal's algorithm ? How can we solve this problem using Turing machine ?*

SPPU : Dec.-17, 18, End Sem, Marks 8

6.8 Examples of Problems in NP

SPPU : May-16,18, Dec.-17,19, Marks 8

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 :

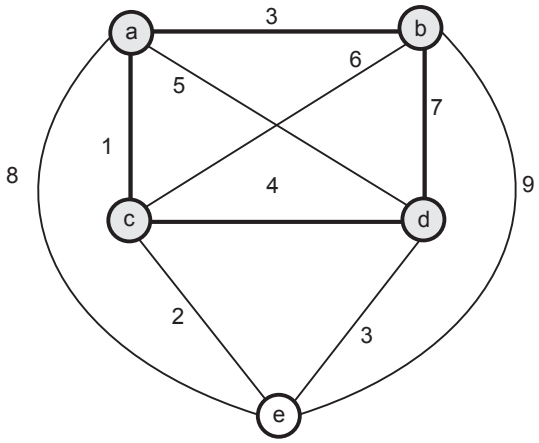


Fig. 6.8.1

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 Question

1. *What do you mean by NP problems ? Justify why the Traveling Salesman problem is a NP problem ?* **SPPU : May-16, End Sem, May-18, End Sem, Dec.-17, 19, Marks 8**

6.9 NP-completeness and NP-hard Problems

SPPU : Dec.-16, Marks 8

- As we know, P denotes the class of all deterministic polynomial language problems and NP denotes the class of all non-deterministic polynomial language problems. Hence

  $$P \subseteq NP.$$

- The question of whether or not

  $$P = NP$$

  holds, is the most famous outstanding problem in the computer science.
- Problems which are known to lie in P are often called as *tractable*. Problems which lie outside of P are often termed as *intractable*. Thus, the question of whether P = NP or P ≠ NP is the same as that of asking whether there exist problems in NP which are intractable or not.

The relationship between P and NP is depicted by Fig. 6.9.1.

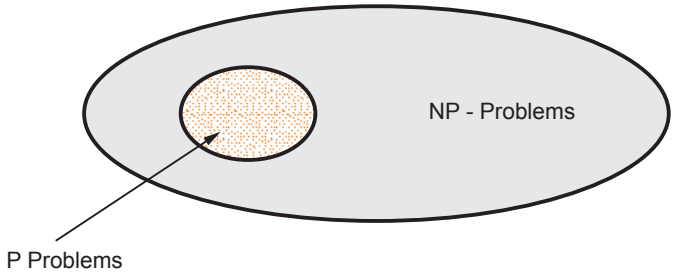


Fig. 6.9.1 P and NP problems assuming P ≠ NP

- We don't know if P = NP. However in 1971, S. A. Cook proved that a particular NP problem known as SAT (Satisfiability of sets of Boolean clauses) has the property that, if it is solvable in polynomial time, so are all NP problems. This is called a **"NP-complete"** problem.
- Let L_1 and L_2 are two problems then problem L_1 reduces to L_2 if and only if there is a way to solve L_1 by deterministic polynomial time algorithm using a deterministic algorithm that solves L_2 in polynomial time.
  L_1 reduces to L_2 can be denoted as $L_1 \propto L_2$. In other words we can say that if there exists any polynomial time algorithm which solves L_2 then we can solve L_1 in polynomial time. We can also state that if $L_1 \propto L_2$ and $L_2 \propto L_3$ then $L_1 \propto L_3$.
- A NP problem such that, if it is in P, then NP = P. If a (not necessarily NP) problem has this same property then it is called **"NP-hard"**. Thus the class of NP-complete problem is the intersection of the NP and NP-hard classes.
- Normally the decision problems are NP-complete but optimization problems are NP-hard. However if problem L_1 is a decision problem and L_2 is optimization problem then it is possible that $L_1 \propto L_2$. For instance the Knapsack decision problem can be Knapsack optimization problem.

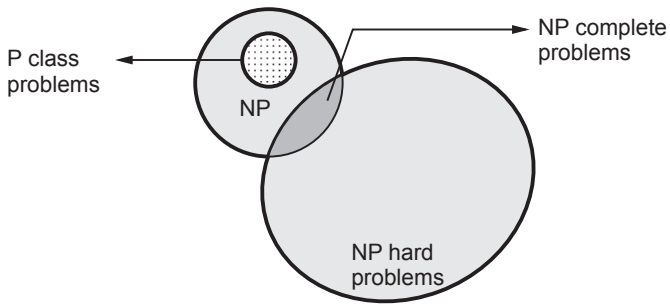


Fig. 6.9.2 Relationship between P, NP, NP-complete and NP-hard problems

- There are some NP-hard problems that are not NP-complete. For example *halting problem.* The halting problem states that : "Is it possible to determine whether an algorithm will ever halt or enter in a loop on certain input ? "

Two problems P and Q are said to be polynomially equivalent if and only if $P \propto Q$ and $Q \propto P$.

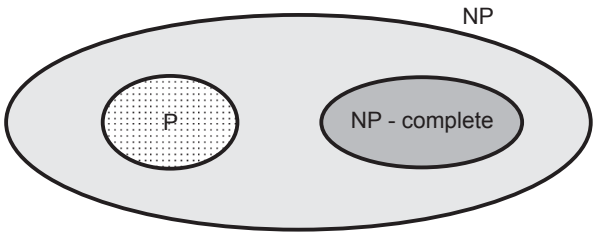


Fig. 6.9.3

Review Question

1. *What do you mean by NP complete problem ? List all the problems in this class and explain any one with suitable example.* **SPPU : Dec.-16, End Sem, Marks 8**

6.10 Polynomial Time Reduction

SPPU : May-08,16, Dec.-18,19, Marks 8

To prove whether particular problem is NP complete or not we use polynomial time reducibility. That means if

$A \xrightarrow{\text{Poly}} B$ and $B \xrightarrow{\text{Poly}} C$ then $A \xrightarrow{\text{Poly}} C$.

The **reduction** is an important task in NP completeness proofs. This can be illustrated by Fig. 6.10.1. (See Fig. 6.10.1 on next page.)

Various types of reductions are :

Local replacement - In this reduction A→B by dividing input to A in the form of components and then these components can be converted to components of B.

Component design - In this reduction A→B by building special component for input B that enforce properties required by A.

The reduction can be denoted as $A \leq_T^P B$.

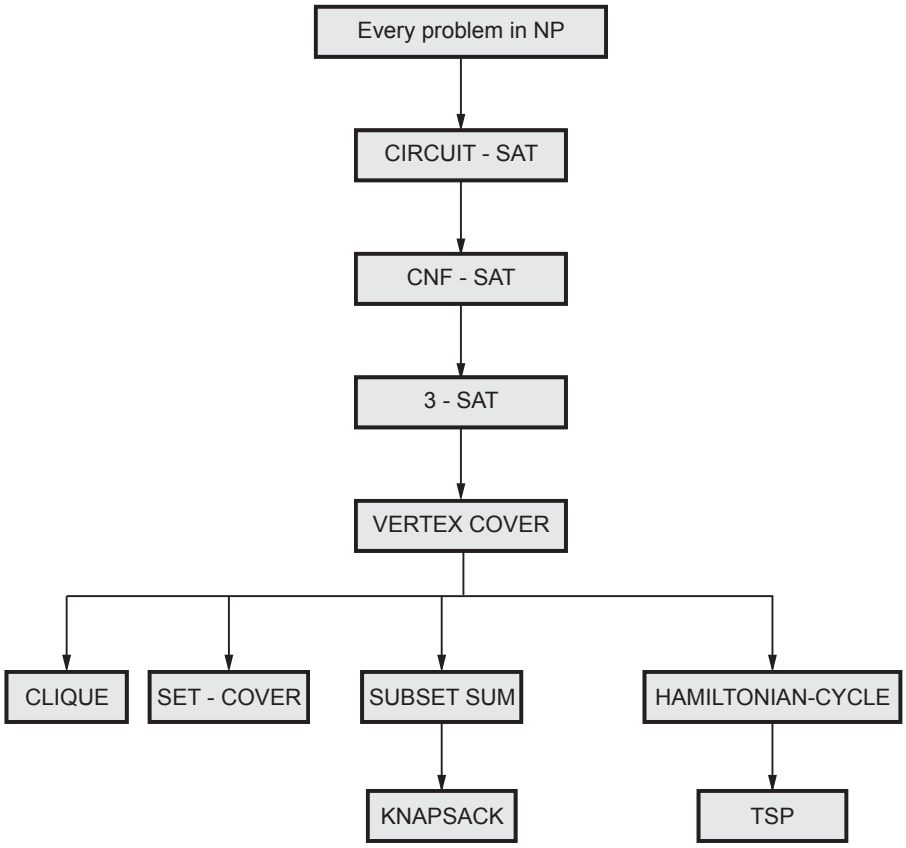


Fig. 6.10.1 Reduction in NP completeness

6.10.1 Proving NP Problems

1. HAMILTONIAN CYCLE : This is a problem in which graph G is accepted as input and it is asked to find a simple cycle in G that visits each vertex of G exactly once and returns to its starting vertex. Such a cycle is called an **Hamiltonian cycle**.

Theorem : HAMILTONIAN CYCLE is in NP.

Proof : Let A be some non-deterministic algorithm to which graph G is given as input. The vertices of graph are numbered from 1 to N. We have to call the algorithm recursively in order to get the sequence S. This sequence will have all the vertices without getting repeated. The vertex from which the sequence starts must be ended at the end. This check on the sequence S must be made in polynomial time n. Now if there is a Hamiltonian cycle in the graph then algorithm will output "yes". Similarly if we get output of algorithm as "yes" then we could guess the cycle in G with every vertex appearing exactly once and the first visited vertex getting visited at the last. That means

A non-deterministically accepts the language HAMILTONIAN CYCLE. It is therefore proved that HAMILTONIAN CYCLE is in NP.

2. CIRCUIT-SAT : This is a problem in which a Boolean circuit is taken as input with single output node. And then finds whether there is an assignment of values to the circuit's input so that we get its output value as "1". This assignment of values is called **satisfying assignment.**

Theorem : CIRCUIT-SAT is in NP.

Proof :

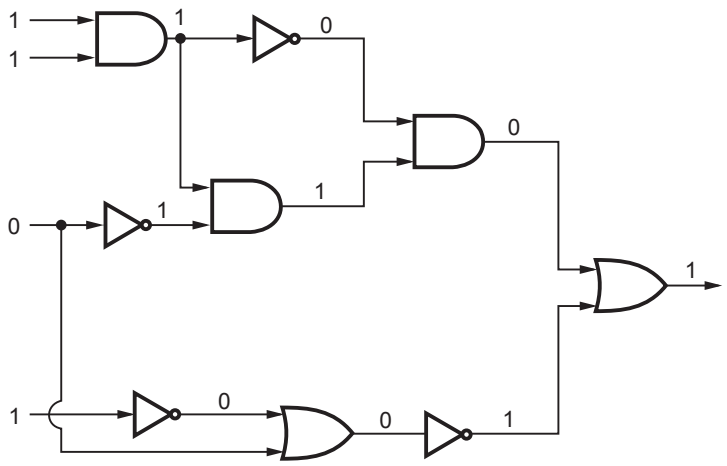


Fig. 6.10.2

The logic gates that are used are

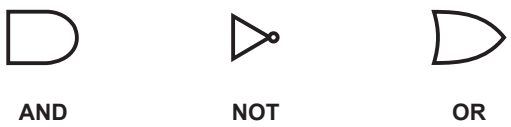


Fig. 6.10.3

Let us construct a non-deterministic algorithm. Which works in polynomial time. Then we should have **choose** ( ) method which can guess the values of input node as well as the output, then the algorithm says "no" if we get output of Boolean circuit as 0. Similarly the algorithm says "yes" if we get output of Boolean circuit as 1.

Now from the output of algorithm we can guess the inputs to logic gates in the circuit. If the algorithm has output "yes" then we can say that Boolean circuit has satisfying input values. Thus we can say that CIRCUIT-SAT is in NP.

Theorem : Prove that a directed Hamiltonian cycle ∝ the travelling salesman decision problem.

SPPU : May-08, Marks 6

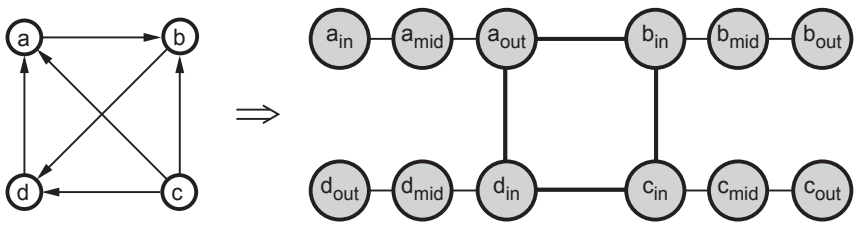
Proof

The directed Hamiltonian cycle is a cycle in which each vertex of a directed graph is visited only once. To prove that directed Hamiltonian cycle (DHC) ∝ Travelling Salesperson Problem (TSP) we will apply following steps -

If there are two problems A and B and if we wish to prove that B is NP complete then -

1) Pick up a known NP complete problem A.
2) Reduce A to B. This can be done in following steps.
   a) Describe a transformation that maps the instance of A to instance of B in such a manner that "yes" for B = "yes" for A.
   b) Prove that this transformation runs in polynomial time.
3) This proves that problem B is also NP complete.

To prove that DHC ∝ TSP we will reduce Directed Hamiltonian cycle to undirected Hamiltonian Cycle. The conversion is as follows -



In this undirected graph if there exists a Hamiltonian cycle, then that denotes visiting cities in the order of nodes being visited in Hamiltonian cycle.

If no Hamiltonian cycle exists that means TSP has no solution. The Hamiltonian cycle problem is NP complete. This proves that TSP is also NP complete.

Review Question

1. *What do you mean by polynomial time reduction ? Explain with suitable example.*

SPPU : May-16, End Sem, Marks 8, Dec.-18, 19, End Sem, Marks 6

6.11 Satisfiability Problems

SPPU : Dec.-16,18, Marks 8

6.11.1 The Satisfiability Problems

The boolean expression are created using -

1) The variables whose value can be 0 or 1.
2) The operators that are used in expression can be $\vee$, $\wedge$. The $\vee$ means OR and $\wedge$ means AND, operators.
3) The unary operator $\neg$ stands for negation.
4) The parenthesis are used to group the operands and operators in the expression. The highest precedence is to $\neg$, then $\wedge$, then $\vee$.

For example :

The boolean expression is $(x \wedge y) \vee (y \wedge z)$. If $y \wedge z$ and $x \wedge y$ both are true then the boolean expression is true if $(x \wedge y)$ and $(y \wedge z)$ both are false the boolean expression is false.

- The study of boolean functions generally concerned with set of **truth assignments** (i.e. assignments of 0 or 1 to each variable). Due to this assignment the function becomes true.
- SAT problem : Does there exist a truth assignment making function true ?
- $(x \wedge y \wedge z)$ is satifiable when x = 1, y = 1 and z = 1.

  But $x \wedge (\neg y)$ is not satisfiable.

6.11.2 Representing SAT Instances

- An instance of SAT is a boolean function.
- The symbols $\wedge$, $\vee$, $\neg$, (and) are used in these instances.
- The variable x_i is represented by symbol x followed by 0's and 1's. For example x_2 is represented as x_{10}. (Here 2 is replaced by binary value 10).
- **Example :** $(x_1 \vee x_2) \wedge (x_2 \vee \neg x_3)$ is represented as $(x_1 \vee x_{10}) \wedge (x_{10} \vee \neg x_{11})$

6.11.3 NP Completeness of SAT Problem

In 1971, **Stephen Cook** proved one popular theorem which is popularly known as Cook's theorem. This theorem refers a specific problem as is P = NP ? That means if we find a polynomial time algorithm for **satisfiability** then P is definitely equal to NP and then we can solve all the NP problems in polynomial time, no matter how different they might be from the **satisfiability**. He states the theorem in following manner.

Theorem : The satisfiability problem is NP complete.

Proof :

- **Deterministic algorithms** are those algorithms which **accept** or **reject** the strings. **Non-deterministic algorithms** are those algorithms which generally **accept** the strings but generally **do not reject** strings.
- A problem X is an element of NP if there is a non-deterministic polynomial time algorithm that solves X.
- All the algorithms can be represented by Turing Machine TM - M.
- Hence, when we say $X \in NP$, then that means there exists a non-deterministic turing machine M_{NP} which accepts any element of problem X in polynomial time.
- By Cook's theorem we have to show that there exists some boolean expression which is satisfiable (i.e. the expression evaluates to TRUE). Note that the Boolean expression must be in **Conjuctive Normal Form** (CNF). The Boolean expression must be created for the element $x \in X$.
- Let, M_{NP} will be a turing machine and a polynomial time bound can be denoted by **p(n)**.
- The turing machine will have a state diagram and input tape. The tape consists of cells arranged adjacently. It will be a two way tape which is infinitely long in both the directions.
- Σ is the set of input symbols that can be placed on the input tape where each cell consists of single letter.
- Γ is set symbols which is used by TM while processing the input. The empty cells are occupied by blank symbols (denoted by Δ).
- Q is a set of states. Along with other states Q consists of three type of states.

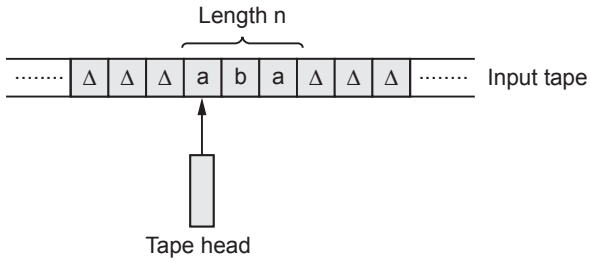
  q_0 - Initial state

  q_{ACCEPT} - It is accept state which represents that input string is accepted.

  q_{REJECT} - It is a reject state which represents that input string is not accepted.

When the turing machine M_{NP} is in state q_{ACCEPT} or q_{REJECT} then it gets halted and there is no further transition from this state.

- The δ is a transition function which determines the move of the Turing Machine M_{NP}.
  There will be two possible directions for each move L or R.



- Each element will be accessed within **unit of time**, and then the transition is made. To accept x the M_{NP} must enter is q_{ACCEPT} state within p(n) polynomial time but it is not necessary for M_{NP} to enter in q_{REJECT} state to reject x.
- Each string fed to machine M_{NP} must be **modeled in CNF form.** Hence it is necessary to define two things - 1) Number of **variables** representing conditions in M_{NP} and 2) **Clauses** to represent legal accepting string for M_{NP}. Let us get introduced with the variables.
- **Q [i, q_m]** where i runs from 0 to p(n) and q_m runs through all states of M_{NP}. The Q [i, q_m] represents that at time i, M_{NP} is in state q_m. **H[i, j]** where i runs from 0 to p(n) and j runs from – p(n) to p(n) + 1. This variable H[i, j] represents that at time i, the Turing machine M_{NP} will scan the cell j on the input tape. This will be performed in p(n) transitions. **S [i, j, s_m]** where i runs from 0 to p(n), j runs from – p(n) to p(n) + 1 and s_m runs through all the tape symbols. This variable S [i, j, s_m] represents the content s_m of jth cell at time i.
- There will be six clauses which can make accepting sequence of M_{NP}. These are -

  C - This clause represents that at each time i, M_{NP} will be in one and only one **state**.

  D - This clause represents that at each time i, M_{NP} will be **scanning** one and only one cell of the tape.

  E - This clause represents that at each time i there will be only one **symbol** in each cell, of the input tape.

  F - This clause states that machine (i.e. TM) will be in q_0 state initially with properly **positioned read-write head** for input element x.

  G - This clause represents that by time p(n), M_{NP} will enter in q_{ACCEPT} **state**.

  H - This clause represents that all the **transitions** can be correctly applied.

Now we will represent each clause as a set **for clause C** -

i) Add a clause $\{Q[i,q_1], Q[i,q_2], \ldots, Q[i,q_k]\}$ where i is the time and $q_1, q_2, \ldots q_k$ represents number of states $\in$ Q.

ii) Add a clause $\{\overline{Q[i,q_r]}, \overline{Q[i,q_t]}\}$ where q_r and q_t represent paired states.

For each unit of time i.

The i) Case represents that M_{NP} will be at least in one state at any time i.

The ii) Case represents that the M_{NP} will not be in more than one state at any time i. The time i will be from 0 to p(n).

- **For clause D -**

  i) Add a clause $\{H[i,-p(n)], H[i,-p(n)+1], \ldots H[i,p(n)+1]\}$

  ii) Add a clause $\{\overline{H[i,r]}, \overline{H[i,t]}\}$ where r and t represents pair of tape cells.

  The i) Case represents that at every time i M_{NP} will be scanning at least one cell. The ii) Case represents that at every time i M_{NP} will not scan more than one cell.

- **For clause E -**

  Add a clause that any time i each cell of tape contains at least one symbol and add a clause that guarantees that at any time i no cell contains more than one tape symbol.

- **For clause F -**

  i) Add a clause Q [0, q_0] i.e. M_{NP} is in start state.

  ii) Add a clause H [0, 1] i.e. The tape head is positioned at cell 1.

  ii) Add a clause such that input string is on tape at time 0.

  iv) Add a set of clauses that guarantees that at time 0 rest of the tape cells contain blank characters.

- **For clause G -**

  i) Add a clause $\{Q[p(n), q_{ACCEPT}]\}$ i.e. M_{NP} has entered in Accept state at any polynomial time p(n). This causes M_{NP} to accept the input string and there is no further transition from q_{ACCEPT}.

- **For clause H -**

  Add a clause for mapping function δ where $(q_{curr}, s_{curr}, q_{next}, s_{next}, dir)$ be the element of δ.

  This statement will be modeled in CNF form. That means if at time i, M_{NP} is in q_{curr} and s_{curr} symbol is read then move to q_{next} state and tape may contain s_{next} symbol. This move will be based on **dir** which can be L or R.

- The final expression will be

$$EXP = C \cup D \cup E \cup F \cup G \cup H$$

EXP is satisfiable if and only if M_{NP} accepts $x = x_1\ x_2\ x_3 \ldots x_n$. This expression can be constructed from turing machine M_{NP} in polynomial time. This proves the theorem.

Review Questions

1. *What is SAT problem ? Explain in detail.* **SPPU : Dec.-16, End Sem, Marks 8**
2. *Explain the satisfiability problem with an example.* **SPPU : Dec.-18, End Sem, Marks 4**
3. *Prove that the satisfiability problem is NP-complete.* **SPPU : Dec.-18, End Sem, Marks 6**

6.12 Restricted Satisfiability Problem

The intermediate form of SAT problem is **3-SAT** problem. It is much easier to use 3-SAT instead of SAT. 3-SAT is still a problem of satisfiability of boolean expression.

Defintion : A 3SAT problem is a problem which takes a Boolean formula S in CNF form with each clause having **exactly three literals** and check whether S is satisfied or not. [Note that CNF means each literal is ORed to form a clause, and each clause is ANDed to form Boolean formula S].

Following formula is an instance of 3-SAT problem :

$(\bar{a} + b + \bar{g})\ (c + \bar{e} + f)\ (\bar{b} + d + \bar{f})\ (a + e + \bar{h})$

6.12.1 Normal Forms for Boolean Expression

A Boolean formula is in **Conjuctive Normal Form (CNF)** if it is formed as collection of subexpressions. These subexpressions are called **clauses**.

For example

$(\bar{a} + b + d + \bar{g})\ (c + \bar{e})\ (\bar{b} + d + \bar{f} + h)\ (a + c + e + \bar{h})$

This formula evaluates to 1 if b, c, d are 1.

The CNF-SAT is a problem which takes Boolean formula in CNF form and checks whether any assignment is there to Boolean values so that formula evaluates to 1.

6.12.2 Converting Expressions to CNF

- There are two parts of this conversion -
  1) Push all ¬'s down the expression so that only negations are of variables, the boolean expression becomes AND and OR of literals.

2) Write Boolean expression in CNF form, we will get new variables.

- **DeMorgan's Law**

1) $\neg (A \wedge B) = \neg (A) \vee \neg (B)$

   Pushing $\neg$ below and converting $\wedge$ to $\vee$.

2) $\neg (A \vee B) = \neg (A) \wedge \neg (B)$

   Pushing $\neg$ below and converting $\vee$ to $\wedge$.

3) $\neg (\neg A) = \wedge$ i.e. double negation.

For example

$\neg ((\neg x + y) (x + y))$

$$= (\neg (\neg x + y) + \neg (x + y))$$

$$= x \neg y + \neg (x + y)$$

$$= x \neg y + (\neg x + \neg y)$$

$$= x \neg y \neg (x + y)$$

Theorem : CNF-SAT is in NP complete.

Proof : Let S be the Boolean formula for which we can construct a simple non-deterministic algorithm which can guess the values of variables in Boolean formula and then evaluates each clause of S. If all the clauses evaluate S to 1 then S is satisfied. Thus CNF-SAT is in NP-complete.

Theorem : 3SAT is in NP complete.

Proof : Let S be the Boolean formula having 3 literals in each clause for which we can construct a simple non-deterministic algorithm which can guess an assignment of Boolean values to S. If the S is evaluated as 1 then S is satisfied. Thus we can prove that 3SAT is in NP-complete.

❑❑❑

Notes

SOLVED MODEL QUESTION PAPER (In Sem)
Theory of Computation
T.E. (Computer) Semester - V (As Per 2019 Pattern)

Time : 1 Hour] **[Maximum Marks : 30**

N. B. :

i) Attempt Q.1 or Q.2, Q.3 or Q.4.

ii) Neat diagrams must be drawn wherever necessary.

ii) Figures to the right side indicate full marks.

iv) Assume suitable data, if necessary.

Q.1 a) *Define i) Symbol ii) Alphabet.* **(Refer section 1.2)** **[4]**

b) *Design FA to accept the string that always ends with 00.* **(Refer example 1.9.7)** **[3]**

c) *Design DFA for a language of string 0 and 1 that*
I) Ending with 10 II) Ending with 11 III) Ending with 1.
(Refer example 1.9.20) **[8]**

OR

Q.2 a) *Convert the following NFA with ε-moves into NFA without ε moves.*
(Refer example 1.12.3) **[5]**

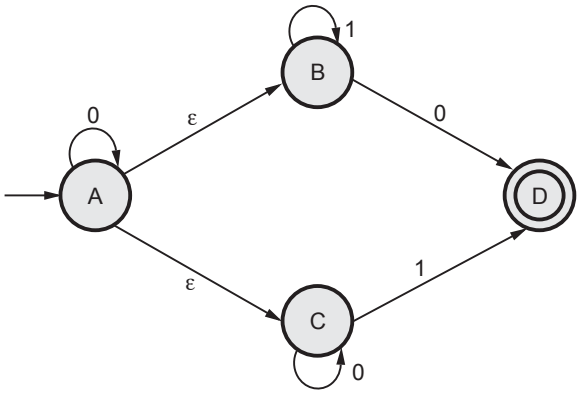


Fig. 1

b) *Give difference between NFA and DFA* **(Refer section 1.10)** **[4]**

c) *Construct the minimum state automaton for the following transition diagram.* **(Refer example 1.15.1)** **[6]**

Q.3 a) *Let L be a language. It is clear from the definition that* $L^+ \subseteq L^*$. *Under what circumstances are they equal ?* **(Refer example 2.2.22)** **[4]**

b) *What are algebraic laws of regular expression* **(Refer section 2.4)** **[3]**

c) *Use pumping lemma to show whether or not the following languages are regular :*
i) $A_1 = \{www \,/\, w \in (a, b)^*\}$ *ii)* $A_2 = \{a^{2n} \,/\, n \geq 0\}$. **(Refer example 2.8.6)** **[8]**

OR

Q.4 a) *State and prove any two closure properties of regular languages* **(Refer section 2.9)** **[5]**

b) *State and explain pumping lemma for regular sets* **(Refer section 2.8)** **[3]**

c) *Show that :* *i)* $(a.b)^* \neq a^* \cdot b^*$ *ii)* $(a+b)^* = (a+b)^* \cdot (a+b)^*$
(Refer example 2.6.14) **[7]**

SOLVED MODEL QUESTION PAPER (End Sem)
Theory of Computation
T.E. (Computer) Semester - V (As Per 2019 Pattern)

Time : 2 $\frac{1}{2}$ Hours] **[Maximum Marks : 70**

N. B. :

i) Attempt Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8.
ii) Neat diagrams must be drawn wherever necessary.
ii) Figures to the right side indicate full marks.
iv) Assume suitable data, if necessary.

Q.1 a) *Convert the given CFG to CNF.*
Consider $G = (V, T, P, S)$
Where $V = \{S, A, B\}$
$T = \{a, b\}$
P consists of

$S \to aB$ $A \to bAA$
$S \to bA$ $B \to b$
$A \to a$ $B \to bS$
$A \to aS$ $B \to aBB$ **(Refer example 3.9.4)** **[10]**

b) *Construct the production rules for defining a language* $L = \left\{a^x b^y \mid x \neq y\right\}$.
(Refer example 3.4.13) **[8]**

OR

Q.2 a) *Explain CYK algorithm in detail with the help of illustrative example*
(Refer section 3.14) **[10]**

b) *Show that CFL are not closed under intersection and complementation*
(Refer section 3.11) **[8]**

Q.3 a) *Construct PDA for the language* $L = \{a^n b^{2n} \mid n \geq 1\}$. **(Refer example 4.2.2)** **[5]**

b) *Consider the PDA with following moves.*
$\delta(q_0, a, Z_0) = \{(q_0, a, Z_0)\}$
$\delta(q_0, a, a) = \{(q_0, a\ a)\}$
$\delta(q_0, b, a) = \{(q_1\ e)\}$
$\delta(q_1, b, a) = \{(q_1, e)\}$
$\delta(q_0, e, Z_0) = \{(q_1, e)\}$
Obtain CFG equivalent to PDA **(Refer example 4.7.1)** **[12]**

OR

Q.4 a) *Construct PDA that accepts language generated by following CFG :*
$S \to SS \mid (S) \mid (\ )$ **(Refer example 4.7.8)** **[3]**

b) *Prove that - If L is a CFL and R is a regular language then* $L \cap R$ *is a CFL*
(Refer section 4.8) **[8]**

c) *Construct NPDA that accepts the language generated by* $S = S + S \mid S * S \mid 4$.
(Refer example 4.6.1) **[6]**

Q.5 a) *Explain representation of TM* **(Refer section 5.3)** **[4]**

b) *Construct TM for the language* $L = \left\{a^n b^n\right\}$ *where* $n \geq 1$. **(Refer example 5.5.4)** **[8]**

c) *Construct TM for obtaining two's complement of a given binary number.* **(Refer example 5.6.2)** **[6]**

OR

Q.6 a) *Construct a TM for checking the palindrome of a string of odd palindrome for* $\Sigma = \{0,1\}$. **(Refer example 5.5.7)** **[8]**

b) *Explain halting problem of turing machine.* **(Refer section 5.8)** **[6]**

c) *Write short note on*
i) Universal turing machine ii) Multitape turing machine. **(Refer section 5.7)** **[4]**

Q.7 a) *Explain with example decidable and non decidable language.* **(Refer section 6.1)** **[6]**

b) *Obtain the solution for the following system of posts correspondence problem.*
A = {100, 0, 1}, B = {1, 100, 00}. **(Refer example 6.4.3)** **[6]**

c) *What do you mean by polynomial time reduction? Explain with example.* **(Refer section 6.10)** **[5]**

OR

Q.8 a) *Define and explain recursive and recursively enumerable languages.* **(Refer section 6.3)** **[6]**

b) *Prove that following decision problems are recursive i) Two DFA's are equivalent or not ii) NFA accepts a word or not.* **(Refer section 6.1.1)** **[8]**

c) *Write short note on - Church turing thesis.* **(Refer section 6.2)** **[3]**

❑❑❑